Introduction

Computation happens everywhere around us. Computational problems are done on a computation machine. It is necessary for us understand the capabilities and limitations of a computing machine.
The fundamental idea of any given computational problem is to investigate computability and complexity. A computational problem can either be solvable or unsolvable. For a solvable computational problem π the input represented as discrete structures, and by using paradigms of programming (such as functional programming, modular programming, etc.,) and algorithm design (such as dynamic programming, greedy, etc.,) the design for π is done along with its proof of correctness. For π there can be more than one algorithms which can use any data structure for representing data and for each data structure the time complexity defers and by using time complexity we can compare and find the efficient one. Finally, after deciding algorithm for π the implementation of π can be done by using any of C, C++, Python, etc., on a hardware.
We say that a computational problem is solvable when there exists an algorithm and we say a computational problem is unsolvable when there does not exist an algorithm. When we say a computational problem is solvable we assume that there exists an underlying computation machine that solves it. The objective of this course is to study:
What problems can be computed by a computing machine?
What problems cannot be computed by a computing machine?
For any computing machine we need to understand:

- the problems that are solvable Vs unsolvable.
- Limits of computing / Limitations of computing machine such that:
  for an ATM it can dispense cash but not coins, and a calculator can solve $\sin x$, c^x but cannot perform sorting. Further, a computer can perform sorting, image processing and machine learning, etc., but cannot print list of all natural numbers, print all C-programs. Similarly for any computing model, we need to understand the limitations of the machine under consideration.

A small history: Theory of computation or Automata theory or Languages, machines and computation are the study of abstract computing machines. Before there were computers, in 19367, Alan Turing described an abstract machine when he was a doctoral student under Alonzo church. Turings automatic machines, as he termed them in 1936, were specifically devised for the computing of real numbers. They were first named Turing machines by Alonzo Church in a review of Turings paper (Church 1937). Turing machine had all the capabilities of today's computers as far as computing is concerned.
In 1940's and 1950's, simpler kinds of machines, which we call finite automata were studied. The first people to consider the concept of a finite-state machine included a team of biologists, psychologists, mathematicians, engineers and some of the first computer scientists. They all shared a common interest: to model the human thought process, whether in the brain or in a computer. Warren McCulloch and Walter Pitts, two neurophysiologists, were the first to present a description of finite automata in 1943. Their paper, entitled, "A Logical Calculus Immanent in Nervous Activity", made significant contributions to the study of neural network theory, theory of automata, the theory of computation and cybernetics. Later, two computer scientists, G.H. Mealy and E.F. Moore, generalized the theory to much more powerful machines in separate papers, published in 1955-56. The finite-state machines, the Mealy machine and the Moore machine, are named in recognition of their work. While the Mealy machine determines its outputs through the current state and the input, the Moore machine's output is based upon the current state alone.

For any interaction, we need a language which has alphabets and grammar (rules). Consider a human-human interaction in a language say English. Before, we define a language, we shall understand the following:
An *alphabet* (Σ) is a finite, nonempty set of symbols.
Ex: For English language, Σ={A, B,. . ., Z, a, b,. . ., z}.
A *string (word)* (Σ^*) is a finite sequence of symbols chosen from some alphabet.

Ex: For English language, Σ^*={a, an, the, is, India, country, alphabet, course, ...}
A set of strings all of which are chosen from some Σ^*, where Σ is a particular alphabet, is called a *language*.
Note that L need not contain all of Σ^*.
Consider the following statement S and its parse tree:
India is a country.

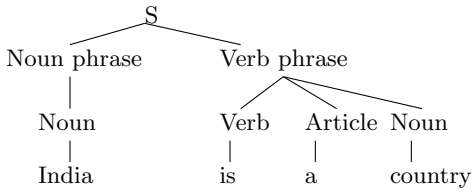


Fig. 1: Parse tree

From the parse tree we can observe that the rules for S in which the verb phrase can be rewritten further where as the words at leaves cannot be rewritten further.
Simliarly, for any human-machine interaction or machine-machine interaction, it has an associated language which has alphabets and grammer (rules). In this course for each of computing models that we look into we shall analyze it's associated language and grammer. We shall look into the following computing models:

1. Finite automaton
2. Pushdown automaton
3. Turing machine
4. Variants of above computing models.

For these computing models, we shall analyse the computing power, grammer and language.

1 Finite State Automaton

We call a machine as finite state machine, when the number of states in the machine is finite and the machine does not have memory. However, by using states the machine can remember the computation performed.
Consider the following Finite State Automaton (FSA) in Figure 2 of an electric switch.
When energy is supplied, FSA goes to ON state and when no energy is supplied then it goes to OFF state.

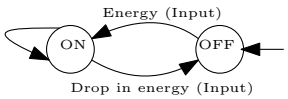


Fig. 2: FSA representing a switch

FSA in context of scheduling can be viewed as in Figure 3.

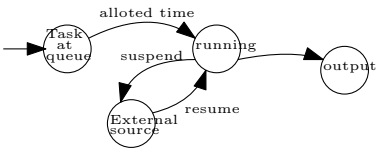


Fig. 3: FSA representing a scheduling

FSA of a binary adder is represented in Figure 4. Note that the representation of FSA in Figure 4 is a FSA with output.

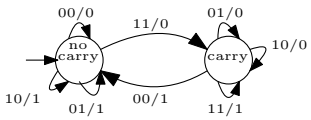


Fig. 4: FSA representing a binary adder

At any time it receives two binary inputs say x_1, x_2. Initially, the machine is in no carry state, and in any time the adder can be in any one of 'carry' or 'no carry' state. In the adder, x_1x_2/x_3 denotes that adder gets input x_1x_2 and goes to the respective state by outputting x_3. On input, at 'no carry' state on input 00 output is 0 and machine stays in 'no carry' state. Similarly, on input, at 'no carry' state on input 01 and 10 output is 0 and machine stays in 'no carry' state, where as on input, at 'no carry' state on input 11, output is 0 and carry is 1, hence machine goes to state 'carry'. At state 'carry', on inputs 01 and 10, output is 0 and carry is 1, hence machine stays in 'carry' state itself. At state 'carry', on inputs 11, output is 1 and carry is 1, hence machine stays in 'carry' state it self. Whereas At state 'carry', on inputs 00, output is 1 and no carry, hence machine goes to 'no carry' state.
We shall define FSA formally as follows:
A Finite State Automation (FSA) is a 5-tuple $M = \{Q,\ \Sigma,\ \delta,\ q_0,\ F\}$, where Q is a finite set of states, Σ is a finite set of input symbols, δ is a transition function such that $\delta : Q \times \Sigma \to Q$, $q_0 \in Q$ is the start state, and $F \subseteq Q$ is set of final states.
Let us analyze the transition function in FSA:

1. $\delta(q, \epsilon) = q'\ q, q' \in Q$. This means that at state q on reading ϵ (reading nothing), it goes to state q'. Note that q' can be q also.
2. $\delta(q, a) = q''$. This means that at state q on reading a, it goes to q''.
3. $\hat{\delta}(q, x)$, where $x \in \Sigma^*$ is a string. Let $x = abba$. Then $\hat{\delta}(q, x) = \delta(\hat{\delta}(q, abba), a)$. Note that we use δ in transition function that takes a state and a symbol and returns a state, whereas $\hat{\delta}$ in transition function that takes a state and a string and returns a state that automaton reaches after processing the sequence of symbols in input string.

Intuitively, δ is a function that tells which state to move to in response to an input. Transition function is the process or algorithm or logic for the given language.
We can also look a FSA like this:

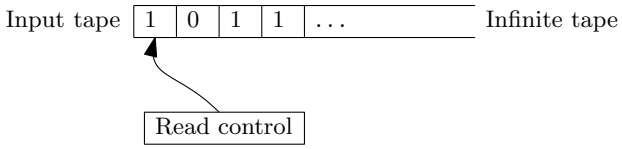


Fig. 5: Pictorial representation of FSA

The input string is placed on the input tape. Each cell contains a symbol. The read head initially points to leftmost symbol the input string. Reading can be done one cell at a time, write operation cannot be done and left move cannot be performed. FSA cannot determine the number of 0's or number of 1's read so far, however, some partial compuation can be remembers through states which we shall explore later.
Consider a FSA A represented in Figure 6.

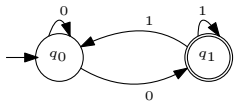


Fig. 6: FSA representing a language $L = \{x \in \Sigma^* \mid \Sigma = \{0, 1\} \wedge x \text{ is ending with } 1\}$

Note that for A, $M = (Q = \{q_0, q_1\},\ \Sigma = \{0, 1\},\ \delta,\ q_0,\ F = \{q_1\})$. Transition function δ is:
$\delta(q_0, 0) = q_0$
$\delta(q_0, 1) = q_1$
$\delta(q_1, 0) = q_0$
$\delta(q_1, 1) = q_1$
Note that transition function δ is a function and can also be represented in a table as in Figure 7.

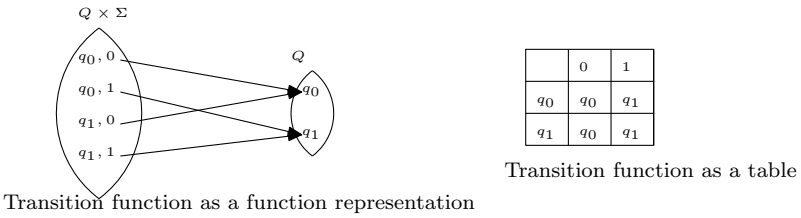


Fig. 7: A transition function representation for A

In a transition function $\forall q \in Q\ \forall a \in \Sigma\ \delta(q, a)$ must be defined. In transition function, each element in codomain has a preimage. Suppose that if an element has no preimage, then it is a good as designing a FSA without that state.

Acceptance of FSA:

- A string x is accepted by FSA, if stating from start state q_0, after reading the input string, if the automaton goes to one of the final states.
- If $\hat{\delta}(q_0, x) = q \in F$, then x is accepted by FSA.

Language for FSA is $L = \{x \mid x \in \Sigma^* \wedge \hat{\delta}(q_0, x) = q,\ q \in F\}$.
For A, language $L = \{x \in \Sigma^* \mid \Sigma = \{0,1\} \wedge x \text{ is ending with } 1\}$. Meaning all strings of type $x = x'1$ is accepted by A whereas all string of type $x = x''0$ is rejected by A, where $x', x'' \in \Sigma^*$. For example, $L = \{1,\ 11,\ 101,\ 10001, \ldots\}$. Observe that any string that does not belong to L is rejected by FSA as represented in Figure 6. For instance in A, $\delta(q_0, 0) = q_0 \notin F$, the string 0 is not accepted by A and $0 \notin L$.
We shall now prove formally that A accepts $L = \{x \in \Sigma^* \mid \Sigma = \{0,1\} \wedge x \text{ is ending with } 1\}$ and rejects $\Sigma^* - L$.

