

CS402 Short Notes final term

Lec 23 to 45

BY vuonlinehelp.blogspot.com

YOUTUBE LINK:

WEBSITE LINK:

[HTTPS://VUONLINEHELP.BLOGSPOT.COM/](https://vuonlinehelp.blogspot.com/)

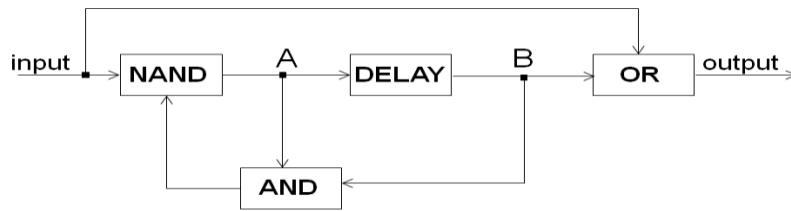
IMPORTANT NOTE

You should never copy paste the same answer which we are providing. Please do make some changes in the solution file. If you appreciate our work, please share our website vuonlinehelp.blogspot.com with your friends.

آپ کو کبھی بھی وہی جواب کاپی پیسٹ نہیں کرنا چاہیے جو ہم فراہم کر رہے ہیں۔ براہ کرم حل فائل میں کچھ تبدیلیاں کریں۔ اگر آپ ہمارے کام کی تعریف کرتے ہیں تو براہ کرم ہماری ویب سائٹ vuonlinehelp.blogspot.com کو اپنے دوستوں کے ساتھ شیئر کریں

LECTURE NO. 23 CS402

Consider the following sequential circuit



The following four types of boxes are used in this circuit

1. NAND box (NOT AND): For given inputs, it provides the complement of Boolean AND output.
2. DELAY box (Flip Flop box): It delays the transmission of signal along the wire by one step (clock pulse).
3. OR box: For given inputs, it provides the Boolean OR output.
4. AND box: For the given inputs, it provides the Boolean AND output.

The current in the wire is indicated by 1 and 0 indicates the absence of the current.

There are two points A and B w.r.t. to which following four states of the machine are identified according to the presence and absence of current at these points *i.e.*

- 1) $q_0(A=0, B=0) = (0,0)$
- 2) $q_1(A=0, B=1) = (0,1)$
- 3) $q_2(A=1, B=0) = (1,0)$
- 4) $q_3(A=1, B=1) = (1,1)$

The operation of the circuit is such that the machine changes its state after reading 0 or 1. The transitions are determined using the following relations

new B = old A

new A = (input) NAND (old A AND old B)

output = (input) OR (old B)

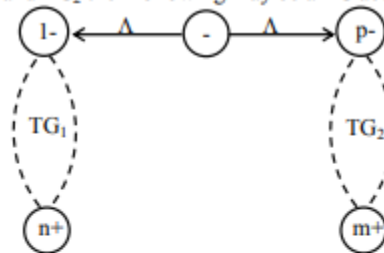
Old state	Inputting 0		Inputting 1	
	State	Output	State	Output
$q_0 \equiv (0,0)$	$(1,0) \equiv q_2$	0	$(1,0) \equiv q_2$	1
$q_1 \equiv (0,1)$	$(1,0) \equiv q_2$	1	$(1,0) \equiv q_2$	1
$q_2 \equiv (1,0)$	$(1,1) \equiv q_3$	0	$(1,1) \equiv q_3$	1
$q_3 \equiv (1,1)$	$(1,1) \equiv q_3$	1	$(0,1) \equiv q_1$	1

Regular languages

Any language that can be expressed by a RE is said to be regular language, so if L_1 and L_2 are regular languages then $L_1 + L_2$, L_1L_2 and L_1^* are also regular languages. This fact can be proved by the following two methods.

- **By Regular Expressions:** As discussed earlier that if r_1, r_2 are regular expressions, corresponding to the languages L_1 and L_2 then the languages $L_1 + L_2$, L_1L_2 and L_1^* generated by $r_1 + r_2$, r_1r_2 and r_1^* , are also regular languages.
- **By TGs:** If L_1 and L_2 are regular languages then L_1 and L_2 can also be expressed by some REs as well and hence using Kleene's theorem, L_1 and L_2 can also be expressed by some TGs. Following are the methods showing that there exist TGs corresponding to $L_1 + L_2$, L_1L_2 and L_1^* .

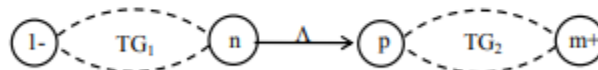
If L_1 and L_2 are expressed by TG_1 and TG_2 then following may be a TG accepting $L_1 + L_2$



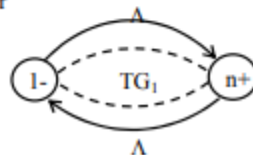
If L_1 and L_2 are expressed by the following TG_1 and TG_2



then following may be a TG accepting L_1L_2



also a TG accepting L_1^* may be as under



Complement of a language IMP

Let L be a language defined over an alphabet Σ , then the language of strings, defined over Σ , not belonging to L , is called Complement of the language L , denoted by L^c or $L^/$. To describe the

complement of a language, it is very important to describe the alphabet of that language over which the language is defined. For a certain language L , the complement of L^c is the given language L i.e. $(L^c)^c = L$

Theorem:

If L is a regular language then, L^c is also a regular language.

Proof: since L is a regular language, so by Kleene's theorem, there exists an FA, say F , accepting the language L . Converting each of the final states of F to non-final states and old non-final states of F to final states. FA, thus, obtained will reject every string belonging to L and will accept every string, defined over Σ , not belonging to L . This shows that the new FA accepts the language L^c . Hence using Kleene's theorem L^c can be expressed by some RE. Thus L^c is regular.

Theorem:

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

Proof:

Using De-Morgan's law for sets

$$(L_1^c \cup L_2^c)^c = (L_1^c)^c \cap (L_2^c)^c = L_1 \cap L_2$$

Since L_1 and L_2 are regular languages, so are L_1^c and L_2^c . L_1^c and L_2^c being regular provide that $L_1^c \cup L_2^c$ is also regular language and so $(L_1^c \cup L_2^c)^c = L_1 \cap L_2$, being complement of regular language is regular language.

