Nondeterministic Finite Automaton (NFA)

Definition

An NFA is a TG with a unique start state and a property of having single letter as label of transitions. An NFA is a collection of three things

Finite many states with one initial and some final states

Finite set of input letters, say, Σ = {a, b, c}

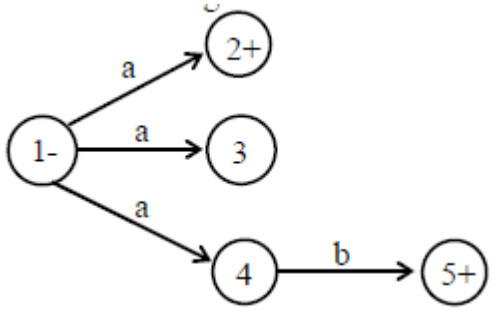
Finite set of transitions, showing where to move if a letter is input at certain state (Λ is not a valid transition), there may be more than one transition for certain letters and there may not be any transition for certain letters.

Observations

It may be observed, from the definition of NFA, that the string is supposed to be accepted, if there exists at least one successful path, otherwise rejected.

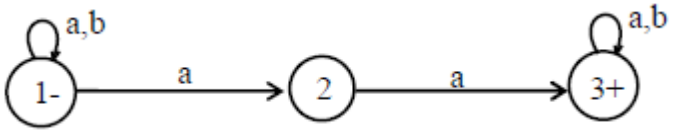
It is to be noted that an NFA can be considered to be an intermediate structure between FA and TG.

The examples of NFAs can be found in the following example.



It is to be noted that the above NFA accepts the language consisting of a and ab.

Example



It is to be noted that the above NFA accepts the language of strings, defined over Σ = {a, b}, containing aa.

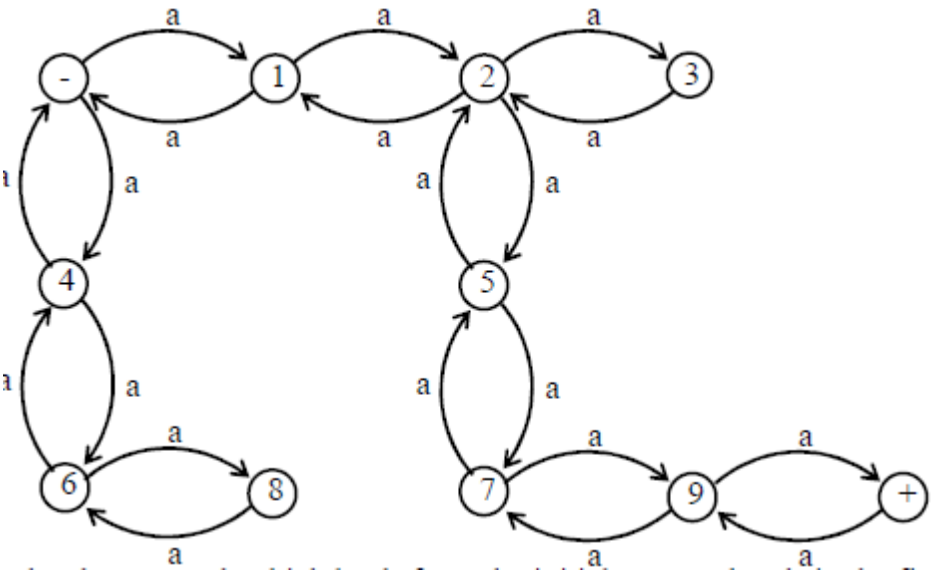
Application of an NFA

There is an important application of an NFA in artificial intelligence, which is discussed in the following

example of a maze

-	1	2	3
4	L	5	O
6	M	7	P
8	N	9	+

- and + indicate the initial and final states respectively. One can move only from a box labeled by other then L,M, N, O, P to such another box. To determine the number of ways in which one can start from the initial state and end in the final state, the following NFA using only single letter a, can help in this regard



It can be observed that the shortest path which leads from the initial state and ends in the final state, consists of

six steps i.e. the shortest string accepted by this machine is aaaaaa. The next larger accepted string is aaaaaaaa.

Thus if this NFA is considered to be a TG then the corresponding regular expression may be written as

aaaaaa(aa)*

Which shows that there are infinite many required ways

Note

It is to be noted that every FA can be considered to be an NFA as well , but the converse may not true.

It may also be noted that every NFA can be considered to be a TG as well, but the converse may not true.

NFA with Null String

Definition

If in an NFA, Λ is allowed to be a label of an edge then the NFA is called NFA with Λ (NFA-Λ).

An NFA-Λ is a collection of three things

Finite many states with one initial and some final states.

Finite set of input letters, say, Σ = {a, b, c}.

Finite set of transitions, showing where to move if a letter is input at certain state.