Proof. Suppose x is a string of form $x'0$; that is, 0 is the last symbol of x and x' is the string (over $\{0,1\}^*$) consisting of all but the last symbol. On reading x', A may be at q_0 or at q_1. If A is in q_0 after reading x' and on reading 0, A stays in q_0 which is not a final state and hence string of the form $x'0$ is rejected. If A is in q_1 after reading x' and on reading 0, A goes to q_0 which is not a final state and hence string of the form $x'0$ is rejected.
Suppose x is a string of form $x'1$; that is, 1 is the last symbol of x and x' is the string consisting of all but the last symbol. On reading x', A may be at q_0 or at q_1. If A is in q_0 after reading x' and on reading 1, A goes to q_1 which is a final state and hence string of the form $x'1$ is accepted by A. If A is in q_1 after reading x' and on reading 1, A stays in q_1 which is a final state and hence string of the form $x'1$ is accepted.
That is A accepts strings over $\{0,1\}$ that ends with 1. □

Now having constructed FSA A for L, can we construct any other FSA accepting $L = \{x \in \Sigma^* \mid \Sigma = \{0,1\} \wedge x \text{ is ending with } 1\}$?
Consider the two finite automatons B and C in Figure 8. Observe that B at q_0 on reading 1 can either stay

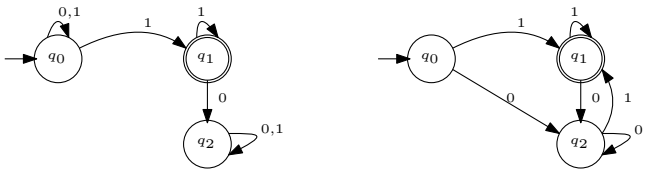
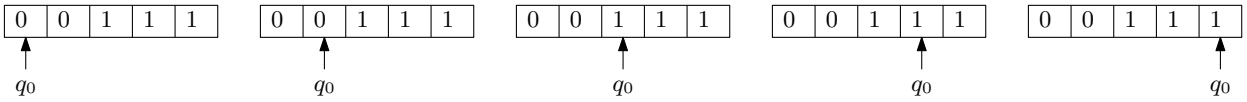


Fig. 8: FSA representing a language $L = \{x \in \Sigma^* \mid \Sigma = \{0,1\} \wedge x \text{ is ending with } 1\}$

at q_0 or go to q_1, meaning $\delta(q_0, 1) = \{q_0, q_1\}$, this brings non-determinism to FSA. A non-deterministic finite state automaton accepts input string if there exist a path (parsing) that leads to an accepting state. For example, on input 00111, B at least two ways of parsing as represented in Figure 9 and parse 2 leads to a final state. Hence the string 00111 is accepted by B.

Parse 1:



Parse 2:

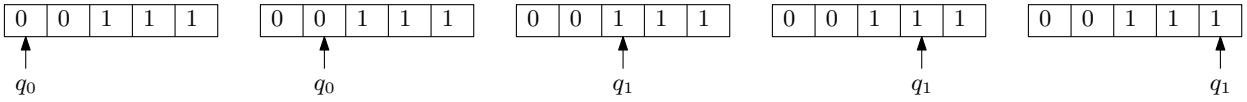


Fig. 9: Parsing of string 00111 by FSA B

Remarks:

– when FSA M as represented in Figure 10 is such that it has no final state $F = \emptyset$, then $L(M) = \emptyset$

Fig. 10: A FSA accepting nothing.

– when FSA M as represented in Figure 11 is such that start state is also one of the final state, then $\epsilon \in L(M)$ i.e. M accepts empty string. The language accepted by M in Figure 11 is $L(M) = \{\epsilon\}$.

Fig. 11: FSA

– Consider FSAs in Figure 12, M_1 is such that it accepts set of all strings over a^*, $L(M_1) = \{\epsilon,\ a,\ aa,\ aaa, \ldots\}$. Consider FSA M_2, it accepts any string over $\{a, b\}^*$, $L(M_2) = \{\epsilon,\ a,\ b,\ ab,\ aa,\ ba,\ bb, \ldots\}$. Thus $L(M_2) = \Sigma^*$, where $\Sigma = \{a, b\}$. Consider FSA M_3, it accepts any string over $\{c, d\}^*$ and rejects any string that contains a or b. Thus $L(M_3) \subseteq \Sigma^*$, where $\Sigma = \{a, b, c, d\}$.

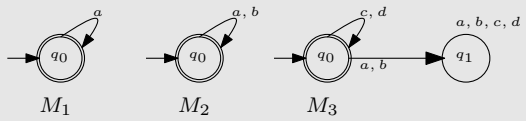


Fig. 12: FSA

– Strings accepted by a FSA is infinite if there exist a loop involving a final state. Consider FSAs in Figure 13, $L(M_1) = a$. Observe that the loop in M_1 is not related to final state, hence the language accepted by M_1 is finite. For M_2, the language accepted is infinite and there exist a loop containing one of the final states.

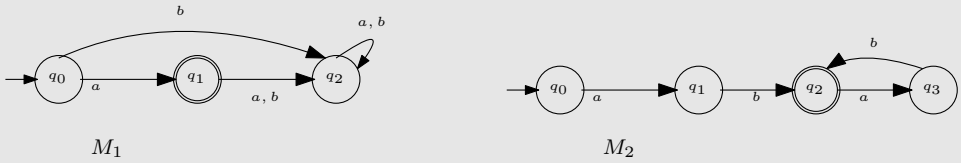


Fig. 13: FSA

Non-Deterministic Finite State Automata

1. Design an NFA which accepts the set of all strings over $\{0, 1\}$ such that ends with '011'.
   Let L be the language, $L = \{x | x \in \{0, 1\}^*$ and x ends with $'011'\}$
   Some of the strings accepted by the language L are
   $L = \{011, 0011, 1011, 00011, 01011, \ldots\}$. Note that strings accepted by L can have any strings over $\{0, 1\}$ as a prefix of '011". Consider the transition diagram given in Figure 14. The state q_0 handles the prefix part (any strings over $\{0, 1\}$). Note that $\delta(q_0, 0) = \{q_0, q_1\}$. This shows that on reading 0, q_0 can go to two states q_0 and q_1. Therefore the machine is a non-deterministic machine. Suppose the state q_3 reads 0 or 1 it goes to the state q_4 which is not a final state. Since this is an NFA, strings can be parsed in more than one way. Consider the string 0000111.
   $P_1 = q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{1} q_3$ - Accepted
   $P_2 = q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0$ - Rejected
   Note that P_1 accepted the string whereas P_2 rejected the string. In such case if we find at least one parsing which is moving from q_0(Start state) to q_3(Final state), then we say the string is accepted. For a language one can construct more than one NFA. Another NFA is given in Figure 15.
   Note: For each symbol in $\sum$ the transition must be well defined. The transition function $\delta : Q \times \sum \to Q$ says on reading a symbol it should go to exactly one state $q' \in Q$. We know that NFA can go to more than one state so the transition function is not a function it is just a mapping. In the context of NFA we modify the transition function as follows
   Transition function for NFA $\delta : Q \times \sum \to 2^Q$(Power set of Q)

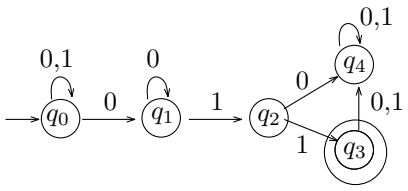


Fig. 14: *NFA 1*

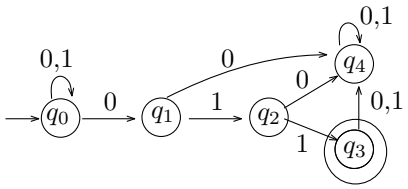


Fig. 15: *NFA-2*

$\delta(q_0, 0) = \{q_0\}$.
$\delta(q_1, 0) = \{q_0, q_1\}$.
Here $\{q_0, q_1\}$ is considered as a single state which is state in 2^Q.
Given a language there exist many DFAs that accept the given language. However, given a DFA, there is only a unique language that the DFA accepts.

2. Design a DFA which accepts the set of all strings over $\{0, 1\}$ such that ends with '011'.
   Consider the base string which is '011'. We start our construction that accepts the base string. Here at least four states are needed to accept the base string. So the minimum number of states needed is at least four and less than four is not possible. When the machine is at q_2 it ensures that it has read '011'. The minimum state DFA is given in Figure 16. Figure 17 shows other possible DFA for the language.

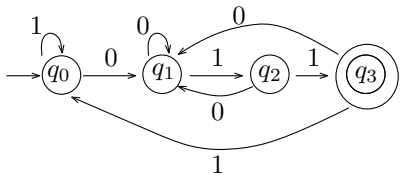


Fig. 16: *DFA 1*

   Note: To construct minimum DFA:
   - Focus on base string and construct automaton.
   - Fill other entries.

3. Construct an NFA that accept all strings containing '101 'as a substring.
   $L = \{x | x \in \{0, 1\}^*$ such that x contains '101' as a substring $\}$
   $L = \{101, 01010, 1011, 1101, 01011, 101101101, \ldots\}$.
   The prefix and suffix of '101' can be any string over $\{0, 1\}$. We construct a base NFA (Figure 18) that accepts the base string '101' with four states. Prefix and suffix of '101' are taken care by the states q_0 and q_3, respectively. The automaton shown in Figure 19 is an another NFA that accepts the language L.

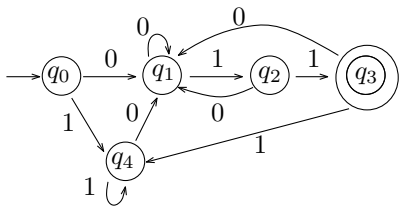


Fig. 17: *DFA 2*

Note: We observe that for pattern matching(substring matching), FA (DFA and NFA) exists.

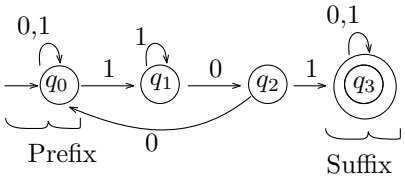


Fig. 18: *NFA 1*

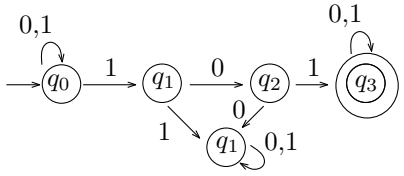


Fig. 19: *NFA 2*

Food for thoughts:
- Is it possible to construct DFA for every NFA?
- Are there a language such that NFA exists but DFA does not exist?

4. Design a DFA that accepts the strings containing 101 as a substring.
   $L = \{x \mid x \in \{0,1\} \mid x \text{ contains 101 as a substring}\}$.

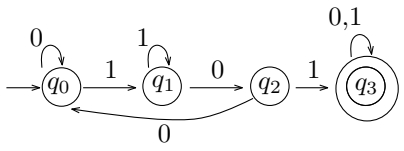


Fig. 20: *DFA 1*

For any DFA to accept the strings containing 101 as a substring, it needs at least 4 states. In this DFA (Figure 20), we have exactly 4 states, hence the constructed DFA is a minimum DFA. Let us take a closer look at the DFA constructed, the state q_1 ensures that at least one 1 is read and at state q_2, the automata ensure that the input string has 10 as a substring, whereas at state q_3 the automata ensures that at least one 101 is read in the input string. Note that the input string can have any number of 0's in the prefix which is handled by state q_0 and similarly any number of 1's before 01 is possible which is ensured at q_1 by a self-loop. At q_2, if a 0 is read then it has 100 hence the machine goes back to q_0 in order to ensure 101. If the machine is in q_3, then the string has 101 as a substring and hence suffix of 101 can be any string over $\{0,1\}^*$, and hence there is a loop at q_3 for input 0 or 1.