Lecture NO. 25 CS402

Statement:

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

Proof:

Using De-Morgan's law for sets

$$(L_1^c \cup L_2^c)^c = (L_1^c)^c \cap (L_2^c)^c = L_1 \cap L_2$$

Since L_1 and L_2 are regular languages, so are L_1^c and L_2^c . L_1^c and L_2^c being regular provide that $L_1^c \cup L_2^c$ is also regular language and so $(L_1^c \cup L_2^c)^c = L_1 \cap L_2$, being complement of regular language is regular language.

FA corresponding to intersection of two regular languages (short method)

Let FA_3 be an FA accepting $L_1 \cap L_2$, then the initial state of FA_3 must correspond to the initial state of FA_1 and the initial state of FA_2 .

Since the language corresponding to $L_1 \cap L_2$ is the intersection of corresponding languages L_1 and L_2 , consists of the strings belonging to both L_1 and L_2 , therefore a final state of FA_3 must correspond to a final state of FA_1 and FA_2 . Following is an example regarding short method of finding an FA corresponding to the intersection of two 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. It is to be noted that a nonregular language, by Kleene's theorem, can't be accepted by any FA or TG.

Lecture NO. 26 CS402

Pumping Lemma

Statement: Let L be any infinite regular language (that has infinite many words), defined over an alphabet Σ then there exist three strings x , y and z belonging to Σ^* (where y is not the null string) such that all the strings of the form $xy^n z$ for $n=1, 2, 3 \dots$ are the words in L .

Proof: If L is a regular language, then according to Kleene's theorem, there exists an FA, say, F that accepts this language. Now F , by definition, must have finite no of states while the language has infinitely many words, which shows that there is no restriction on the length of words in L , because if there were such restriction then the language would have finite many words.

Let w be a word in the language L , so that the length of word is greater than the number of states in F . In this case the path generated by the word w , is such that it cannot visit a new state for each letter *i.e.* there is a circuit in this path.

The word w , in this case, may be divided into three parts

1. The substring which generates the path from initial state to the state which is revisited first while reading the word w . This part can be called x and x can be a null string.
2. The substring which generates the circuit starting from the state which was lead by x . This part can be called as y which cannot be null string.
3. The substring which is the remaining part of the word after y , call this part as z . It may be noted that this part may be null string as the word may end after y or z part may itself be a circuit.

Thus the word may be written as $w = xyz$ where x, y and z are the strings, also y can't be a null string.

Now this is obvious that, looping the circuit successively, the words $xyyz, xyxyz, xyxyyz,$ will also be accepted by this FA i.e. $xy^n z, n=1, 2, 3 \dots$ will be words in L .

In the above theorem, it is not affected if the z -part has circuit. To prove the theorem it is only to find a circuit and then looping that circuit, is all that is needed. While looping the circuit the volume of the string y (or z) is pumped, so the theorem is also called the Pumping lemma.

If the pumping lemma is applied directly on the language $L = \{a^n b^n : n=0,1,2,3,\dots\}$, it can be observed that for the word $w = (aaa)(aaaabbbb)(bbb)$ where $x = aaa, y = aaaabbbb$ and $z = bbb$

Example

Consider the language EQUAL, of strings, defined over $\Sigma=\{a,b\}$, with number of a 's equal to number of b 's, i.e. $EQUAL = \{\Lambda, ab, aabb, abab, baba, abba, \dots\}$ From the definition of EQUAL, it is clear that $\{a^n b^n\} = a^* b^* \cap EQUAL$ Obviously $a^* b^*$ defines a regular language while $\{a^n b^n\}$ has been proved nonregular. Using the theorem that intersection of two regular languages is, regular; it can be proved that the EQUAL is not regular.

Lecture NO. 27 CS402

Pumping Lemma version II

Let L be an infinite language accepted by a finite automaton with N states, then for all words w in L that have length more than N , there are strings x, y and z (y being non-null string) and $\text{length}(x) + \text{length}(y) \leq N$ s.t. $w = xyz$ and all strings of the form $xy^N z$ are in L for $n = 1, 2, 3, \dots$

Proof The lemma can be proved, considering the following examples

Example PALINDROME

Consider the language PALINDROME which is obviously infinite language. It has already been shown that the PALINDROME satisfies pumping lemma version I (previous version). To check whether the new version of pumping lemma still holds in case of the PALINDROME, let the PALINDROME be a regular language and be accepted by an FA of 78 states. Consider the word $w = a^{85} b a^{85}$.

$\text{length}(x) + \text{length}(y) \leq 78$, which shows that the substring xy is consisting of a 's and $xyyz$ will become $a^{\text{more than } 85} b a^{85}$ which is not in PALINDROME.

Example PRIME

Consider the language PRIME, of strings defined over $\Sigma=\{a\}$, as

$\{a^p : p \text{ is prime}\}$, i.e.

$\text{PRIME} = \{aa, aaa, aaaaa, aaaaaaa, \dots\}$

To prove this language to be nonregular, suppose contrary, i.e. PRIME is a regular language, then there exists an FA accepts the language PRIME. Let the number of states of this machine be 345 and choose a word w from PRIME with length more than 345, say, 347 i.e. the word $w = a^{347}$

Consider $n=348$, then $xy^n z = xy^{348}z = xy^{347}yz$. Since x, y and z consist of a 's, so the order of x, y, z does not matter i.e.

Myhill Nerode theorem

The Myhill Nerode theorem is a fundamental result coming down to the theory of languages. This theory was proven by John Myhill and Anil Nerode in 1958. It is used to prove whether or not a language L is regular and it is also used for minimization of states in DFA(Deterministic Finite Automata).

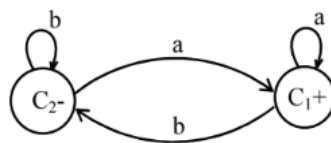
To understand this theorem, first we need to understand what Indistinguishability is : Given a language L and x, y are string over Σ^* , if for every string $z \in \Sigma^*$, $xz, yz \in L$ or $xz, yz \notin L$ then x and y are said to be indistinguishable over language L . Formally, we denote that x and y are indistinguishable over L by the following notation : $x \equiv_L y$. $\equiv_L$ is an equivalence relation as it is :

Example

Consider the language L of strings, defined over $\Sigma=\{a,b\}$, ending in a . It can be observed that L partitions Σ^* into the following two classes

C_1 = set of all strings ending in a .

C_2 = set of all strings not ending in a .



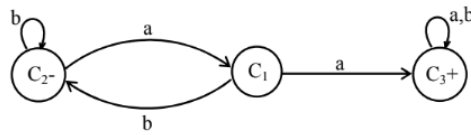
Example

Consider the language L of strings, defined over $\Sigma=\{a,b\}$, containing double a . It can be observed that L partitions Σ^* into the following three classes

C_1 = set of all strings without aa but ending in a.