There may be more than one transitions for certain letter and there may not be any transition

for a certain letter. The transition of Λ is also allowed at any state.

NFA to FA

Two methods are discussed in this regard.

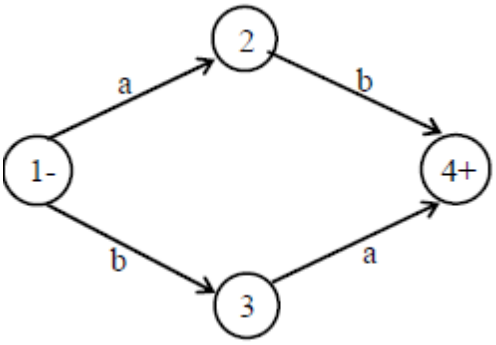
Method 1: Since an NFA can be considered to be a TG as well, so a RE corresponding to the given NFA can be determined (using Kleene's theorem). Again using the methods discussed in the proof of Kleene's theorem, an FA can be built corresponding to that RE. Hence for a given NFA, an FA can be built equivalent to the NFA.

Examples have, indirectly, been discussed earlier.

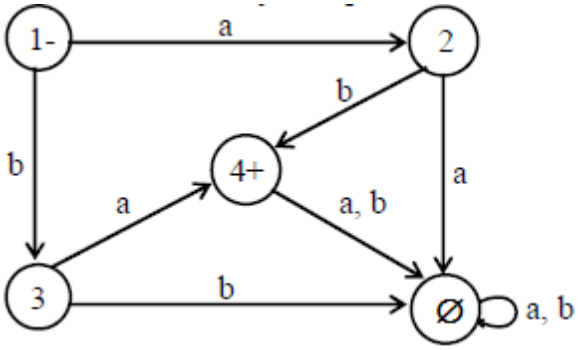
Method 2: Since in an NFA, there may be more than one transition for a certain letter and there may not be any transition for certain letter, so starting from the initial state corresponding to the initial state of given NFA, the transition diagram of the corresponding FA, can be built introducing an empty state for a letter having no transition at certain state and a state corresponding to the combination of states, for a letter having more than one transitions. Following are the examples

Example

Consider the following NFA



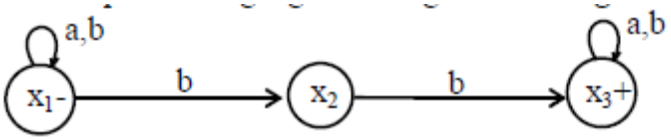
Using the method discussed earlier, the above NFA may be equivalent to the following FA



Method 3: As discussed earlier that in an NFA, there may be more than one transition for a certain letter and there may not be any transition for certain letter, so starting from the initial state corresponding to the initial state of given NFA, the transition table along with new labels of states, of the corresponding FA, can be built introducing an empty state for a letter having no transition at certain state and a state corresponding to the combination of states, for a letter having more than one transitions. Following are the example

Example

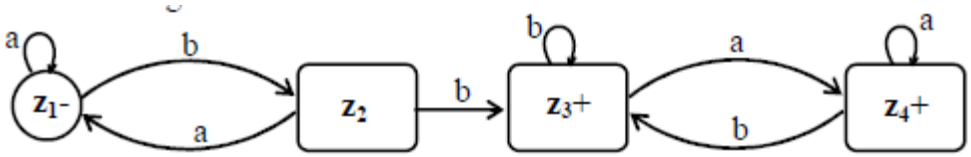
Consider the following NFA which accepts the language of strings **containing bb**



Using the method discussed earlier, the transition table corresponding to the required FA may be constructed as

Old States	New States after reading	
	a	b
$z_1\text{-}\equiv x_1$	$x_1\equiv z_1$	$(x_1,x_2)\equiv z_2$
$z_2\equiv(x_1,x_2)$	$(x_1\ ,\ \varnothing)\equiv x_1\equiv z_1$	$(x_1\ ,x_2,x_3)\equiv z_3$
$z_3+\equiv(x_1,x_2,x_3)$	$(x_1,x_3)\equiv z_4$	$(x_1\ ,x_2,x_3)\equiv z_3$
$z_4+\equiv(x_1,x_3)$	$(x_1,x_3)\equiv z_4$	$(x_1\ ,x_2,x_3)\equiv z_3$

The corresponding transition diagram follows as


a
b
a
b
b
a
b
a
z_1-
z_2
z_3+
z_4+

Finite Automaton with output

Finite automaton discussed so far, is just associated with the RE or the language.

There is a question whether does there exist an FA which generates an output string corresponding to each input string ? The answer is yes. Such machines are called machines with output.