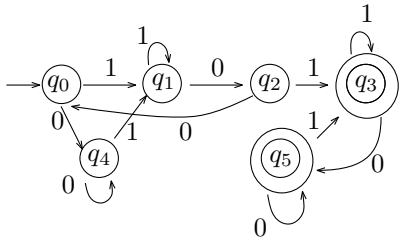


Fig. 21: *DFA 2*

We modify the DFA-1 (Figure 20) and we get another DFA with more number states. From the state q_3, on reading 0 it moves to q_5 which is also a final state. Note: For any automaton, the start state is unique but the final states need not be unique. The DFA shown in Figure 21 has two final states.

5. Design a DFA that accepts the strings does not containing 101 as a substring.
   $L1 = \{x \mid x \in \{0,1\} \mid x \text{ does not contains 101 as a substring}\}$.
   In the previous case study, we have constructed a DFA for the language L that accepts all the strings that contain '101' as a substring. Here, L_1 accepts all those strings that are rejected by L ie, $L_1 = \sum^* - L$. Consider the DFA (Figure 20), after exhausting all the symbols if the machine is at q_0 or q_1 or q_2, then they have the property that none of them have '101' as a substring. To construct our required DFA, we just flip the non-final state to the final state and the final state to non-final state.
   Note: Consider the language $L \subseteq \sum^*$ such that L respects some property. Let that property be pattern P and it is finite. For example, P occurs at the beginning or at the end or P is a substring. for a complementary question (Not containing P [$\sum^*$-containing P]), we construct a FA containing pattern P and flip the final state to non-final state and vice-versa. The infinite pattern is not well defined. Suppose if P is infinite, FA might accept invalid stings.

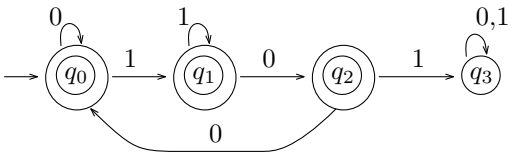


Fig. 22: *CE*

Can FA accept $L1 = \{x \mid x \in \sum^*, x$ has property(pattern) P ,$P \in \sum^*\}$. Interestingly, the answer is YES. Having explored pattern matching, we shall the other computing power of FA. Now we shall see some computation related to 'counting'.

6. Design a DFA that accepts even number of $a's$.
   $L = \{x \mid x \in \{a, b\} \mid x$ has even number of 'a's$\}$.
   $L = \{\epsilon, b, bbb, aabbb, abababa, \ldots\}$.
   Before attempting the above question, we shall see the simpler version of the question.
   $L_1 = \{x \mid x \in \{a, b\} \mid x$ has exactly 2 'a's$\}$.

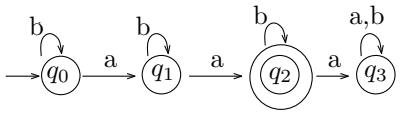


Fig. 23: *DFA that accepts strings has exactly 2 'a's*

At any instant of time, if the machine (Figure 24) is at q_0, then the strings have an even number of $a's$. Similarly, if the machine is at q_1, then strings have an odd number of $a's$.

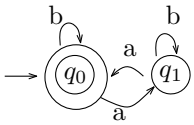


Fig. 24: *DFA that accepts strings has even number 'a's*

$L_2 = \{x \mid x \in \{a, b\} \mid x$ has odd number of 'a's$\}$.
At any point in time, if the machine (Figure 25) is at q_0, then the strings have an even number of $a's$. Similarly, if the machine is at q_1, then strings have an odd number of $a's$.

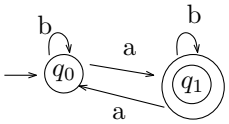


Fig. 25: *DFA that accept strings has odd number 'a's*

7. Design a DFA that accepts even number of $a's$ and even number of $b's$.
   $L = \{x \mid x \in \{a, b\} \mid x$ has even number of 'a's and even number of 'b's$\}$.
   $L = \{bb, bbbb, aabb, abababab, \ldots\}$.
   We have designed DFA for the following languages
   $L_1 = \{x \mid x \in \{a, b\} \mid x$ has even number of 'a's$\}$. (Figure 26)
   $L_2 = \{x \mid x \in \{a, b\} \mid x$ has even number of 'b's$\}$. (Figure 27)
   We observe that $L = L_1 \cap L_2$. Note that if a state has a self-loop on a specific symbol, then we do not

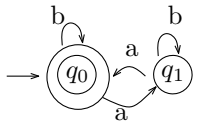


Fig. 26: *DFA that accept strings has even number 'a's*

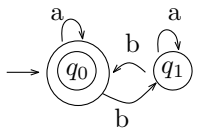


Fig. 27: *DFA that accept strings has even number 'b's*

have control over that specific symbol. If we want to have a specific count of a symbols, then we should not have a self-loop on that symbol in any state. We now design a DFA that accepts the language L. When the machine is at q_2, we see that the machine has read an odd number of $a's$ and odd number of $b's$. We force the machine to read one more a and one more b to reach the final state. In this way, even number of $a's$ or an odd number of $b's$ are ensured. Similarly, when the machine is at q_3, we see that the machine read odd number of $b's$ and even number of $a's$. In this case, we fore machine to read 2 $a's$ or one b. A similar argument can be given for the state q_1.
We slightly modify the above DFA (Figure 28) to design for the following
$L_3 = \{x \mid x \in \{a, b\} \mid x$ has even number of 'b's and odd number of 'a's$\}$ (Figure 29).
$L_4 = \{x \mid x \in \{a, b\} \mid x$ has odd number of 'b's and odd number of 'a's$\}$ (Figure 30).
$L_5 = \{x \mid x \in \{a, b\} \mid x$ has even number of 'a's and odd number of 'b's$\}$ (Figure 31).

FA framework for Union, Intersection, and Minus
Let $L_1 = \{x \mid x \in \{a, b\}^* \mid x$ ends with 'ab'$\}$
$L_2 = \{x \mid x \in \{a, b\}^* \mid x$ does not contain 'aa'as a substring$\}$
We know that L_1 and L_2 are sets that accepts set of strings over $\sum$. Now it is natural to ask, can we apply some set theoretic operations like union, intersection, and set minus to obtain a new language. Interestingly, the answer is YES. In this section we shall see how to obtain a new language by applying set operations. We shall design FA that accepts
$L_3 = L_1 \cup L_2 = \{x \mid x \in \{a, b\}^* \mid x$ ends with 'ab' or does not contain 'aa'as a substring$\}$
$L_4 = L_1 \cap L_2 = \{x \mid x \in \{a, b\}^* \mid x$ ends with 'ab' and does not contain 'aa'as a substring$\}$
$L_5 = L_1 - L_2 == L_1 \cap L_2^c = \{x \mid x \in \{a, b\}^* \mid x$ ends with 'ab' and not (x does not contain 'aa'as a substring)$\}$
$L_6 = L_1^c = \{x \mid x \in \{a, b\}^* \mid x$ not ending with 'ab'$\}$
$L_7 = L_2^c = \{x \mid x \in \{a, b\}^* \mid x$ containing 'aa' as a substring$\}$

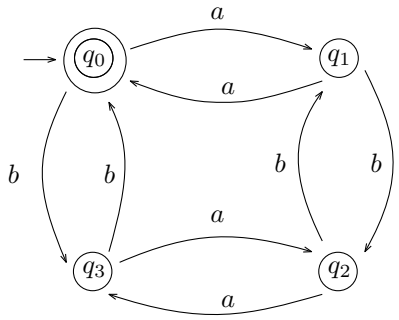


Fig. 28: $L = \{x \mid x \in \{a, b\} \mid x$ *has even number of 'a's and even number of 'b's*$\}$

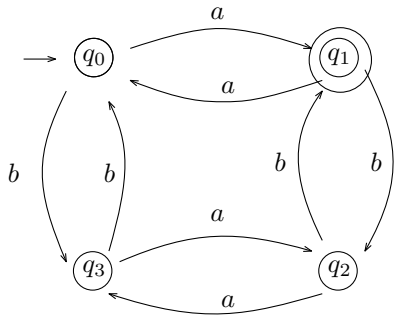


Fig. 29: $L_3 = \{x \mid x \in \{a, b\} \mid x$ *has even number of 'b's and odd number of 'a's*$\}$

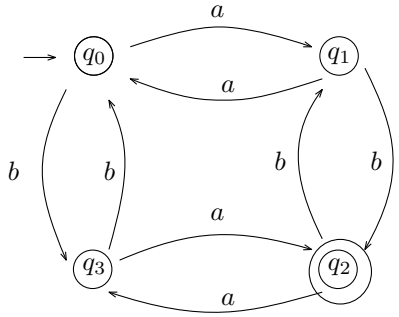


Fig. 30: $L_4 = \{x \mid x \in \{a, b\} \mid x$ *has odd number of 'b's and odd number of 'a's*$\}$

We next develop a framework which will be of use in answering L_3 to L_5. We first design a DFA for L_1 and L_2 separately and then we present a framework to design a DFA for the Languages L_3 to L_5. To obtain $L_3 = L_1 \cup L_2$, we simultaneously parse the string in L_3 if any one of L_1 or L_2 accept the string then L_3 is accepted. Similarly, for $L_4 = L_1 \cap L_2$, we simultaneously parse the string in L_3 and both L_1 and L_2 must be accepted. For $L_5 = L_1 - L_2$, L_1 must be accepted and L_2 must be rejected. Consider the input symbol x is $'ab'$. We shall simulate $'ab'$ with respect to DFA shown in Figure 32 and Figure 34. Initial state of FSA-1 is P and FSA-2 is A.
$PA \xrightarrow{a} QB \xrightarrow{b} RA$. When we parse the string $'ab'$, we see that both FSA-1 and FSA-2 reaches final state. If the language is L_3 or L_4 we say that the string $'ab'$ is accepted. Note: The framework that we are going to develop will have states and those labels are precisely the elements of the cross

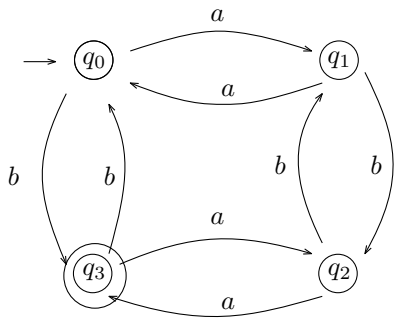


Fig. 31: $L_5 = \{x \mid x \in \{a, b\} \mid x$ *has even number of 'a's and odd number of 'b's*$\}$.

product. For example, here FSA-1 has $\{P, Q, R\}$ and FSA-2 has $\{A, B, C\}$. $\{P, Q, R\} \times \{A, B, C\} = \{PA, PB, PC, QA, QB, QC, RA, RB, RC\}$. In the developed framework the state will be the start state of FSA-1 and FSA-2 which is RA. After exhausting x, if machine is in a state where (i) at least one of them is a final state ($L_3 = L_1 \cup L_2$) (ii) both of them are a final state ($L_4 = L_1 \cap L_2$) (iii) one is final state of L_2 and other is non final state of L_2 ($L_5 = L_1 - L_2$).