C_2 = set of $\wedge$ and all strings without aa but ending in b.

C_3 = set of all strings containing aa.



Lecture N0. 28 CS402

Examples of Myhill Nerode theorem,

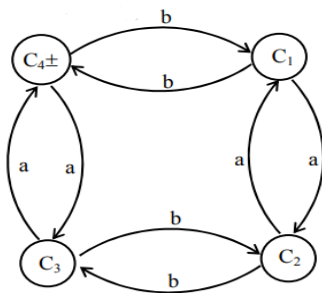
Consider the language L which is EVEN-EVEN, defined over $\Sigma = \{a, b\}$. It can be observed that L partitions Σ^* into the following four classes

C_1 = set of all strings with even number of a's and odd number of b's.

C_2 = set of all strings with odd number of a's and odd number of b's.

C_3 = set of all strings with odd number of a's and even number of b's.

C_4 = set of all strings with even number of a's and even number of b's.



Consider the language $L = \{w \in \{a, b\}^* : \text{length}(w) \geq 2, w \text{ ends in either } ab \text{ or } ba\}$. It can be observed that L partitions Σ^* into the following seven classes

C_1 = set containing only null string.

C_2 = set containing only letter a.

C_3 = set containing only letter b.

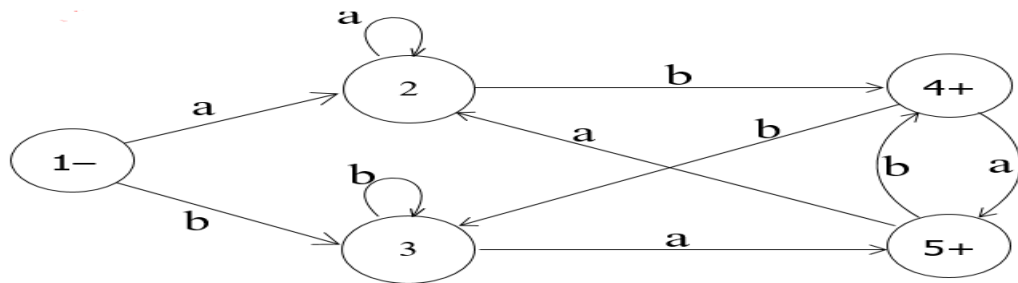
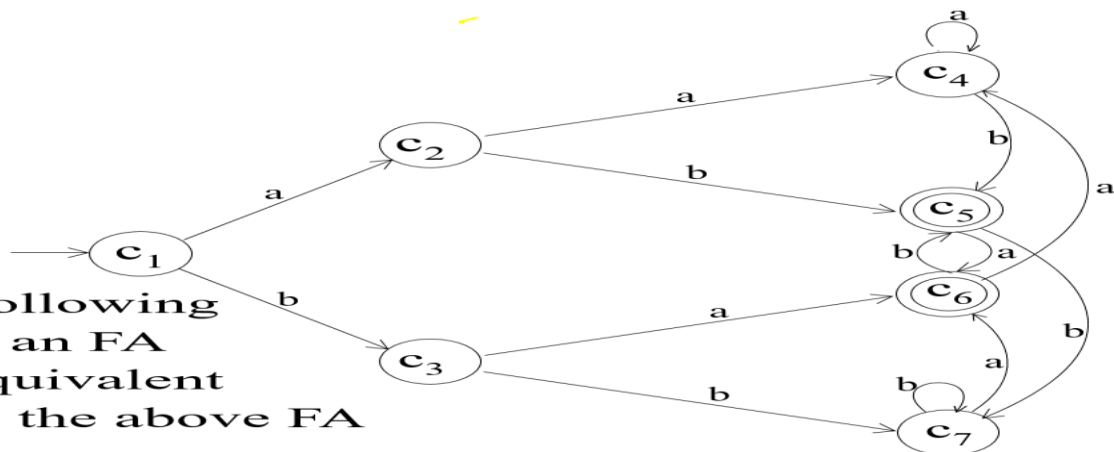
C_4 = set of strings ending in aa.

C_5 = set of strings ending in ab.

C_6 = set of strings ending in ba.

C_7 = set of strings ending in bb.

Following
is an FA
equivalent
to the above FA



Quotient of a language into another

The quotient of a regular language with any other language is regular. The quotient of a context free language with a regular language is context free. The quotient of two context free languages can be any recursively enumerable language. The quotient of two recursively enumerable languages is recursively enumerable.

Denoted by $Q=R/P$ if $PQ=R$.

Pseudo theorem

For three languages P, Q and R , while $PQ=R$ the language Q must be regular if both P and R are regular. (Note: It is to be noted that since this theorem is not true, so the theorem is called pseudo theorem.) The theorem can be disproved by contradiction i.e. supposing that Q is regular.

Let $P=a^*$, Q be the product of $\{a^n b^n : n=0,1,2,\dots\}$ and b^* then $PQ=a^*\{a^n b^n\}b^*=a^*b^*=R$ which shows that R is regular.

Prefixes of a language in another language

If two languages R and Q are given, then the language the prefixes of Q in R denoted by $\text{Pref}(Q \text{ in } R)$ is the set of strings of letters that, when concatenated to the front of

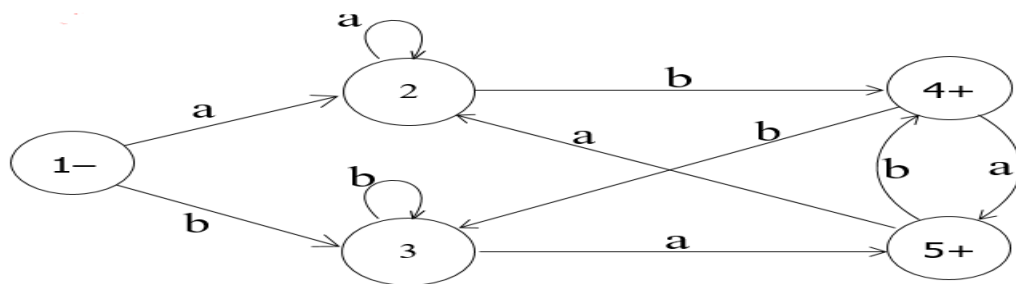
some word in Q to produce some word in R i.e. $\text{Pref}(Q \text{ in } R) = \text{the set of all strings } p \text{ such that there exists words } q \text{ in } Q \text{ and } w \text{ in } R \text{ such that } pq = w.$

Lecture N0. 29 CS402

Theorem

If R is regular language and Q is any language (regular/ no regular), then $\text{Pref}(Q \text{ in } R)$ is regular. If this state traces out a path ending up in a final state for any of the words of Q , mark this state with certain color. Now build a new FA with some marked states by considering the initial state that of original FA and final states which are marked.