There are two types of machines with output. Moore machine and Mealy machine

Moore machine

A Moore machine consists of the following

- A finite set of states q0, q1, q2, ⋯ where q0 is the initial state.
- An alphabet of letters Σ = {a,b,c,⋯} from which the input strings are formed.
- An alphabet Γ={x,y,z,⋯} of output characters from which output strings are generated.
- A transition table that shows for each state and each input letter what state is entered the next.
- An output table that shows what character is printed by each state as it is entered.

Note

It is to be noted that since in Moore machine no state is designated to be a final state, so there is no question of accepting any language by Moore machine. However in some cases the relation between an input string and the corresponding output string may be identified by the Moore machine. Moreover, the state to be initial is not important as if the machine is used several times and is restarted after some time, the machine will be started from the state where it was left off. Following are the examples

Example

Consider the following Moore machine having the states q0, q1, q2, q3 where q0 is the start state and

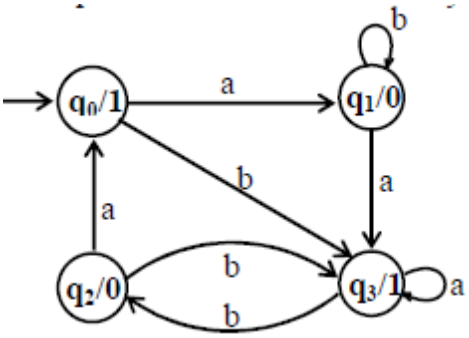
Σ = {a,b},

Γ={0,1}

the transition table follows as

Old States	New States after reading		Characters to be printed
	a	b	
q_0-	q_1	q_3	1
q_1	q_3	q_1	0
q_2	q_0	q_3	0
q_3	q_3	q_2	1

the transition diagram corresponding to the previous transition table may be



It is to be noted that the states are labeled along with the characters to be printed. Running the string abbabbba over the above machine, the corresponding output string will be 100010101, which can be determined by the following table as well

Input		a	b	b	a	b	b	b	a
State	q_0	q_1	q_1	q_1	q_3	q_2	q_3	q_2	q_0
output	1	0	0	0	1	0	1	0	1

It may be noted that the length of output string is l more than that of input string as the initial state prints out the extra character 1, before the input string is read.

Mealy machine

A Mealy machine consists of the following

- A finite set of states q0, q1, q2, ⋯ where q0 is the initial state.
- An alphabet of letters Σ = {a,b,c,⋯} from which the input strings are formed.
- An alphabet Γ={x,y,z,⋯} of output characters from which output strings are generated.
- A pictorial representation with states and directed edges labeled by an input letter along with an output character. The directed edges also show how to go from one state to another corresponding to every possible input letter.

(It is not possible to give transition table in this case.)

Note

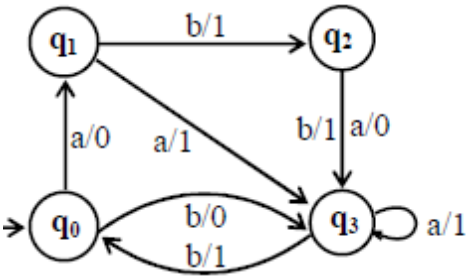
It is to be noted that since, similar to Moore machine, in Mealy machine no state is designated to be a final state, so there is no question of accepting any language by Mealy machine. However in some cases the relation between an input string and the corresponding output string may be identified by the Mealy machine. Moreover, the state to be initial is not important as if the machine is used several times and is restarted after some time, the machine will be started from the state where it was left off. Following are the examples

Example

Consider the Mealy machine , having the states q0, q1, q2, q3 , where q0 is the start state and

Σ = {a,b},

Γ={0,1}



Running the string abbabbba over the above machine, the corresponding output string will be 11011010, which can be determined by the following table as well

Input		a	b	b	a	b	b	b	a
States	q_0	q_1	q_2	q_3	q_3	q_0	q_3	q_0	q_1
output		0	1	1	1	1	0	1	0

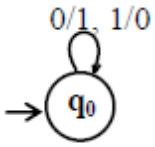
It may be noted that in Mealy machine, the length of output string is equal to that of input string.

Example

Consider the following Mealy machine having the only state q0 as the start state and

Σ = {0,1},

Γ= {0,1}



If 0011010 is run on this machine then the corresponding output string will be 1100101.

This machine is called **Complementing machine.**

Constructing the incrementing machine

In the previous example of complementing machine, it has been observed that the input string and the

corresponding output string are 1's complement of each other. There is a question whether the Mealy machine can be constructed, so that the output string is increased, in magnitude, by 1 than the corresponding input string?

The answer is yes.

This machine is called the incrementing machine. Following is how to construct the incrementing machine.

Before the incrementing machine is constructed, consider how 1 is added to a binary number.

Since, if two numbers are added, the addition is performed from right to left, so while increasing the binary number by 1, the string (binary number) must be read by the corresponding Mealy machine from right to left, and hence the output string (binary number) will also be generated from right to left.