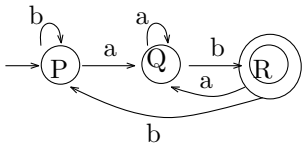


Fig. 32: *FSA-1 for* L_1

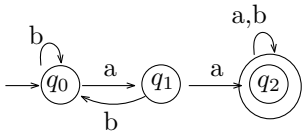


Fig. 33: *FSA for containing 'aa' as a substring*

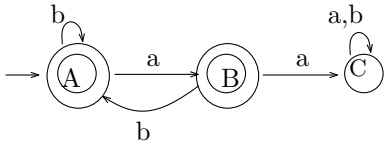


Fig. 34: *FSA-2 for* L_2

We now design a FA framework for union, intersection, and minus. We shall first list all the states in our framework. For each element in the cross product, we have a state. The start state of the framework is the start state of L_1 and L_2. Now we define a transition as follows
$\delta(PA, a) = \delta(P, a) \cup \delta(A, a)$
$= Q \cup B$
$= QB$
. On a similar line, we define transition for all the states in FA framework.
i) Final states of $L_3 = L_1 \cup L_2$ area $\{RA, RB, RC, PA, QA, PB, QB\}$
ii) Final states of $L_4 = L_1 \cap L_2$ are $\{RA, RB\}$
iii) Final state of $L_5 = L_1 - L_2$ is $\{RC\}$
iv) Final states of $L_6 = L_2 - L_1$ are $\{AP, AQ, BP, BQ\}$

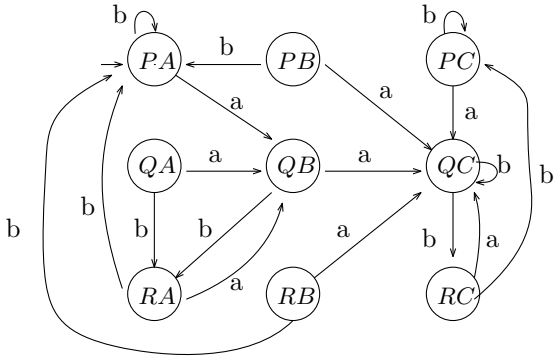


Fig. 35: *FA Framework for Union, Intersection and Minus*

We now construct FSA for $L_4 = L_1 \cap L_2$ from the FA framework given above. In the FA framework, there are some states that cannot be reached from the start state. So we remove such states. since the states PB, QA, RB are not reachable from the start state, we remove such states. Now it has only one final state which is RA. The modified FA framework (see Figure 36) is our required FSA for $L_4 = L_1 \cap L_2$. Having constructed FSA, a natural question is that, is this a minimum state FSA for $L_4 = L_1 \cap L_2$? The answer is NO. Here the states PC and RC are redundant. Hence this not a minimum DFA for $L_4 = L_1 \cap L_2$. The minimum DFA is shown in Figure 37 with four states.

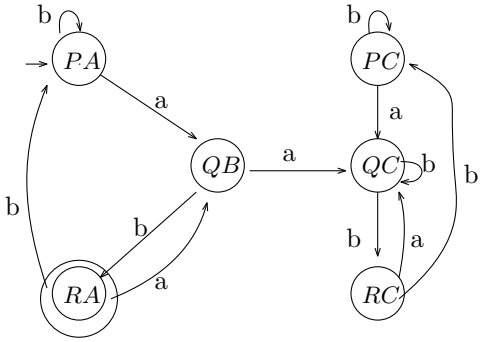


Fig. 36: *DFA for* $L_4 = L_1 \cap L_2$

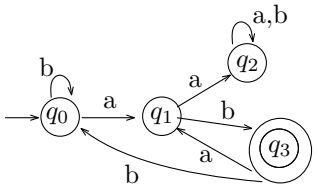


Fig. 37: *Minimum DFA for* $L_4 = L_1 \cap L_2$

Note: If both the machines are deterministic, the framework will give a deterministic machine. The framework certainly gives an FSA. The FSA constructed from the framework need not be minimum. This framework can be used when it is difficult to construct an FSA for $L_1 \cup L_2$ or $L_1 \cap L_2$ and so on.

2 Equivalence of DFA and NFA

We have seen that given a language we have constructed a DFA and NFA. Natural question that may arise is that Does there exists a DFA for every NFA? or are there languages for which NFA exist but not DFA? We know that every DFA is a NFA, but it is interesting fact that every language that can be described by some NFA can also be described by some DFA. We show that for every NFA there exists an equivalent DFA by using subset construction method. We shall look into some case studies:
Consider this language $L = \{x \mid x \in \{0,1\}^*,\ x \text{ has 011 as a substring}\}$

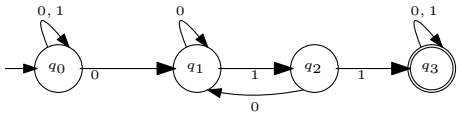


Fig. 38: NFA

On reading an input symbol from a state, NFA goes to subset of states instead of a state as in DFA. In this case, $\delta(q_0, 0) = \{q_0, q_1\}$. In general, $\delta(q, a) = \{q_0, \ldots, q_k\}$ where $\{q_0, \ldots, q_k\} \subseteq Q$. When we construct a corresponding DFA from a NFA as follows:

- The start state in the DFA corresponds to the start state of the NFA.
- Each state in the DFA is associated with a set of states in the NFA.
- If a state q in the DFA corresponds to a set of states S in the NFA, then the transition from state q on a character a is found as follows:
  - Let S' be the set of states in the NFA that can be reached by following a transition labeled a from any of the states in S.
  - The state q in the DFA transitions on a to a DFA state corresponding to the set of states S'. i.e $\delta(\{q_0, \ldots, q_k\}, a) = \delta(q_0, a) \cup \ldots \cup \delta(q_k, a)$

Note that the input alphabets of the two automata are the same.
Now we shall construct a corresponding DFA for NFA defined for language $L = \{x \mid x \in \{0,1\}^*,\ x \text{ has 011 as a substring}\}$.
We know that start state is $\{q_0\}$.
For $\delta(q_0, 0) = \{q_0, q_1\}$ and $\delta(q_0, 1) = q_0$, we have state $\{q_0, q_1\}$ in DFA.
For $\{q_0, q_1\}$, we find $\delta(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \{q_1\} = \{q_0, q_1\}$, and $\delta(\{q_0, q_1\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0, q_2\}$.

Now we have a state $\{q_0, q_2\}$, $\delta(\{q_0, q_2\}, 0) = \delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0, q_1\}$, and $\delta(\{q_0, q_2\}, 1) = \delta(q_0, 1) \cup \delta(q_2, 1) = \{q_0, q_3\}$.
Now we have a state $\{q_0, q_3\}$, $\delta(\{q_0, q_3\}, 0) = \delta(q_0, 0) \cup \delta(q_3, 0) = \{q_0, q_1, q_3\}$, $\delta(\{q_0, q_3\}, 1) = \delta(q_0, 1) \cup \delta(q_3, 1) = \{q_0, q_3\}$.
Now we have a state $\{q_0, q_1, q_3\}$, $\delta(\{q_0, q_1, q_3\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_3, 0) = \{q_0, q_1, q_3\}$, $\delta(\{q_0, q_1, q_3\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_3, 1) = \{q_0, q_2, q_3\}$.
For state $\{q_0, q_2, q_3\}$, $\delta(\{q_0, q_2, q_3\}, 0) = \delta(q_0, 0) \cup \delta(q_2, 0) \cup \delta(q_3, 0) = \{q_0, q_1, q_3\}$, $\delta(\{q_0, q_2, q_3\}, 1) = \delta(q_0, 1) \cup \delta(q_2, 1) \cup \delta(q_3, 1) = \{q_0, q_3\}$.

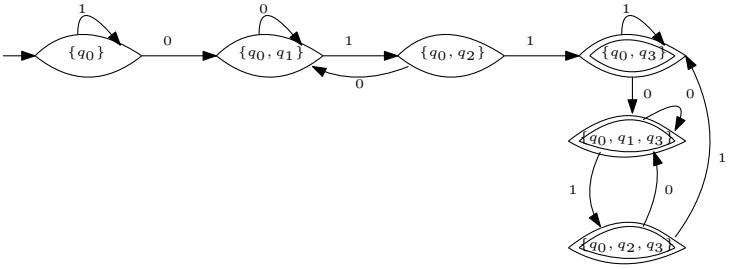


Fig. 39: DFA

The automata obtained from subset construction method is a DFA and it need not be minimum DFA meaning DFA having minimum number of states. For example the language $L = \{x \mid x \in \{0,1\}^*,\ x \text{ has 011 as a substring}\}$ the DFA constructed above has 6 states, and the states $\{q_0, q_1, q_3\}$ and $\{q_0, q_2, q_3\}$ are redundant because at the state $\{q_0, q_3\}$ the automaton has already read 011. Hence on reading a 0 at $\{q_0, q_3\}$, the machine can stay at $\{q_0, q_3\}$ itself.
Consider the language $L = \{a^n b^m \mid n, m \geq 1\}$. Form the language we can interpret that any FSA we construct should accept the strings that contain at least one a and at least one b and b should not be followed by a a. Observe that the strings in the language has no relation with number of a's and number of b's.

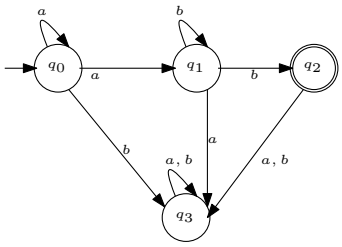


Fig. 40: NFA

By subset construction method, we know that start state of DFA is same as the start state of NFA. We shall define the transition function:

$\delta(q_0, a) = \{q_0, q_1\}$
$\delta(q_0, b) = \{q_3\}$
$\delta(\{q_0, q_1\}, a) = \{q_0, q_1, q_3\}$
$\delta(\{q_0, q_1\}, a) = \{q_1, q_2, q_3\}$
$\delta(\{q_0, q_1, q_3\}, a) = \{q_0, q_1, q_3\}$
$\delta(\{q_0, q_1, q_3\}, b) = \{q_1, q_2, q_3\}$
$\delta(\{q_1, q_2, q_3\}, a) = \{q_3\}$
$\delta(\{q_1, q_2, q_3\}, b) = \{q_1, q_2, q_3\}$
$\delta(\{q_3\}, a) = \{q_3\}$
$\delta(\{q_3\}, b) = \{q_3\}$

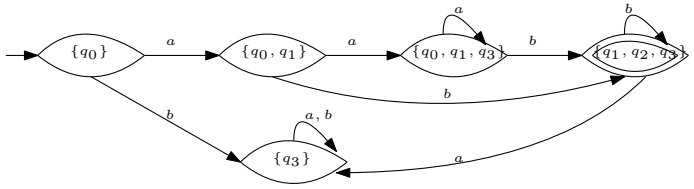


Fig. 41: DFA

Given a NFA having n states in the process of listing the subsets, DFA can have 2^n states. Since a n states NFA is transformed into 2^n state DFA, conversion is not polynomial-time algorithm. Note that we can transform any NFA to DFA using subset construction method. Hence, class of languages accepted NFA is same as the class of language accepted by DFA.

Power of Non-determinism:

There are many case studies where constructing a NFA is straight forward compared to constructing DFA. Consider the following case studies, where NFA has much simpler structure than DFA:

1. $L_1 = \{x \mid x \in \{0,1\}^*$ such that tenth symbol from the right is $1\}$.