Consider the languages $Q = \{aba, abb\}$ and $R = \{w \in \{a,b\}^* : \text{length}(w) \geq 2, w \text{ ends in either } ab \text{ or } ba\}$, where R may be accepted by the following FA.



Decidability

A problem is said to be effectively solvable if there exists an algorithm that provides the solution in finite number of steps e.g.

Decision procedure

If an effectively solvable problem has answer in yes or no, then this solution is called decision procedure.

Decidable problem

A problem that has decision procedure is called decidable problem e.g. the following problems

1. The two regular expressions define the same language.
2. The two FAs are equivalent.

Determining whether the two languages are equivalent or not?

Problems that solve in finite steps and there is yes or no

Method 1

For FA corresponding to $(L_1 L_2^c) \cup (L_1^c L_2)$, the regular expression can be determined that defines the language accepted by this FA. From that regular expression one can determine whether this regular expression defines any word or not? Following are the steps to be followed

1. Remove all *s from the regular expression.
2. Separate the right part of + and the plus itself.

The regular expression thus obtained if contains atleast one word then the language is not empty otherwise the language is empty.

Method 2

To examine whether a certain FA accepts any words, it is required to seek the paths from initial to final state. But in large FA with thousands of states and millions of directed edges. Without an effective procedure it is impossible to find a path from initial to final state. Following are the steps of this procedure

1. Mark the initial state.
2. For every marked state follow each edge that leads out of it and marked the concatenating states and delete these edges.
3. Repeat step 2 until no new state is marked.

If any of the final states are marked then the FA accepts some word, otherwise not. Following is an example regarding this method.

Lecture N0. 30 ^{CS402}

Method 3

If the FA has N states, then test the words of length less than N. If no word is accepted by this FA, then it will accept no word. It is to be noted that all the methods discussed above, to determine whether an FA accepts certain word, are effective procedures.

Observation

If a regular expression contains $*$ then it may define an infinite language, with the exception Λ^* as $\Lambda^* = \Lambda$ e.g.

$(\Lambda + a \Lambda^*)(\Lambda^* + \Lambda)^*$ defines finite language.

While $(\Lambda^* + a \Lambda^*)^*(\Lambda^* + \Lambda)^*$ defines an infinite language.

There are so many ways to decide whether an FA accepts a finite language or an infinite. Following is a theorem regarding this

Let F be an FA having N states

1. If F accepts a word w s.t.

$N \wedge \text{length}(w) < 2N$ then F accepts infinite language.

2. If F accepts an infinite language then there are some words w s.t.

$N \wedge \text{length}(w) < 2N$ the first part can be proved, using the pumping lemma version II.

There is an effective procedure to decide whether an FA accepts finite or infinite language. For a machine with N number of states, the total number of strings to be tested, defined over an alphabet of m letters, is $m^N + m^{N+1} + m^{N+2} + \dots + m^{2N-1}$.

Lecture N0. 31 CS402

Context Free Grammar (CFG)

The earliest computers accepted no instructions other than 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

$$S = \sqrt{\frac{(8-0)^2 + (7-10)^2 + (11-10)^2}{2}}$$

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

The high level language is converted into assembly language codes by a program called compiler. 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. $X = B + 10$, $X = B + 999$

Remark

In general, the rules of computer language grammar, are all syntactic and not semantic.

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

1. An alphabet Σ of letters called **terminals** from which the strings are formed, that will be the words of the language.
2. A set of symbols called **non-terminals**, one of which is S , stands for "start here".
3. A finite set of productions of the form Non-terminal $\rightarrow$ finite string of terminals and /or non-terminals.

Context Free Language (CFL)

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

Example:

$\Sigma = \{a\}$ productions:

1. $S \rightarrow aS$

2. $S \rightarrow \Lambda$

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

$S \Rightarrow aS$
 $\Rightarrow aaS$
 $\Rightarrow aaaS$
 $\Rightarrow aaaaS$
 $\Rightarrow aaaaaS$
 $\Rightarrow aaaaaaS$
 $\Rightarrow aaaaaa\Lambda$
 $= aaaaaa$

Lecture N0. 32 CS402

Context Free Grammar, Terminals, non-terminals, productions, CFG, context free language, examples.

Example

$\Sigma = \{a,b\}$

Productions:

1. $S \rightarrow SS$
2. $S \rightarrow XS$
3. $S \rightarrow \Lambda$
4. $S \rightarrow YSY$
5. $X \rightarrow aa$
6. $X \rightarrow bb$
7. $Y \rightarrow ab$
8. $Y \rightarrow ba$

This grammar generates EVEN-EVEN language.

Example

Consider the following CFG

$S \rightarrow SS | XaXaX | \Lambda$

$X \rightarrow bX | \Lambda$

It can be observed that, using prod.2, X generates Λ . X generates any number of b's. Thus X generates the strings generated by b^* . It may also be observed that the above CFG generates the language expressed by $(b^*ab^*ab^*)^*$.

Example

Consider the following CFG $\Sigma = \{a,b\}$ productions: $S \rightarrow aSa | bSb | a | b | \Lambda$ The above CFG generates the language PALINDROME. It may be noted that the CFG

$S \rightarrow aSa | bSb | a | b$ generates the language NON-NULLPALINDROME.

Example

Consider the following CFG $\Sigma = \{a,b\}$ productions: $S \rightarrow aSb | ab | \Lambda$ It can be observed that the CFG generates the language $\{a^n b^n : n = 0, 1, 2, 3, \dots\}$. It may also be noted that the language $\{a^n b^n : n = 1, 2, 3, \dots\}$ can be generated by the CFG,

$S \rightarrow aSb | ab$

Example

Consider the following CFG $S \rightarrow aXb | bXa$ $X \rightarrow aX | bX | \Lambda$ The above CFG generates the language of strings, defined over $\Sigma = \{a,b\}$, beginning and ending in different letters.