Consider the following additions

a)
$$\begin{array}{r} 100101110 \\ +\ 1 \\ \hline 100101111 \end{array}$$

b)
$$\begin{array}{r} 1001100111 \\ +\ 1 \\ \hline 1001101000 \end{array}$$

It may be observed from the above that If the right most bit of binary number, to be incremented, is 0, the output binary number can be obtained by converting the right most bit to 1 and remaining bits unchanged.

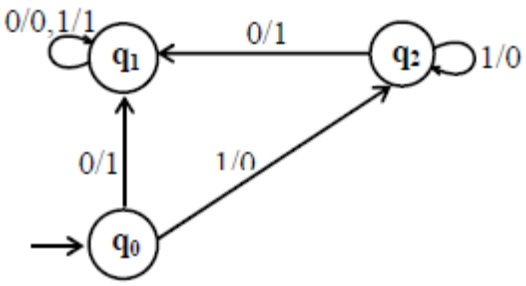
If the right most bit of binary number is 1 then the output can be obtained, converting that 1 along with all its concatenated 1's to 0's, then converting the next 0 to 1 and remaining bits unchanged.

The observations (a) and (b) help to construct the following Incrementing (Mealy) machine.

The Mealy machine have the states q0, q1, q2 , where q0 is the start state and

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

$\Gamma = \{0,1\}$



It may be observed that, in the incrementing machine, if 0 is read at initial state **q0**, that 0 is converted to 1 and a no change state **q1** (no carry state) is entered where all 0's and all 1's remain unchanged. If 1 is read at initial state, that 1 is converted to 0 and the state q2**(**owe carry state**)** is entered, where all 1's are converted to 0's and at that state if 0 is read that 0 is converted to 1 and the machine goes to no change state.

If the strings 100101110 and 1001100111 are run over this machine, the corresponding output strings will be 100101111 and 1001101000 respectively.

Note

It is to be noted that if the string 111111 is run over the incrementing machine, the machine will print out 000000, which is not increased in magnitude by 1. Such a situation is called an overflow situation, as the length of output string will be same as that of input string.

It may also be noted that there exists another incrementing machine with two states.

Applications of Incrementing and Complementing machines

1's complementing and incrementing machines which are basically Mealy machines are very much helpful in computing.

The incrementing machine helps in building a machine that can perform the addition of binary numbers.

Using the complementing machine along with incrementing machine, one can build a machine that can perform the subtraction of binary numbers, as shown in the following method

Subtracting a binary number from another

Method

To subtract a binary number b from a binary number a

Add 1's complement of b to a (ignoring the overflow, if any)

Increase the result, in magnitude, by 1 (use the incrementing machine). Ignoring the overflow if any.

Note: If there is no overflow in (1). Take 1's complement once again in (2), instead. This situation occurs when b is greater than a, in magnitude. Following is an example of subtraction of binary numbers

Example

To subtract the binary number 101 from the binary number 1110, let

a = 1110 and b = 101 = 0101.

(Here the number of digits of b are equated with that of a)

Adding 1's complement (1010) of b to a.

```
  1110
 +1010
 11000  which gives 1000 (ignoring the overflow)
```

Using the incrementing machine, increase the above result 1000, in magnitude, by 1

```
  1000
   +1
 1001 which is the same as obtained by ordinary subtraction.
```

Note

It may be noted that the above method of subtraction of binary numbers may be applied to subtraction of decimal numbers with the change that 9's complement of b will be added to a, instead in step (1).

Equivalent machines

Two machines are said to be **equivalent** if they print the same output string when the same input string is run on them.

Remark:

Two Moore machines may be equivalent. Similarly two Mealy machines may also be equivalent, but a Moore machine can't be equivalent to any Mealy machine. However, ignoring the extra character printed by the Moore machine, there exists a Mealy machine which is equivalent to the Moore machine.

Theorem

Statement

For every Moore machine there is a Mealy machine that is equivalent to it (ignoring the extra character printed by the Moore machine).

Proof:

Let M be a Moore machine, then shifting the output characters corresponding to each state to the labels of corresponding incoming transitions, machine thus obtained will be a Mealy machine equivalent to M.

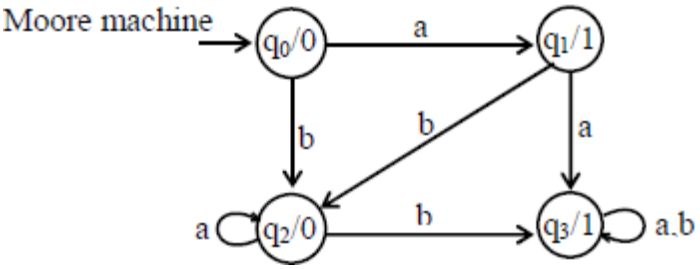
Note

It may be noted that while converting a Moore machine into an equivalent Mealy machine, the output character of a state will be ignored if there is no incoming transition at that state. A loop at a state is also supposed to be an incoming transition.