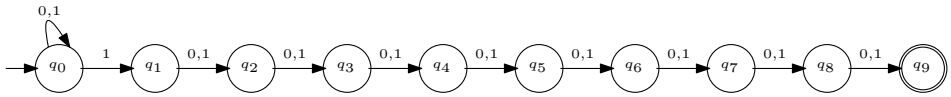


Fig. 42: NFA

2. $L_2 = \{x \mid x \in \{0,1\}^*$ such that there are two 0's in x that are separated by number of symbols that is a multiple of $4\}$.

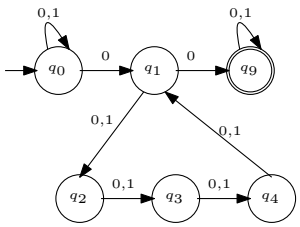


Fig. 43: NFA

3. $L_3 = \{x \mid x \in \{0,1,2,3\}^*$ such that the final digit in the string must be appeared before in that string$\}$.

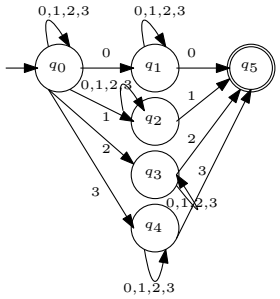


Fig. 44: NFA

3 NFA with epsilon-transition:

An NFA is allowed to make a transition without receiving an input symbol. We allow the automaton to change state without reading any symbol. This is represented by an ϵ-transition.
In the Figure 45, 0 can be interpreted as $\epsilon 0 = 0\epsilon = \epsilon\epsilon\epsilon 0 = 0\epsilon\epsilon\epsilon = \epsilon\epsilon 0\epsilon\epsilon$.

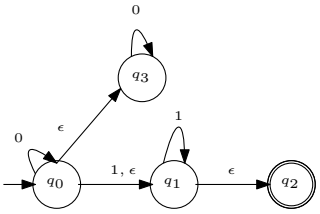


Fig. 45: ϵ-NFA

A *NFA with ϵ-transition* is a five tuple $N = (Q, \Sigma, \delta, q_0, F)$, where $Q,\ \Sigma,\ q_0,\ F$ are same as in NFA and δ is a mapping from $Q \times (\Sigma \cup \{\epsilon\})$ into finite subset of Q.

3.1 Epsilon-closure

ϵ-closure of a state q is the set of states which can be reached from q by reading ϵ only. The ϵ-closure of a state includes itself.
We shall define ϵ-NFA formally as follows:
$M = (Q, \Sigma, \delta, q_0, F)$ were Q is set of states, Σ is set of input alphabets, q_0 is the start state and F is the set of final states, and δ is a function from $Q \times (\Sigma \cup \epsilon)$ to set of all subset of states.

3.2 Converting a NFA with ϵ to a NFA without ϵ

Let us eliminate ϵ-transitions from the ϵ-NFA, we construct a NFA as follows by considering the following example:

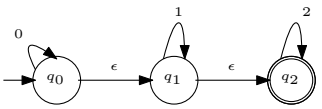


Fig. 46: ϵ-NFA

Step 1: For each state find the state that are reachable (Find epsilon closure for each state).
$E(q_0) = \{q_0, q_1, q_2\}$, $E(q_1) = \{q_1, q_2\}$, $E(q_2) = \{q_2\}$.
Step 2: Compute $\hat{\delta}(q, a)$ for each $q \in Q$ and for each $a \in \Sigma$.
$\hat{\delta}(q_0, 0) = E(\hat{\delta}(E(q_0, 0))) = E(\hat{\delta}(\{q_0, q_1, q_2\}, 0)) = E(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)) = E(q_0 \cup \emptyset \cup \emptyset) = E(q_0) = \{q_0, q_1, q_2\}$.
$\hat{\delta}(q_0, 1) = E(\hat{\delta}(E(q_0), 1)) = E(\hat{\delta}(\{q_0, q_1, q_2\}, 1)) = E(q_1) = \{q_1, q_2\}$.
$\hat{\delta}(q_0, 2) = E(\hat{\delta}(E(q_0), 2)) = E(\hat{\delta}(\{q_0, q_1, q_2\}, 2)) = E(q_2) = \{q_2\}$.
$\hat{\delta}(q_1, 0) = E(\hat{\delta}(E(q_1, 0))) = E(\hat{\delta}(\{q_1, q_2\}, 0)) = E(\emptyset) = \emptyset$.
$\hat{\delta}(q_1, 1) = E(\hat{\delta}(E(q_1), 1)) = E(\hat{\delta}(\{q_1, q_2\}, 1)) = E(q_1) = \{q_1, q_2\}$.
$\hat{\delta}(q_1, 2) = E(\hat{\delta}(E(q_1), 2)) = E(\hat{\delta}(\{q_1, q_2\}, 2)) = \{q_2\}$.
$\hat{\delta}(q_2, 0) = E(\hat{\delta}(E(q_2, 0))) = E(\hat{\delta}(\{q_2\}, 0)) = E(\emptyset) = \emptyset$.
$\hat{\delta}(q_2, 1) = E(\hat{\delta}(E(q_2), 1)) = E(\emptyset) = \emptyset$.
$\hat{\delta}(q_2, 2) = E(\hat{\delta}(E(q_2), 2)) = E(\hat{\delta}(\{q_2\}, 2)) = \{q_2\}$.
From this the obtained NFA is

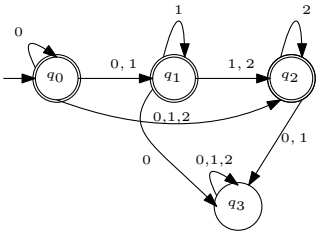


Fig. 47: NFA

The final states are the states whose epsilon closure in ϵ-NFA contains at least one final state. In this example, $E(q_0)$, $E(q_1)$, $E(q_2)$ contains final state. Hence, q_0, q_1, q_2 are final states.

3.3 Converting a ϵ-NFA to a DFA

We can also convert ϵ-NFA to DFA by using the following subset construction approach:

1. The start state of DFA is $E(q_0)$.
   We shall consider ϵ-NFA in Figure 46,
   $E(\delta(q_0, q_1, q_2), 0) = E(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)) = E(q_0) = \{q_0, q_1, q_2\}$
   $E(\delta(q_0, q_1, q_2), 1) = E(q_1) = \{q_1, q_2\}$
   $E(\delta(q_0, q_1, q_2), 2) = E(q_2) = \{q_2\}$
2. For each subset obtained, find the transitions on $0, 1, 2$.
   $E(\delta(q_1, q_2), 0) = \emptyset$
   $E(\delta(q_1, q_2), 1) = \{q_1, q_2\}$

$E(\delta(q_1, q_2), 2) = \{q_2\}$.
$E(\delta(q_2), 0) = \emptyset$
$E(\delta(q_2), 1) = \emptyset$
$E(\delta(q_2), 2) = \{q_2\}$.

3. The DFA obtained is

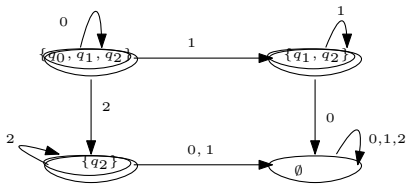


Fig. 48: DFA

The final states are the states that contains q_2.

4 Regular Expressions

Regular expression is an expression using which one can express languages accepted by any finite state automaton.
Regular expression denotes the languages. The operators using in regular expression are concatenation, or, Kleene closure (star closure).

- The *concatenation* of languages L and M is denoted as $L.M$, it is the set of strings that can be formed by taking a string in L and concatenating it with any string in M.
- The *or* of languages L and M is denoted as $L + M$, is the set of strings that are in either L or M.
- The *closure (or star, or kleene closure)* of a language L is denoted as L^*, it represents the set of those strings that can be formed by taking any number of strings from L, possible with repetitions and concatenating all of them.

Let us look into some examples for each of the operations with $\Sigma = \{0, 1\}$:
Concatenation: Let $L = 001$ and $M = 101$, $L.M = 001101$.
Or: $0 + 1$ is either 0 or 1.
Closure: $0^* = \epsilon,\ 0,\ 00, \ldots$
$0 + 1$ possible strings are $0, 1$
$(0 + 1)^*$ possible strings are $\epsilon,\ (0+1),\ (0+1)^2,\ (0+1)^3, \ldots,\ (0+1)^k, \ldots$, where $(0+1)$ is all possible strings of size one such as $0, 1$, $(0 + 1)^2$ is all possible strings of size two such as $00, 01, 10, 11, \ldots$, $(0 + 1)^k$ is all possible strings of size k.
We shall write the regular expressions for the case studies which we have seen:

1. $L_1 = \{x \mid x \in \{0, 1\}^*,\ x$ contains 110 as a substring$\}$.
   Regular expression for L_1 is $(0 + 1)^*110(0 + 1)^*$
2. $L_2 = \{x \mid x \in \{0, 1\}^*,\ x$ begins with 01 and ends with 11$\}$.
   Regular expression for L_1 is $01(0 + 1)^*11$

4.1 Converting Regular expression to NFA

We shall define some framework for constructing NFA for a regular expression

1. Regular expression is 0. The corresponding NFA is in Figure 49.

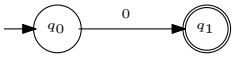


Fig. 49: NFA

2. Regular expression is $0 + 1$. The corresponding NFA is in Figure 50.

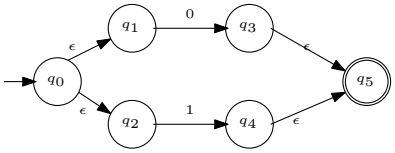


Fig. 50: NFA

3. Regular expression is 00. The corresponding NFA is in Figure 51.

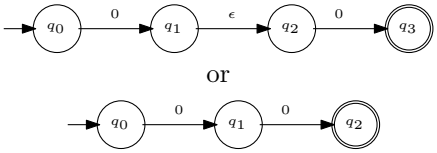


Fig. 51: NFA for regular expression 00

4. Regular expression is 0^*. The corresponding NFA is in Figure 52.

Fig. 52: NFA for regular expression 0^*

5. Regular expression is $(0 + 1)^*$. The corresponding NFA is in Figure 53.

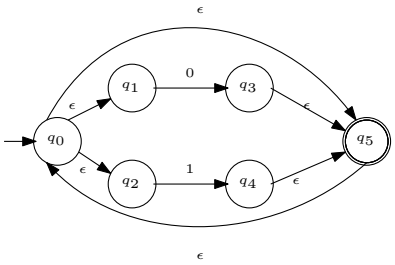


Fig. 53: NFA for regular expression $(0+1)^*$

6. Regular expression is $(0+1)^*110(0+1)^*$. The corresponding NFA is in Figure 54.

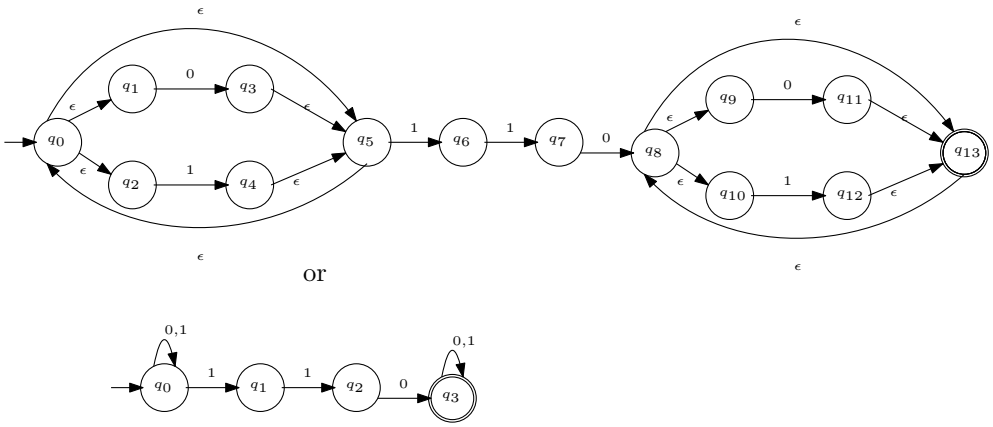
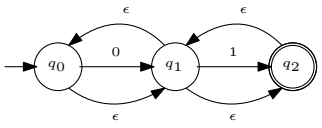
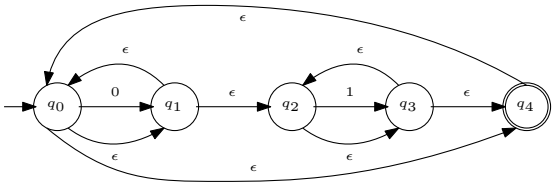