Trees

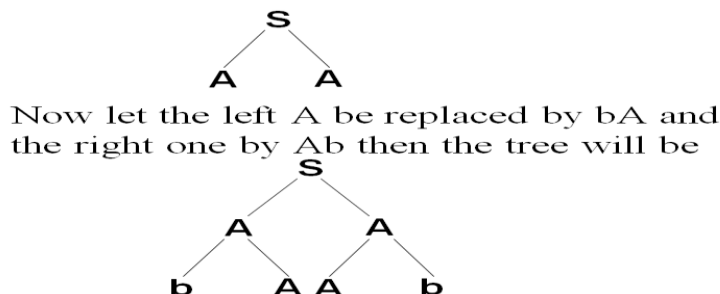
As in English language any sentence can be expressed by parse tree, so any word generated by the given CFG can also be expressed by the parse tree, *e.g.*

Consider the following CFG

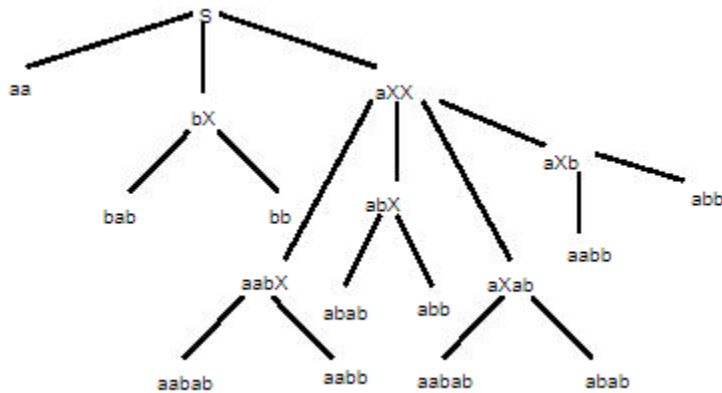
$S \rightarrow AA$

$A \rightarrow AAA | bA | Ab | a$

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 string as follows



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



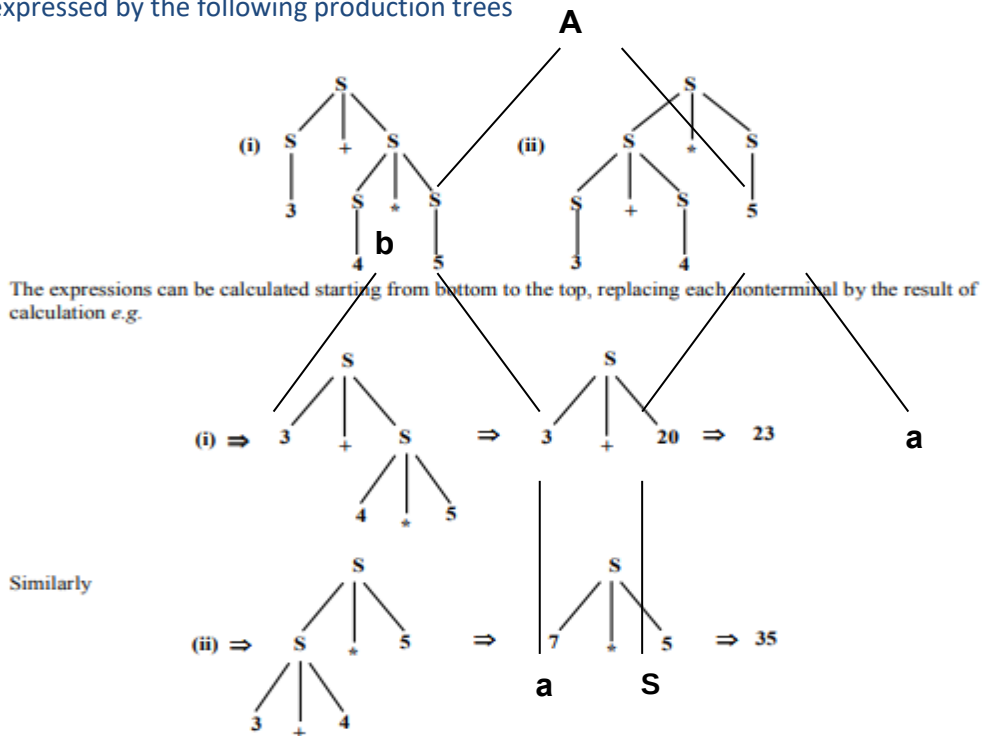
Thus the word baab is generated. The above tree to generate the word baab is called **Syntax tree** or **Generation tree** or **Derivation tree** as well.

Lecture N0. 33 CS402

Consider the following CFG

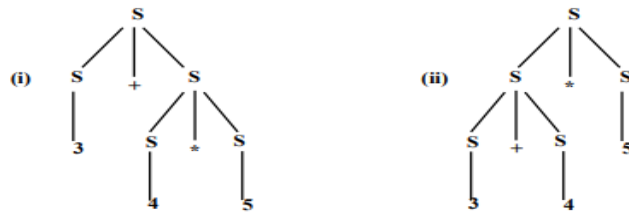
$S \rightarrow S+S | S*S | \text{number}$

Where S and number are non-terminals and the operators behave like terminals. The above CFG creates ambiguity as the expression $3+4*5$ has two possibilities $(3+4)*5=35$ and $3+(4*5)=23$ which can be expressed by the following production trees

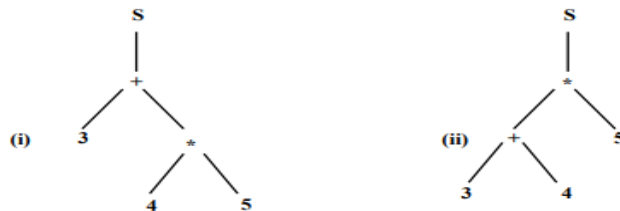


Polish Notation (o-o-o)

There is another notation for arithmetic expressions for the CFG, $S \rightarrow S+S | S*S | \text{number}$. Consider the following derivation trees



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



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

$$(i) + 3 * 4 5 \quad (ii) * + 3 4 5$$

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.