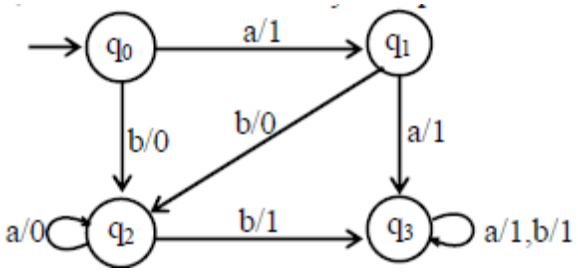
Following is the example of converting a Moore machine into an equivalent Mealy machine

Example

Consider the following Moore machine



Using the method described earlier, the above machine may be equivalent to the following Mealy machine



Running the string abbabbba on both the machines, the output string can be determined by the following table

Input		a	b	b	a	b	b	b	a
States	q_0	q_1	q_2	q_3	q_3	q_3	q_3	q_3	q_3
Moore	0	1	0	1	1	1	1	1	1
Mealy		1	0	1	1	1	1	1	1

Theorem

Statement

For every Mealy machine there is a Moore machine that is equivalent to it (ignoring the extra character printed the Moore machine).

Proof

Let M be a Mealy machine. At each state there are two possibilities for incoming transitions

The incoming transitions have the same output character.

The incoming transitions have different output characters.

If all the transitions have same output characters, then shift that character to the corresponding state.

If all the transitions have different output characters, then the state will be converted to as many states as the number of different output characters for these transitions, which shows that if this happens at state qi then qi will be converted to qi1 and qi2 *i.e.* if at qi there are the transitions with two output characters then qi1 for one character and qi2 for other character.

Shift the output characters of the transitions to the corresponding new states qi1 and qi2. Moreover, these new states qi1 and qi2 should behave like qi as well. Continuing the process, the machine thus obtained, will be a Moore machine equivalent to Mealy machine M.

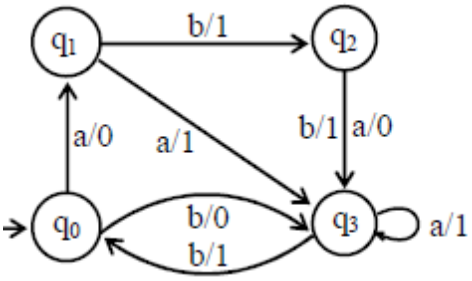
Note

It may be noted that if there is no incoming transition at certain state then any of the output characters may be associated with that state.

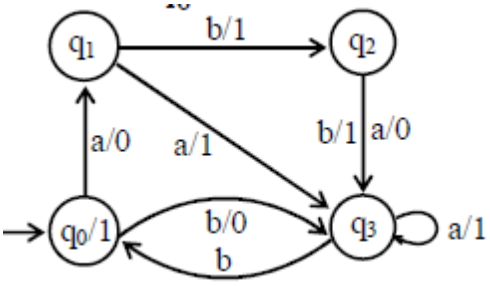
It may also be noted that if the initial state is converted into more than one new states then only one of these new states will be considered to be the initial state. Following is an example

Example

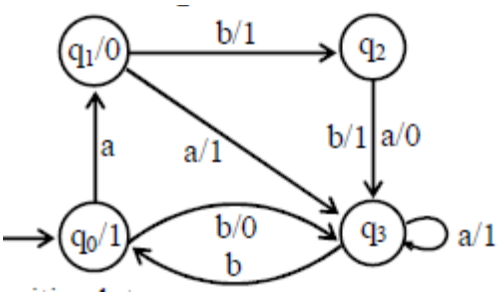
Consider the following Mealy machine



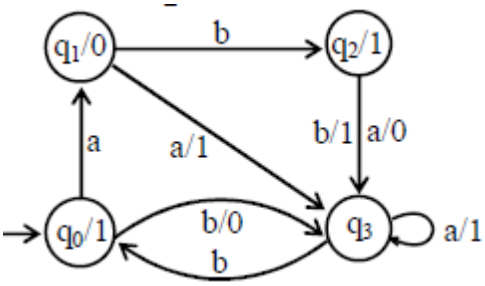
Shifting the output character 1 of transition b to **q0**