Fig. 54: NFA for regular expression $(0+1)^*110(0+1)^*$

7. Regular expression is $(0^*1^*)^*$. The corresponding NFA is in Figure 55. NFA N_1 accepts 0101 but it is not in $(0^*1^*)^*$, where as N_2 rejects all invalid strings.



NFA N_1



NFA N_2

Fig. 55: NFA for regular expression $(0^*1^*)^*$.

Regular expression represents *regular language*. Regular languages are the languages accepted by finite automata. We shall see examples of regular language and its corresponding regular expression.

1. Consider the regular language L_1 such that L_1 contains set of strings over $\{a, b\}^*$ ending with 'b' not containing 'aa'.
   $L_1 = \{b, ab, bb, b \ldots b, bab \ldots bab, \ldots\}$. Note that whenever there exist a next symbol is b which forced.
   Regular expression is $(b + ab)^*(b + ab)$.
   Consider the regular language L_2 such that L_2 contains set of strings starting with 'ab', ending with b having even length.
   The strings in L_2 can be either ab or strings starting with ab followed by $(a + b)$, since the strings should end with b, then any substring before b should be of even length. Therefore, $(aa + ab + ba + bb)$ generates the set of all even length strings over $\{a, b\}$.
   Regular expression is $ab + (ab(a + b)(aa + ab + ba + bb)^*b)$.

2. Let us see some example languages.
   (i) $L = \emptyset$
   (ii) $L = \{\epsilon\}$ i.e it contains one element namely empty set
   (iii) $L =$ Set of strings ending with b not containing aa.
   Let us construct FA for (i).
   FA1: It has just start state and no final state.
   FA2: It has two states one is start state and other one is non final state without arrow.
   FA3: It has two states one is start state and other one is final state without arrow
   All the above FAs accepts empty set. The Regular expression corresponds to empty set is empty set itself i.e R.E=$\emptyset$
   Similarly, we construct FA for (ii). FA has just one state which is both start state as well as final state. R.E is ϵ.
   Now on the similar line we construct FA for (iii). Whenever we read a, we force to read b. Some example strings are $L = \{b, ab, bb, bb \ldots b, babb \ldots bab,$ etc$\}$. Not that b can come any number of times and ab can also come any number of times. The simplest string is $(b + ab)$. Let b^* denotes the zero or more occurrence of b. The R.E is $(b + ab)^*(b + ab)$. If we want to get n number of b's, then we get $n - 1$ $b's$ from $(b + ab)^*$, and n^{th} b from $(b + ab)$.
   Now let us construct finite automaton with epsilon transition for (iii)

3. L = Set of strings over $\{a, b\}$, starting with ab, ending with b,having even length.
   We first identify the base string for this language. Let us find base string of length 4. Thee are only two possibilities, $abab$ and $abbb$.
   Now we extend the following 4 length base string. It has to start with ab and end with b. After ab, $aorb$ can come i.e $aba \ldots\ldots b$. We observe that the in between string length must be even. It can be from any of the following four $(aa + ab + ba + bb)$.
   R.E is $ab(a + b)(aa + ab + ba + bb)^*b$.

4. L = set of strings over $\{a, b\}$ containing ab. Here, prefix and suffix could be any string over $\{a, b\}$. ie $\ldots\ldots ab \ldots\ldots$. Therefore, R.E is $(a + b)^*ab(a + b)^*$

5. L = set of strings over $\{a, b\}$ not containing ab. Here, any number of $a's$, any number of $b's$, and any number of $ab's$ are allowed. R.E is b^*a^*.
   we combine (iv) + (v), i.e $L_1 \cup L_2$ and we obtain a language that accepts all possible string over $\{a, b\}$. R.E is $(a + b)^*$.
6. L = set of strings over $\{a, b\}$ having odd number of $a's$.

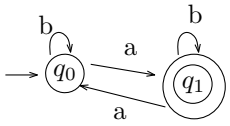


Fig. 56: *DFA that accepts set of strings over* $\{a, b\}$ *having odd number of* $a's$.

Let us analyze the pattern.
$b \ldots\ldots ba$
$b \ldots\ldots bab \ldots\ldots bab \ldots bab \ldots b$ i.e between 2 $a's$ any number $b's$ can come.
Looking at this in different perspective, We have more than one R.E.
- $b^*ab^*(ab^*ab^*)^*$.
- $b^*a(b^*ab^*a)^*b^*$.
- $b^*ab^*(b + ab^*a)^*$.
- $b^*a(b + ab^*a)^*$.

7. L = set of strings over $\{a, b\}$ having even number of $a's$ and even number of $b's$.

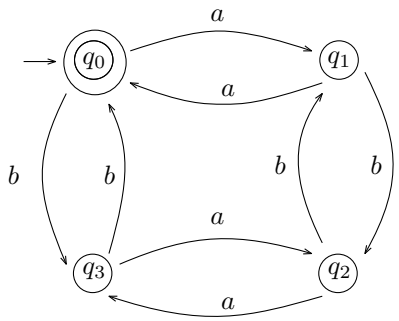


Fig. 57: *DFA that accepts set of strings over* $\{a, b\}$ *having even number of* $a's$ *and even number of* $b's$.

Let us analyze base strings.
$aa\ aa\ aa\ aa\ \ldots\ldots$
$bb\ bb\ bb\ bb\ \ldots\ldots$
$aa\ aa\ bb\ bb\ \ldots\ldots$
$ab\ ba$
$ab\ bbbb\ldots bb\ ba$
$ab\ bbbb\ldots bb\ ab$
$ba\ bbbb\ldots bb\ ba$
$ba\ aaaabbbb\ ba$
This tells that we can begin with ab and end with ab or begin with ab and end with ba or begin with ba and end with ba.
R.E $(aa + bb + (ab + ba)(aa + bb)^*(ab + ba))^*$. Note that every string generated out of this expressions has even number of $a's$ and even number of $b's$.

DFA to R.E conversion Algorithm
Now we shall see how to convert a DFA to an equivalent R.E.

1. Consider the following language $L = \{a^n b^m \mid n, m \geq 1\}$

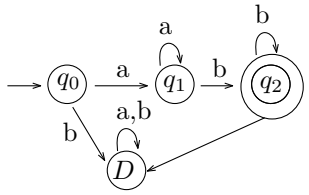


Fig. 58: *DFA that accepts* $a^n b^m, n, m \geq 1$

Let us analyze this language with respect to some example strings. Base string is ab
$aa\ldots\ldots a\ bb\ldots\ldots b$. This tell us, any a followed by zero or more $a's$ and b followed by zero or more $b's$. Using this observation we obtain R.E as follows
R.E is aa^*bb^*. Note: Finding R.E for an arbitrary DFA is difficult, so we look for a systematic way of constructing regular expressions out of DFA. The idea behind this construction is that for each state what substring can be generated until that state.
We shall now construct system of equation w.r.t each state. $q_0 = \epsilon$ (Since q_0 is the star state we write epsilon. i.e no incoming arrow)
$q_1 = q_0a + q_1a$ (there is an incoming arrow from q_0, so we write q_0a. Similarly it has self loop, so q_1a)
Note: While writing equation we write State (q_0) first followed by a symbol (a).
$q_2 = q_1b + q_2b$
$D = q_0b + q_2a + Da + Db$
Now we start simplifying the equations. We know $q_0 = \epsilon$, so we substitute q_0 in q_1.
$q_1 = a + q_1a$
We use **Arden's Theorem** to solve this system equations. If $R = RP + Q$ then $R = QP^*$
$q_1 = aa^*$
$q_2 = q_1b + q_2b$
$q_2 = q_2b + aa * b$
$q_2 = aa^*bb^*$
$D = b + aa^*bb^*a + (a = b)D$
$=b + aa^*bb^*a(a + b)^*$

The R.E that we see at the final state is the R.E for this language. $\therefore$ R.E=$b + aa^*bb^*a(a + b)^*$

2. L = set of string having even number of $a's$.

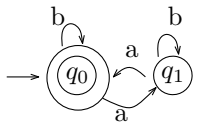


Fig. 59: *DFA that accepts set of string having even number of $a's$.*

$q_0 = \epsilon + q_0b + q_1a$
$q_1 = q_1b + q_0a$
$q_1 = (q_0a)b^*$
$q_0 = \epsilon + q_0b + (q_0a)b^*a$
$q_0 = \epsilon + q_0(b + ab^*a)$
$q_0 = q_0(b + ab^*a) + \epsilon$
$q_0 = \epsilon(b + ab^*a)^* = (b + ab^*a)^*$
Since q_0 is the accepting state, R.E is $(b + ab^*a)^*$.