$$(i) \quad + 3 * 4 5 = + 3 20 = 23$$

$$(ii) \quad * + 3 4 5 = * 7 5 = 35$$

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

Ambiguous CFG

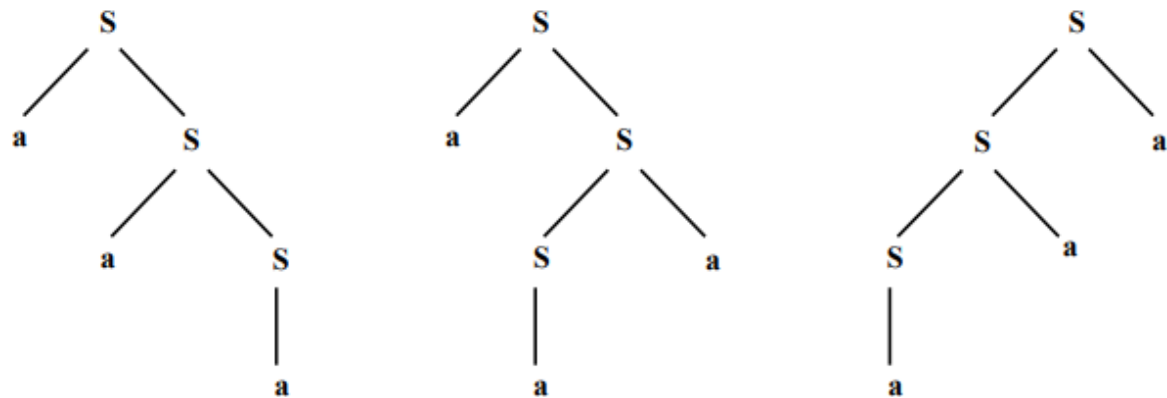
The CFG is said to be ambiguous if there exists atleast one word of its language that can be generated by the different production trees.

Example:

Consider the following CFG

$$S \rightarrow aS | Sa | a$$

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



Lecture N0. 34 CS402

Total language tree

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.

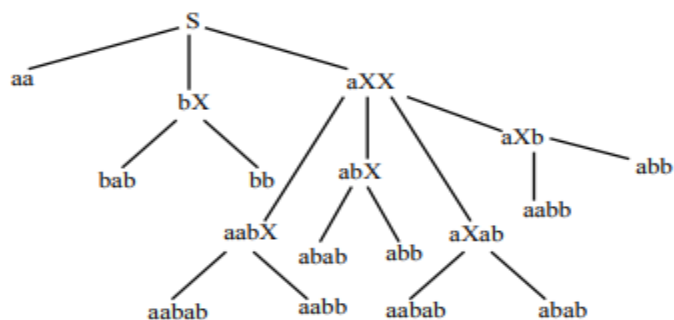
Example

Consider the following CFG

$S \rightarrow aa|bX|aXX$

$X \rightarrow ab|b$

Then the total language tree for the given CFG may be



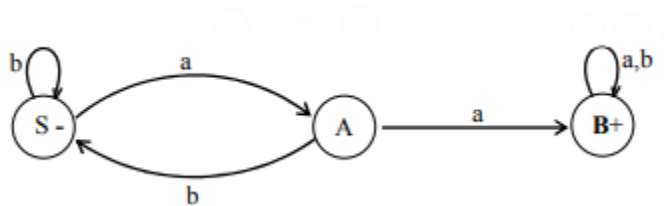
It may be observed from the above total language tree that dropping the repeated words, the language generated by the given CFG is $\{aa, bab, bb, aabab, aabb, abab, abb\}$.

Regular Grammar

All regular languages can be generated by CFGs. Some nonregular languages can be generated by CFGs but not all possible languages can be generated by CFG.

Example

Consider the language L expressed by $(a+b)^*aa(a+b)^*$ i.e. the language of strings, defined over $\Sigma = \{a,b\}$, containing aa.



$S \rightarrow bS \mid aA$
 $A \rightarrow aB \mid bS$
 $B \rightarrow aB \mid bB \mid \Lambda$

Semiword

A semiword is a string of terminals (may be none) concatenated with exactly one nonterminal on the right.

A semiword, in general, is of the following form

(Terminal)(Terminal)... (Terminal)(Nonterminal)

Word

A word is a string of terminals. Λ is also a word.

Theorem

If every production in a CFG is one of the following forms

Nonterminal $\rightarrow$ semiword

Nonterminal $\rightarrow$ word

then the language generated by that CFG is regular.

Regular grammar

A CFG is said to be a regular grammar if it generates the regular language i.e. a CFG is said to be a regular grammar in which each production is one of the two forms

1. Nonterminal $\rightarrow$ semiword
2. Nonterminal $\rightarrow$ word

Examples

The CFG $S \rightarrow aaS | bbS | \Lambda$ is a regular grammar. It may be observed that the above CFG generates the language of strings expressed by the RE $(aa+bb)^*$.

TG for Regular Grammar

For every regular grammar there exists a TG corresponding to the regular grammar. Following is the method to build a TG from the given regular grammar.

1. Define the states, of the required TG, equal in number to that of nonterminals of the given regular grammar. An additional state is also defined to be the final state. The initial state should correspond to the nonterminal S.
2. For every production of the given regular grammar, there are two possibilities for the transitions of the required TG.
 - If the production is of the form nonterminal $\rightarrow$ semiword, then transition of the required TG would start from the state corresponding to the nonterminal on the left side of the production and would end in the state corresponding to the nonterminal on the right side of the production, labeled by string of terminals in semiword.
 - If the production is of the form nonterminal $\rightarrow$ word, then transition of the TG would start from the state corresponding to nonterminal on the left side of the production and would end on the final state of the TG, labeled by the word. Following is an example in this regard.

Lecture NO. 35 CS402

Null Production

The production of the form nonterminal $\rightarrow \Lambda$ is said to be *null production*.

Example:

Consider the following CFG

$S \rightarrow aA | bB | \Lambda$, $A \rightarrow aa | \Lambda$, $B \rightarrow aS$

Here $S \rightarrow \Lambda$ and $A \rightarrow \Lambda$ are null productions.

Following is a note regarding the null productions

Nullable Production

A production is called nullable production if it is of the form $N \rightarrow \Lambda$ or there is a derivation that starts at N and leads to Λ .

$N_1 \rightarrow N_2$, $N_2 \rightarrow N_3$, $N_3 \rightarrow N_4$, ..., $N_n \rightarrow \Lambda$, where N , N_1 , N_2 , ..., N_n are non-terminals.

Unit production

The productions of the form nonterminal $\rightarrow$ one nonterminal, is called the unit production. Following is an example showing how to eliminate the unit productions from a given CFG.

Chomsky Normal Form

If a CFG has only productions of the form
nonterminal $\rightarrow$ string of two nonterminals

or

nonterminal $\rightarrow$ one terminal

then the CFG is said to be in Chomsky Normal Form (CNF)

Lecture N0. 36 ^{CS402}

Leftmost and Rightmost Derivation of a String

- **Leftmost derivation** – A leftmost derivation is obtained by applying production to the leftmost variable in each step.
- **Rightmost derivation** – A rightmost derivation is obtained by applying production to the rightmost variable in each step.