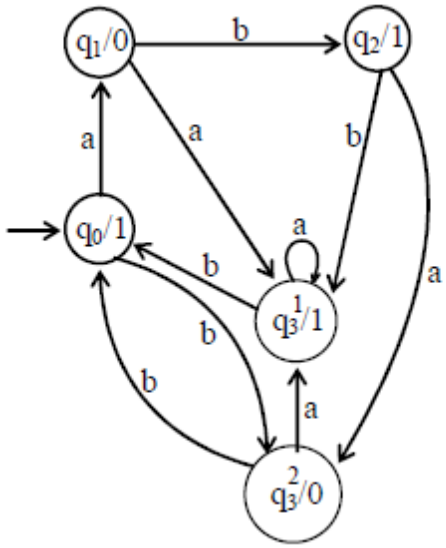
Shifting the output character 0 of transition a to **q1**



Shifting the output character 1 of transition b to **q2**



Splitting **q3** into **$q_1$3** and **$q_2$3**



Running the string abbabbba on both the machines, the output strings can be determined by the following table

Input		a	b	b	a	b	b	b	a
States	q_0	q_1	q_2	q_3	q_3	q_0	q_3	q_0	q_1
Mealy		0	1	1	1	1	0	1	0
Moore	1	0	1	1	1	1	0	1	0

Regular languages

As already been discussed earlier that any language that can be expressed by a RE is said to be regular language,so if L1 and L2 are regular languages then L1 + L2 , L1L2 and L1* are also regular languages.

Nonregular languages

The language that cannot be expressed by any regular expression is called a **Nonregular language**.

The languages **PALINDROME** and **PRIME** are the examples of nonregular languages.

Note: It is to be noted that a nonregular language, by Kleene's theorem, can't be accepted by any FA or TG.

Context Free Grammar (CFG)

The earliest computers accepted no instructions other then their own assembly language. Every procedure, no matter how complicated , had to be encoded in the set of instructions, LOAD, STORE, ADD the contents of two registers and so on. The major problem was to display mathematical formulas as follows

$$A = \frac{\frac{1}{2} + 9}{4 + \frac{8}{21} + \frac{5}{3 + \frac{1}{2}}}$$

So, it was necessary to develop a way of writing such expressions in one line of standard typewriter symbols, so that in this way a high level language could be invented. Before the invention of computers, no one would ever have dreamed of writing such complicated formula in parentheses *e.g.* the right side of formula can be written as

((1/2)+9)/(4+(8/21)+(5/(3+(1/2))))

The high level language is converted into assembly language codes by a program called compiler.

The compiler that takes the user's programs as its inputs and prints out an equivalent program written in assembly language.

Like spoken languages, high level languages for computer have also, certain grammar. But in case of computers, the grammatical rules, don't involve the meaning of the words.

It can be noted that the grammatical rules which involve the meaning of words are called **Semantics**, while those don't involve the meaning of the words are called **Syntactics**.

e.g. in English language, it can not be written "Buildings sing", while in computer language one number is as good as another.

e.g. X = B + 10, X = B + 999

Remark

In general, the rules of computer language grammar, are all syntactic and not semantic. A law of grammar is in reality a suggestion for possible substitutions.

CFG terminologies

Terminals: The symbols that can't be replaced by anything are called terminals.

Non-Terminals: The symbols that must be replaced by other things are called non-terminals.

Productions: The grammatical rules are often called productions.

CFG

CFG is a collection of the followings

- An alphabet Σ of letters called terminals from which the strings are formed, that will be the words of the language.
- A set of symbols called non-terminals, one of which is S, stands for "start here".
- A finite set of productions of the form

  non-terminal → finite string of terminals and /or non-terminals.

Note

The terminals are designated by small letters, while the non-terminals are designated by capital letters.

There is at least one production that has the non-terminal S as its left side.

Context Free Language (CFL)

The language generated by CFG is called Context Free Language (CFL).

Example

Σ = {a}

productions:

S →aS

S→Λ

Applying production (1) six times and then production (2) once, the word aaaaaa is generated as

S ⇒ aS

⇒ aaS

⇒ aaaS

⇒ aaaaS

⇒ aaaaaS

⇒ aaaaaaS

⇒ aaaaaaΛ

= aaaaaa

It can be observed that prod (2) generates Λ, a can be generated applying prod. (1) once and then prod. (2), aa can be generated applying prod. (1) twice and then prod. (2) and so on. This shows that the grammar defines the language expressed by a*.

Example

Σ = {a}

productions:

S→SS

S→a

S→Λ

This grammar also defines the language expressed by a*.

Note: It is to be noted that Λ is not considered to be terminal. It has a special status. If for a certain non-terminal N, there may be a production N→Λ. This simply means that N can be deleted when it comes in the working string.

Example

Σ = {a,b}

productions:

S→X

S→Y

X→Λ

Y→aY

Y→bY

Y→a

Y→b

All words of this language are of either X-type or of Y-type. *i.e.* while generating a word the first production used is S→X or S→Y. The words of X-type give only Λ, while the words of Y-type are words of finite strings of a' s or b' s or both *i.e.* $(a+b)^+$. Thus the language defined is expressed by (a+b)*.