3. L = set of string having odd number of $a's$.

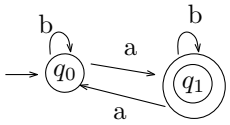


Fig. 60: *DFA that accepts set of strings over $\{a, b\}$ having odd number of $a's$.*

Set of strings having odd number of $a's$ $q_0 = \epsilon + q_0b + q_1a \implies q_0 = q_0b + (\epsilon + q_1a)$
$q_0 = (q_1a + \epsilon)b^*$
$q_1 = q_0a + q_1b$
$= q_1b + q_0a$
$= q_1b + ((q_1a + \epsilon)b^*)a$
$= q_1b + q_1ab^*a + b^*a$
$= q_1(b + ab^*a) + b^*a$
$= (b^*a)(b + ab^*a)^*$
Solution to q_1 is the R.E for this language. R.E = $(b^*a)(b + ab^*a)^*$

4. L = set of string $\{a, b\}$ containing aba.

$q_0 = \epsilon + q_0b + q_2b$
$= \epsilon + q_0b + (q_1b)b$
$= \epsilon + q_0b + (q_0a)a^*bb$
$= q_0(b + aa^*bb) + \epsilon$
$q_0 = \epsilon(b + aa^*bb)^*$

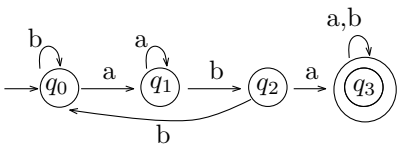


Fig. 61: *DFA that accepts strings containing aba*

$q_1 = q_0a + q_1a$
$= (q_0a)a^*$
$q_2 = q_1b$
$q_3 = q_2a + q_3(a + b)$
$= q_1ba + q_3(a + b)$
$= (b + aa^*bb)^*aa^*ba + q_3(a + b)$
$= (b + aa^*bb)^*aa^*ba(a + b)^*$

DFA with multiple final states

5. L = set of string $\{a, b\}$ not containing aa and bb.

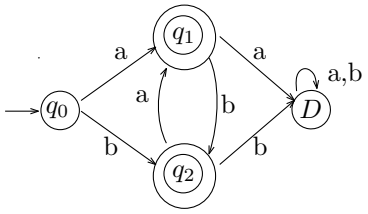


Fig. 62: *DFA that accepts strings not containing aa and bb*

$q_0 = \epsilon$
$q_1 = q_0a + q_2a$
$q_2 = q_0b + q_1b$
$D = Q_1a + q_2b + D(a + b)$
$q_1 = \epsilon a + q_2a \implies a + q_2a$
$q_2 = \epsilon b + q_1b \implies q_2b + q_1b$
$q_1 = a + (b + q_1b)a \implies a + ba + q_1ba$
$q_1 = (a + ba)(ba)^*$
$q_2 = b + (a + ba)(ba)^*b$
$= b + (a + q_2a)b$
$= b + ab + q_2ab$
$= (b + ab)(ab)^*$
R.E = $= (b + ab)(ab)^*$

6. Consider the following DFA
$q_0 = \epsilon + q_1 + q_2b$
$q_1 = q_0a + q_1b + q_2a$

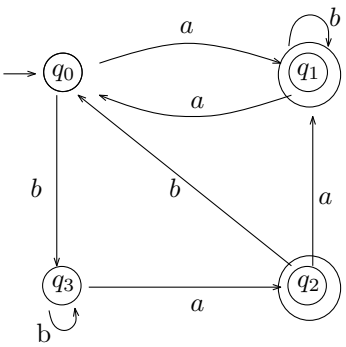


Fig. 63: *DFA*

$q_2 = q_3a$
$q_3 = q_0b + q_3b \implies q_0bb^*$
$q_2 = q_0bb^*a$
$q_1 = q_1b + q_0a + q_0bb^*aa$
$= (q_0a + q_0bb^*aa)b^*$
$q_0 = \epsilon + (q_0a + q_0bb^*aa)b^*a + q_0bb^*ab$
$= \epsilon + q_0((a + bb^*aa)b^*a + bb^*ab)$
$= \epsilon(ab^*a + bb^*aab^*a + bb^*ab)^*$
$q_1 = (ab^*a + bb^*aab^*a + bb^*ab)(ab^* + bb^*aab^*)$
$q_2 = (ab^*a + bb^*aab^*a + bb^*ab)bb^*a$
Since it has two final states q_1, and q_2, solution to the above DFA is $q_1 + q_2$
R.E is $(ab^*a + bb^*aab^*a + bb^*ab)(ab^* + bb^*aab^*) + (ab^*a + bb^*aab^*a + bb^*ab)bb^*a$

5 Pumping Lemma

Properties of regular set: We have established that the class of languages known as the regular languages has four different descriptions. They are the languages accepted by (i) DFA (ii) NFA (iii) ϵ-NFA, and they are also defined by (iv) regular expressions. A natural question is that are there non-regular languages. In this section, we answer this interesting question. We look at the question in two different perspectives. Let $\sum = \{a_1, a_2, \ldots, a_k\}$, k is a fixed integer. Since $\sum$ is finite, $\sum^*$ is countable infinite. We know that $L \subseteq \sum^*$. From this we can infer that each subset is a candidate for regular set (regular language). The number of subsets in $\sum^*$ is $\mathcal{P}(\sum^*)$. We know that $\mathcal{P}(\sum^*)$ is uncountable infinite. This shows that there are uncountable languages with respect to $\sum^*$. Now we look at this question another perspective. By definition, finite automaton implies finite number of states. i.e $\sum$ is finite and number of states are finite. Since $\sum$ is finite and number of states are finite, number of possible DFAs is finite. For each DFA we have regular expression. Therefore, number of regular set is finite. But we know that the number of candidates for regular set (regular language) is uncountable. We observe that the number candidates for competing regular set (regular language) is uncountable and the number of regular set is actually finite. Clearly, this shows that there are non-regular languages. In this section, we shall see a technique known as the "pumping lemma" for showing certain languages is not a regular language.
Consider a four state finite automaton given in Figure 64 without self-loop and cycles.
The language accepted by this FA is $L = \{abc\}$. The set of strings is finite. But if we allow self loop and cycles then the set of strings accepted by the language is infinite. For example consider the following FA.
We can generate infinite strings. String of length 3 is *abc*, string of length 4 is *babc* or *cabc*, string of length 5 is $(b + c)(b + c)abc$. In general, $abc(abc)^*$ or $(b + c)^*abc$. The number of states in the above DFA is 4, i.e $|Q| = 4$. Consider a regular string $w \in L$ whose length is 4. For example $w = babc$ or $w = cabc$. We can

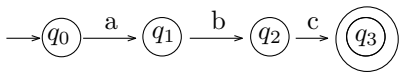


Fig. 64: *DFA without self-loop and cycles*

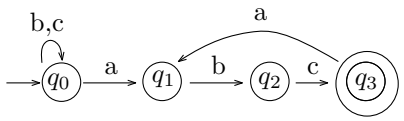
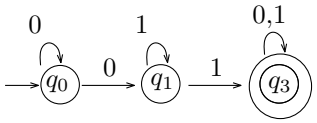


Fig. 65: *DFA with self-loop and cycles*

decompose w into three parts such as u, v and z. The decomposition of $w = babc$, is as follows; $u = \epsilon$, $v = b$, $z = abc$. Note that we can loop over v again and again, i.e $bbabc$ or $bbbabc$ or $bb \ldots babc$. Interestingly, the strings obtained with the help of v is also a regular language. The property that we infer is the following; let w be the string which is accepted by the automaton. There exists a substring in w such that we can loop over multiple times. Interestingly, all those strings generated by looping over the substring is also a part of L. similarly, we decompose $w = cabc$.
Consider $|w| = 6$. $w = abcabc$, $u = abc$, $v = abc$, $z = \epsilon$. Here if we loop over v, then we get $abcabcabc$, $abcabcabcabc$, etc.
In general, $w \in L$
$w = uvz$
$uvvz$ we can loop over v
$uvvvvz$
$\vdots$
uv^iz, An interesting observation is that if $|w = 4$, $|uv| \leq 4$, and $v \geq 1$ then all the generated strings are accepted by L. We cal this decomposition as constrained decomposition. An interesting property is that for a string $w \in L$ there exists a constrained decomposition of w.
In general, Given an FA with $|Q| = n$, if we choose $w \in L$ such that $|w| \geq n$ then $\exists u \exists v \exists z (w = uvz; |uv| \leq n, |v| \geq 1, uv^iz \in L, \forall i \geq 0)$.
Consider the following FA.



$|Q| = 3$, we choose $|w| \geq 3$, $w \in L$. let $w = 001$, we decompose(Constrained decomposition) w as follows. $u = \epsilon$, $v = 0$, $z = 01$. We loop over v and generate many strings. For eg. 0001, 00001, 00...001. Note that all are accepted by L. Let us try another decomposition of w respecting the constraint $|uv| \leq 3$ and $|v| \geq 1$. $u = 0$, $v = 0$, $z = 01$. All the generated strings are accepted by L.
Let L be a regular language. Then there exists a constant n such that for every string w in L such that $|w| \geq n$, we can break w into three strings, $w = uvz$, such that:
1. $v \neq \epsilon$.
2.$|uv| \leq n$.
3. For all $k \geq 0$, the string uv^kz is also in L.

Let L be a regular language. Then, $\exists n \exists w(|w| \geq n \rightarrow \exists u \exists v \exists z(w = uvz \wedge |uv| \leq n \wedge |v| \geq 1 \wedge \forall i(i \geq 0 \rightarrow uv^i z \in L)))$, n denotes number of states in minimum DFA, $w \in L$. If L is Regular then we can write equivalent FOL expression. Contrapositive of this statements says If FOL expression is not possible then L is Non-Regular. We obtain the following expression by negating the FOL.
$\exists L \forall n \exists w(|w| \geq n \wedge \forall u \forall v \forall z(w = uvz \wedge |uv| \leq n \wedge |v| \geq 1 \rightarrow \exists i(i \geq 0 \wedge uv^i z \notin L))) \implies L$ is Non-regular.

1. $L_1 = \{a^n b^m | n, m \geq 1\}$

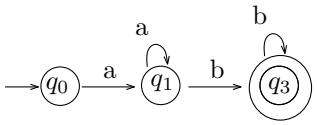


Fig. 66: *DFA that accepts* $L_1 = \{a^n b^m | n, m \geq 1\}$

R.E of L_1 is aa^*bb^*. Therefore, L_1 is regular.

$L_2 = \{a^n b^n | n \geq 1\}$. Here, we need to remember the number of $a's$ read, so that it can be matched with number of $b's$. But intuitively we know the exact count cannot be remembered by FA. So, this language is a candidate for non-regular language.
(i) First step is to choose w, $|w| \geq n$. We choose $w = a^n b^n$, $|w| = 2n$.
(ii) Decomposition of w. $u, v = a^n$, $z = b^n$. (Decomposition should satisfy the constraint $(\forall n \exists w(|w| \geq n \wedge \forall u \forall v \forall z(|uv| \leq n \wedge |v| \geq 1)$
(iii) $|v| = k$ (k $a's$)
$|uv| \leq n$, $|v| = k$, $|u| \leq n - k$
(iv) $w = a^{n-k}(a^k)^i b^n$
$i = 0$, $w = a^{n-k}(a^k)^i b^n$
$= a^{n-k} b^n$, $k \leq 1$
it is clear that $a^{n-k} b^n \notin L$
Therefore, L_2 is Non-regular.

Note: Suppose we choose $i = 1$, then $a^{n-k} a^k b^n \implies a^n b^n \in L$. So, we have to choose i such that $w = uvz$ is not in L.

Let us choose some other w.
(i) $w = a^{\frac{n}{2}} b^{\frac{n}{2}}$, $|w| = n$.
(ii) $a^{\frac{n}{2}} b^{\frac{n}{2}}$, $|uv| \leq n$, $|v| \geq 1$
Example: $n = 4$
Decomposition of w can be $u = a,\ v = a$, $z = bb$ or $u = aa,\ v = b$, $z = b$ or $u = a,\ v = ab$, $z = b$.
Case 1: v contains only $a's$
$a^{\frac{n}{2}-k} a^k b^{\frac{n}{2}}$, $k \geq 1$
we choose $i = 0$ or $i = 2$, we get $a^{\frac{n}{2}-k} b^{\frac{n}{2}} \notin L$
Case 2: v contains only $b's$
$a^{\frac{n}{2}} b^k b^{\frac{n}{2}-k}$, $k \geq 1$
we choose $i = 0$, we get $a^{\frac{n}{2}} b^{\frac{n}{2}-k} \notin L$
Case 3: v has ab in it
$u = a \ldots a$, $k_1 = a \ldots a$, $k_2 = b \ldots b$, $z = b \ldots b$
$a^{\frac{n}{2}-k} a^{k_1} b^{k_2} b^{\frac{n}{2}-k_2}$, $k_1, k_2 \geq 1$
$a^{\frac{n}{2}-k_1} b^{\frac{n}{2}-k_2}$
Choose $i = 2$, $a^{\frac{n}{2}-k_1} a^{k_1} b^{k_2} b^{\frac{n}{2}-k_2} \notin L$

Therefore, L_2 is non-regular.