Example

Let any set of production rules in a CFG be

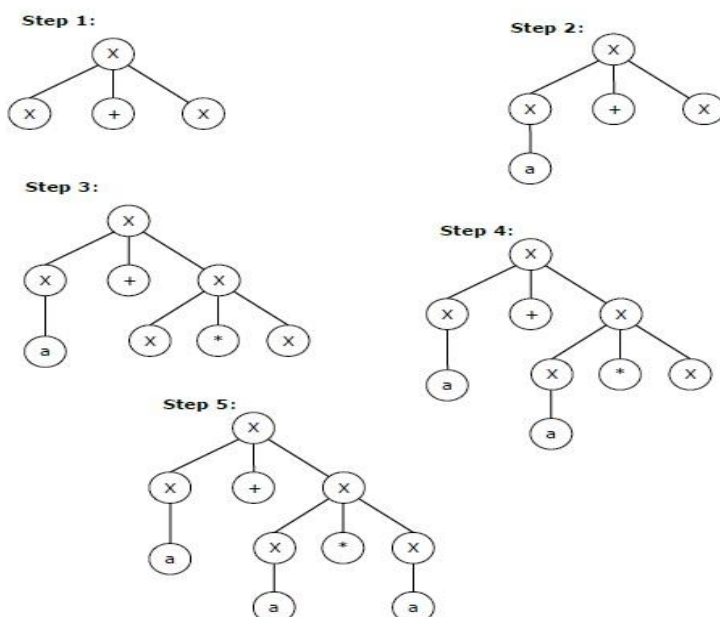
$$X \rightarrow X+X \mid X*X \mid X \mid a$$

over an alphabet $\{a\}$.

The leftmost derivation for the string "a+a*a" may be –

$$X \rightarrow X+X \rightarrow a+X \rightarrow a+X*X \rightarrow a+a*X \rightarrow a+a*a$$

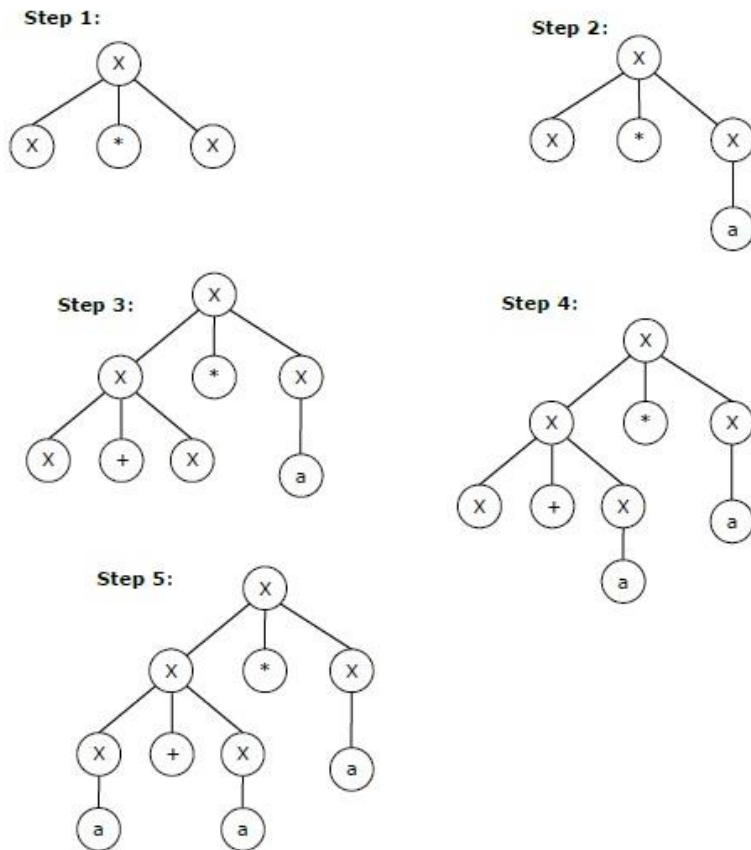
The stepwise derivation of the above string is shown as below –



The rightmost derivation for the above string "a+a*a" may be –

$X \rightarrow X * X \rightarrow X * a \rightarrow X + X * a \rightarrow X + a * a \rightarrow a + a * a$

The stepwise derivation of the above string is shown as below –



Left and Right Recursive Grammars

In a context-free grammar G, if there is a production in the form $X \rightarrow Xa$ where X is a non-terminal and 'a' is a string of terminals, it is called a **left recursive production**. The grammar having a left recursive production is called a **left recursive grammar**.

And if in a context-free grammar G, if there is a production in the form $X \rightarrow aX$ where X is a non-terminal and 'a' is a string of terminals, it is called a **right recursive production**. The grammar having a right recursive production is called a **right recursive grammar**.

Lecture N0. 37 CS402

A new format for Fas

Machine that defined are more powerful and can be constructed with the help of FA's with new format. It can be accept non-regular languages also

Input TAPE

The part of an FA, where the input string is placed before it is run, is called the input TAPE.

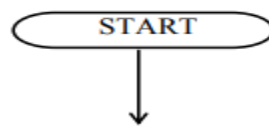
The input TAPE is supposed to accommodate all possible strings. The input TAPE is partitioned with cells, so that each letter of the input string can be placed in each cell. The input string abbaa is shown in the following input TAPE.

Cell i	Cell ii	Cell iii					
a	b	b	a	a	Δ	Δ	.

The character Δ indicates a blank in the TAPE.

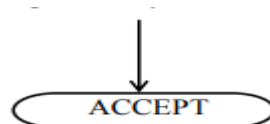
The START state

This state is like initial state of an FA and is represented by



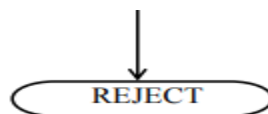
An ACCEPT state

This state is like a final state of an FA and is expressed by



A REJECT state

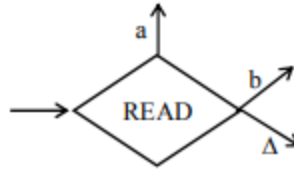
This state is like dead-end non final state and is expressed by



It may be noted that the ACCEPT and REJECT states are called the halt states.

A READ state

This state is to read an input letter and lead to some other state. The READ state is expressed by



PUSHDOWN STACK or PUSHDOWN STORE

PUSHDOWN STACK is a place where the input letters can be placed until these letters are referred again. It can store as many letters as one can in a long column. Initially the **STACK** is supposed to be empty each of its storage location contains a blank.