Example

Σ = {a,b}

productions:

S→aS

S→bS

S→a

S→b

S→Λ

This grammar also defines the language expressed by (a+b)*.

Example

Σ = {a,b}

productions:

S→XaaX

X→aX

X→bX

X→Λ

This grammar defines the language expressed by (a+b)*aa(a+b)*.

Example

Σ = {a,b}

productions:

S → aB

S → bA

A → a

A → aS

A → bAA

B → b

B → bS

B → aBB

This grammar generates the language EQUAL(The language of strings, with number of a’ s equal to number of b’ s).

Note

It is to be noted that if the same non-terminal have more than one productions, it can be written in single line

e.g. S → aS, S → bS, S → Λ can be written as S → aS|bS|Λ

It may also be noted that the productions S → SS|Λ always defines the language which is closed w.r.t.

concatenation *i.e.*the language expressed by RE of type r*. It may also be noted that the production S → SS defines the language expressed by r^+.

Example

Σ = {a,b}

productions:

S → YXY

Y → aY|bY|Λ

X → bbb

It can be observed that, using prod.2, Y generates Λ. Y generates a. Y generates b. Y also generates all thecombinations of a and b. thus Y generates the strings generated by (a+b). It may also be observed that the above CFG generates the language expressed by (a+b)*bbb(a+b)*.

Example

Consider the following CFG

Σ = {a,b}

productions:

S → aSa|bSb|a|b|Λ

The above CFG generates the language PALINDROME. It may be noted that the CFG

S → aSa|bSb|a|b generates the language NON-NULLPALINDROME.

Example

Consider the following CFG

Σ = {a,b}

productions:

S → aSb|ab|Λ

It can be observed that the CFG generates the language $\{a^n b^n : n = 0,1,2,3, \cdots\}$.

Example

Consider the following CFG

S → aXb|bXa

X → aX|bX|Λ

The above CFG generates the language of strings, defined over Σ={a,b}, **beginning and ending in different letters**.

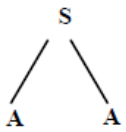
Trees

As in English language any sentence can be expressed by parse tree, so any word generated by the given CFG can also be expressed by the parse tree, *e.g.* consider the following CFG

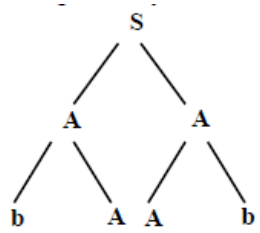
S → AA

A → AAA|bA|Ab|a

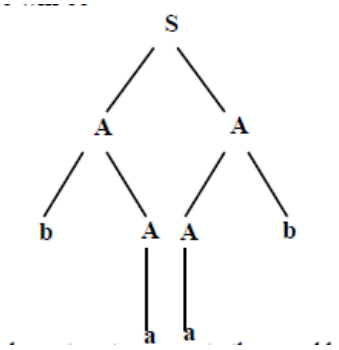
Obviously, baab can be generated by the above CFG. To express the word baab as a parse tree, start with S. Replace S by the string AA, of nonterminals, drawing the downward lines from S to each character of this stringas follows



Now let the left A be replaced by bA and the right one by Ab then the tree will be



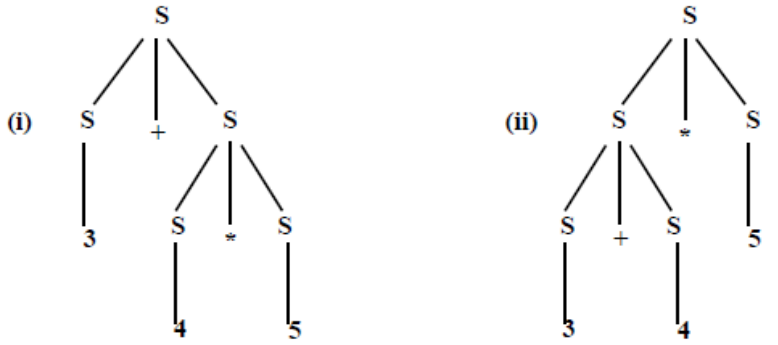
Replacing both A's by a, the above tree will be



Thus the word baab is generated. The above tree to generate the word baab is called **Syntax tree or Generation tree or Derivation tree as well.**