2. Let us show that the language L consisting of all strings with an equal number of 0's and 1's is not a regular language.
   $L = \{x | x \in \{0,1\}^*$ such that x has equal number of 0's and 1's $\}$
   Let us analyze special cases of the language. $L_1 = \{a^n b^n | n \geq 1\}$
   $L_2 = \{b^n a^n | n \geq 1\}$
   We know that L_1, and L_2 are not regular. We mimic special case proof to prove L is not regular.
   choose w' such that $|w| \geq n$
   $w = 0^n 1^n$
   $\forall u, v, z$; $w = uvz$;
   $v \geq 1$, $|uv| \leq n$
   v contains only $0's$
   $0^{n-k} 0^k 1^k$, $k \geq 1$
   $w' = 0^{n-k}(0^k)^i 1^n$
   when $i = 0$, $0^{n-k}1^n \notin L \implies L$ is not regular.
   Note: If subset is not regular then super set is also not regular.
   This proof can be used as frame work to solve many problem.
   For example $L^{=}\{x | x \in \{0,1\}^*$,$x$ is a palindrome$\}$.
   Palindrome can be of the form $0^n 0^n$ or $0^n 10^n$.
   $w = 0^n 10^n$
   $= 0^{n-k}(0^k)^i 10^n$
   when $i = 0$
   $0^{n-k}10^n$, w.k.t $k \geq 1$, Clearly, this shows that $0^{n-k}10^n \notin L$. Therefore, L is not regular.

3. Let us show that the language L consisting of all strings of 0's whose length is a prime is not a regular language.
   $L = \{0^n | n$ is a prime number$\}$
   $L = \{0^3, 0^5, 0^7, \ldots\}$
   $w = 0^n, |w| \geq n$
   we decompose w into three strings u, v, and z is as follows: $0^r\ 0^k\ 0^{n-r-k}$
   $w = 0^r.(0^k)^i.0^{n-r-k}$
   $= 0^r.0^k.(0^k)^{i-1}.0^{n-r-k}$
   $= 0^n.(0^k)^{i-1}$
   $= (0^{n+k(i-1)}$
   when $i = n+1 \implies 0^{n+k.n}$
   $= 0^{n(k+1)}$
   Since $k \geq 1$ and $n \geq 3, k+1 \geq 2$ and $n \geq 3$,
   This shows that $= 0^{n(k+1)}$ is composite number divisible by n or $k+1$
   $w' = 0^{n(k+1)} \notin L$
   Therefore, L is non-regular.
   Note: (i) $L = 0^n$ — $n = m^2$ for $m \geq 1$ (ii)$n = m!$. when n has additional property then L is not regular.
   So, (i), and (ii) is not regular.

4. Show that prefect square testing is not a regular language. $L = \{0^{n^2} | n \geq 1\}$
   $L = \{0^{1^2}, 0^{2^2}, 0^{3^2}, \ldots\}$
   $0^r\ 0^k\ 0^{n^2-r-k}$
   $w' = 0^r.(0^k)^i.0^{n^2-r-k}$
   $= 0^r.0^k.(0^k)^{i-1}.0^{n^2-r-k}$
   $= 0^{n^2}(0^k)^{i-1}$
   when $i = 2$

$w' = 0^{n^2}.0^k$
$= 0^{n^2+k}$
Claim: $n^2 + k \neq m^2$ for any $m \geq 1$
We know that $1 \leq k \leq n$
we obtain $n^2 + 1 \neq n^2 + +2n + 1$
$n^2 + 2 \neq (n+1)^2$
$\vdots$
$n^2 + n \neq (n+1)^2$
$n^2 < n^2 + k < (n+1)^2$
We observe that $n^2 + k$ is not a perfect square. This shows that $w' \neq L \implies L$ is not regular.

6 Minimization of DFA

We have seen that some FA could be simplified either by deleting states inaccessible from the start state or by collapsing states that were equivalent in some sense. In the subset construction and while combining two DFA into one, we have seen that DFA obtained were not having minimum number of states. In the minimization process of DFA, we remove the states inaccessible from the start state or by collapsing equivalent states.

We shall see a technique to prove a lower bound on the number of states of any DFA that recognizes a given language. Given a DFA, we group them into set of final states and set of non-final states. Consider set of final states as one state and set of non-final states as another state. Check whether such a FA is a DFA or not. If it is not a DFA, then subdivide the set which causes non-determinism.
Consider the DFA in Figure 67 accepts a string if and only if it ends in 00 or 11.

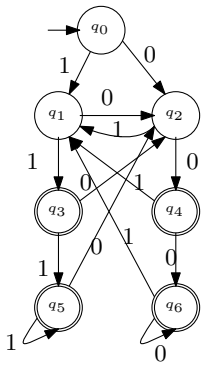


Fig. 67: DFA accepting strings that ends with 00 or 11.

We shall analyze whether the DFA in Figure 67 is a DFA having minimum number of states or not.
Step 1: Group the set of final states as one state and set of non-final states as one state.
Step 2: Check whether such a grouping gives us a DFA or not.

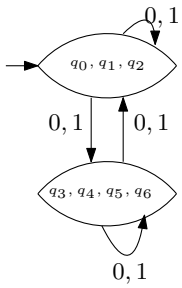
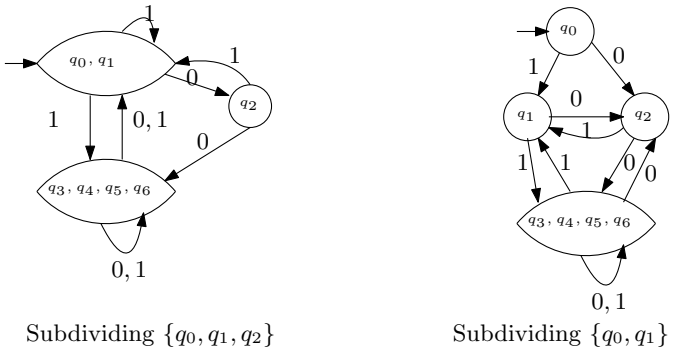


Fig. 68: DFA accepting strings that ends with 00 or 11.

In Figure 68, we can observe the transition function :
$\delta(\{q_0, q_1, q_2\}, 0) = \{q_2, q_4\}$ $\delta(\{q_3, q_4, q_5, q_6\}, 0) = \{q_2, q_6\}$
$\delta(\{q_0, q_1, q_2\}, 1) = \{q_1, q_3\}$ $\delta(\{q_3, q_4, q_5, q_6\}, 1) = \{q_1, q_5\}$
Note that the transitions has state q_i, then we make the transition to the corresponding group.
Step 3: We can observe that the FA obtained in Figure 68 is not a DFA. Hence we subdivide the states as in Figure 69.



Subdividing $\{q_0, q_1, q_2\}$ Subdividing $\{q_0, q_1\}$

Fig. 69: DFA accepting strings that ends with 00 or 11.

Continue the subdivision until we obtain a DFA.
Step 4: Upon subdividing the final states, we obtain the DFA as in Figure 70.

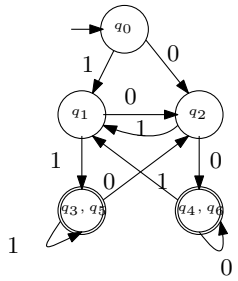


Fig. 70: DFA accepting strings that ends with 00 or 11.

Remark: For every regular language L, there is a unique (up to re-labeling of the states) minimal-state DFA M such that $L = L(M)$.
Consider the DFA M such that $p,\ q$ are two states in M.
We say that the state p is *distinguishable* from state q if and only if there is a $w \in \Sigma^*$ that distinguishes p and q (the strings accepted by state p is not same as strings accepted by state q and strings rejected by state p is not same as strings rejected by state q.)
Similarly, we say that the state p is *not distinguishable (ND)* from state q if and only if p is not distinguishable from q (the strings accepted by state p is same as strings accepted by state q and strings rejected by state p is same as strings rejected by state q.)
Note that pairs of indistinguishable states are redundant, hence we remove one of the ND state.

Properties of DFA or regular sets
We shall define a binary relation R on regular sets whose universe of discourse is Σ^*. We say xRy, $x, y \in \Sigma^*$ if and only if $\delta(q_0, x) = \delta(q_0, y)$. Note that R is an equivalence relation. Since xRx, R is reflexive. Further, since xRy implies yRx, R is symmetric. We can observe that xRy and yRz implies xRz, hence R is transitive.
If R is an equivalence relation defined on the set S, then R partitions S into set of equivalence classes. Similarly, R partitions Σ^* into set of equivalence classes. For each $q \in Q$, there exists an equivalence class.
In addition, if xRy, then $xzRyz$ for all $z \in \Sigma^*$, such an equivalence relation is said to be *right invariant.* now we shall prove that R satisfies right invariant.
Consider, $\hat{\delta}(q_0, xz) = \hat{\delta}(\hat{\delta}(q_0, x), z) = \hat{\delta}(q_0, y), z = \hat{\delta}(q_0, yz)$.
Therefore, R is right invariant.
Consider the example where DFA in Figure 71 accepts the set of strings ending with 'a'.

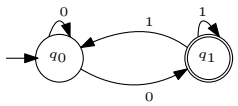


Fig. 71: DFA accepting strings that ends with 'a'.

We can observe that $\hat{\delta}(q_0, ba) = \hat{\delta}(q_0, baba) = \ldots = q_1$ and $\hat{\delta}(q_0, bb) = \hat{\delta}(q_0, bab) = \ldots = q_0$.
Remarks:
In the above example (DFA in Figure 71), Σ^* is partitioned into all those x, y such that $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y) = q_0$ and all those x, y such that $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y) = q_1$.
For each $q \in Q$, there exists an equivalence classes.
For DFA in Figure 71, $\Sigma^* = E_1 \cup E_2$, where $E_1,\ E_2$ are equivalence classes. The equivalence class $E_1 = \{x \mid x\}$
We see that every finite automaton induces a right invariant equivalence, defined as R, on its set of input strings. This result is formalized in the Myhill-Nerode theorem.

Theorem 1. *(The Myhill-Nerode theorem). The following three statements are equivalent:*

1. *The set $L \subseteq \Sigma^*$ is accepted by some finite automaton.*
2. *L is the union of some of the equivalence classes of a right invariant equivalence relation of finite index. (Finite index means that number of equivalence relation is finite)*
3. *Let equivalence relation R be defined by xRy if and only if for all z in Σ^*, xz is in L exactly when yz is in L. Then R is of finite index.*

Corollary 1. *L is not regular if and only if there does not exist equivalence relation having finite index (every right invariant equivalence relation is having infinite index)*