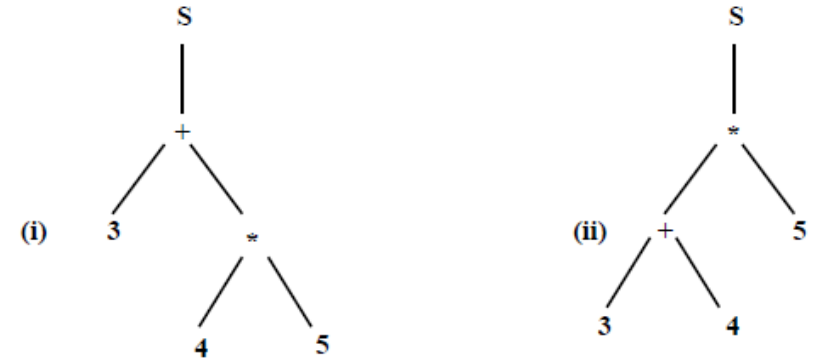
Polish Notation (o-o-o)

There is another notation for arithmetic expressions for the CFG, S→S+S|S*S|number. Consider the following derivation trees



The arithmetic expressions shown by the trees (i) and (ii) can be calculated from the following trees,

Respectively



Here most of the S’ s are eliminated.

The branches are connected directly with the operators. Moreover, the operators + and * are no longer terminals as these are to be replaced by numbers (results).

To write the arithmetic expression, it is required to traverse from the left side of S and going onward around the tree. The arithmetic expressions will be as under

(i) + 3 * 4 5 (ii) * +3 4 5

The above notation is called operator prefix notation.

To evaluate the strings of characters, the first substring (from the left) of the form operator-operand-operand (o-o-o) is found and is replaced by its calculation *e.g.*

+3*4 5 = +3 20 = 23

*+3 4 5 = * 7 5 = 35

It may be noted that 4*5+3 is an infix arithmetic expression, while an arithmetic expression in (o-o-o) form is a prefix arithmetic expression.

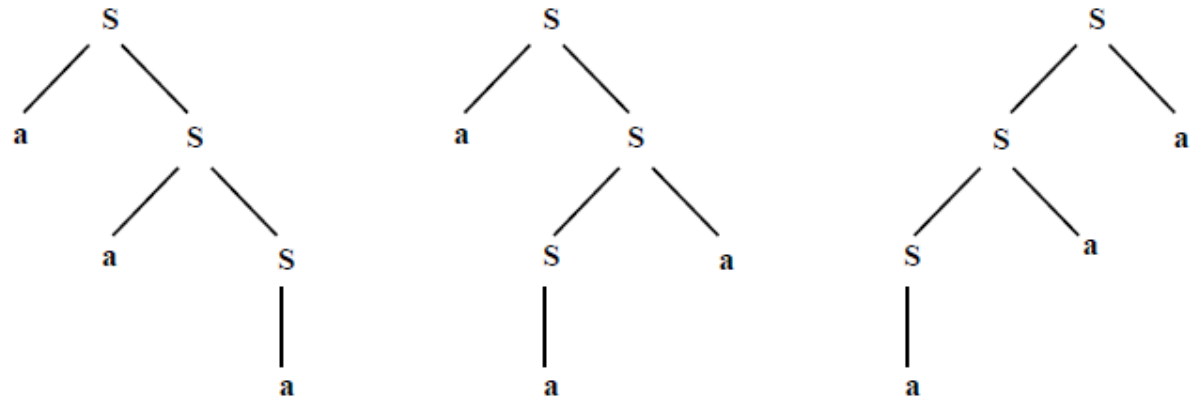
Ambiguous CFG

The CFG is said to be ambiguous if there exists at least one word of it's language that can be generated by the different production trees.

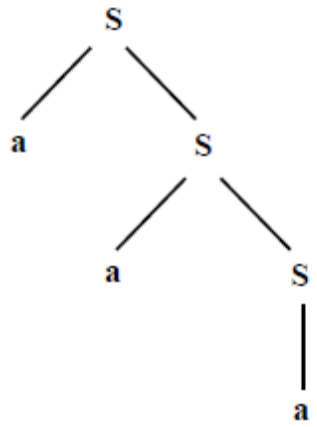
Example: Consider the following CFG

S→aS|Sa|a

The word aaa can be generated by the following three different trees



Thus the above CFG is ambiguous, while the CFG, S→aS|a is not ambiguous as neither the word aaa nor any other word can be derived from more than one production trees. The derivation tree for aaa is as follows



Example

Consider the following CFG

S → aS | bS | aaS | Λ

It can be observed that the word aaa can be derived from more than one production trees. Thus, the above CFG is ambiguous. This ambiguity can be removed by removing the production S → aaS

Example

Consider the CFG of the language PALINDROME

S→aSa|bSb|a|b|Λ

It may be noted that this CFG is unambiguous as all the words of the language PALINDROME can only be

generated by a unique production tree.

It may be noted that if the production S → aaSaa is added to the given CFG, the CFG thus obtained will be no more unambiguous.

Total language tree

For a given CFG, a tree with the start symbol S as its root and whose nodes are working strings of terminals and non-terminals. The descendants of each node are all possible results of applying every production to the working string. This tree is called **total language tree**.