PUSH

A **PUSH** operator adds a new letter at the top of **STACK** if the letters a, b, c and d are pushed to the **STACK** that was initially blank, the **STACK** can be shown as

STACK

d
c
b
a
Δ
-

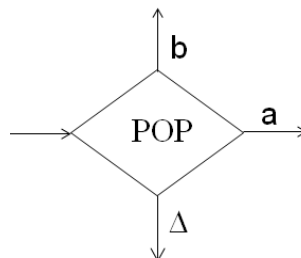
The **PUSH** state is expressed by



When a letter is pushed, it replaces the existing letter and pushes it one position below.

POP and STACK

POP is an operation that takes out a letter from the top of the **STACK**. The rest of the letters are moved



one location up. **POP** state is expressed as

It may be noted that popping an empty **STACK** is like reading an empty **TAPE**, i.e. popping a blank character Δ . It may also be noted that when the new format of an FA contains **PUSH** and **POP** states, it is called **PUSHDOWN Automata** or **PDAs**.

Lecture N0. 38 ^{CS402}

Formal Definition of FA

A finite automaton is a collection of 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where:

Q : finite set of states

Σ : finite set of the input symbol

q_0 : initial state

F : final state

δ : Transition function

Nondeterministic PDA

Like TGs and NFAs, if in a PDA there are more than one outgoing edges at READ or POP states with one label, then it creates non determinism and the PDA is called nondeterministic PDA. In nondeterministic PDA no edge is labeled by string of terminals or non-terminals, like that can be observed in TGs. Also if there is no edge for any letter to be read from the TAPE, the machine crashes and the string are rejected. In nondeterministic PDA a string may trace more than one path. If there exists at least one path traced by a string leading to ACCEPT state, then the string is supposed to be accepted, otherwise rejected.

PUSHDOWN AUTOMATON (PDA) [including the possibility of non-determinism]

Pushdown Automaton (PDA), consists of the following

1. An alphabet Σ of input letters.
2. An input TAPE with infinite many locations in one direction. Initially the input string is placed in it starting from first cell, the remaining part of the TAPE is empty.
3. An alphabet Γ of STACK characters.
4. A pushdown STACK which is initially empty, with infinite many locations in one direction. Initially the STACK contains blanks.
5. One START state with only one out-edge and no in-edge.
6. Two halt states i.e. ACCEPT and REJECT states, with in-edges and no out-edges.
7. A PUSH state that introduces characters onto the top of the STACK.
8. A POP state that reads the top character of the STACK, (may contain more than one out-edges with same label).
9. A READ state that reads the next unused letter from the TAPE, (may contain more than one out-edges with same label).

Lecture N0. 39 ^{CS402}

PDA corresponding to CFG Theorem

Corresponding to any CFG there exists a PDA accepting the language generated by the CFG.

An algorithm has already been discussed to convert the CFG in CNF, so the PDA can be constructed corresponding to the CFG. As the CFG in CNF generates all the non-null words of the corresponding CFL, so accepting the null string (if it is contained in the CFL), can be managed separately.

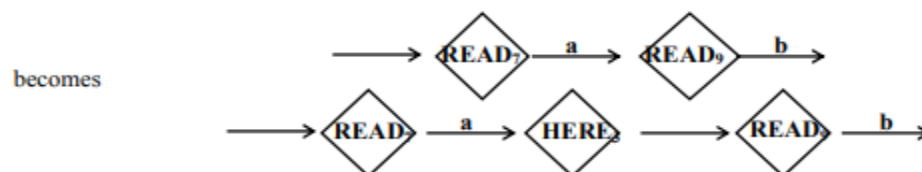
Lecture NO. 40 CS402

Theorem

Given a PDA that accepts the language L, there exists a CFG that generates exactly L.

Before the CFG corresponding to the given PDA is determined, the PDA is converted into the standard form which is called the conversion form. Before the PDA is converted into conversion form a new state HERE is defined which is placed in the middle of any edge.

Like READ and POP states, HERE states are also numbered e.g.



Conversion form of PDA important long /short

A PDA is in conversion form if it fulfills the following conditions:

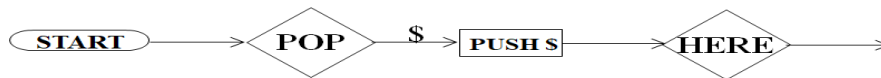
There is only one ACCEPT state.

There are no REJECT states.

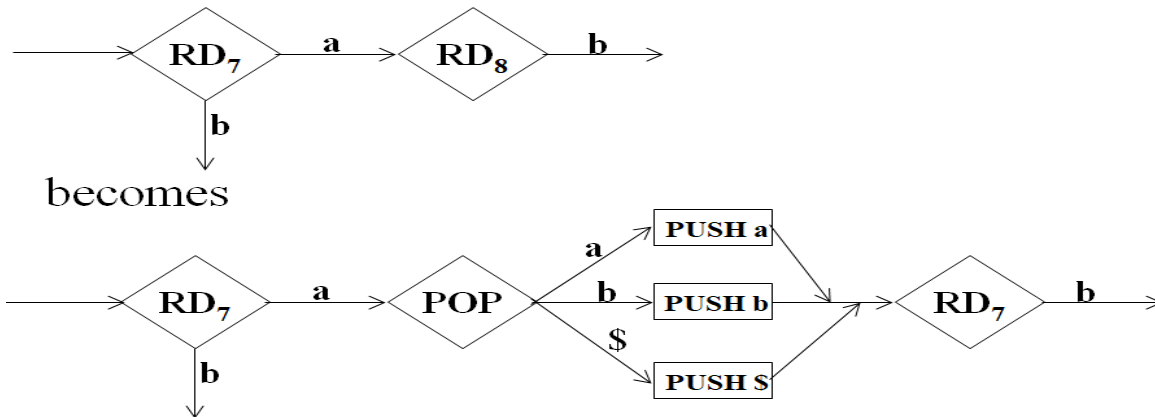
Every READ or HERE is followed immediately by a POP i.e. every edge leading out of any READ or HERE state goes directly into a POP state. No two POPs exist in a row on the same path without a READ or HERE between them whether or not there are any intervening PUSH states (i.e. the POP states must be separated by READs or HEREs).

All branching, deterministic or nondeterministic occurs at READ or HERE states, none at POP states and every edge has only one label.

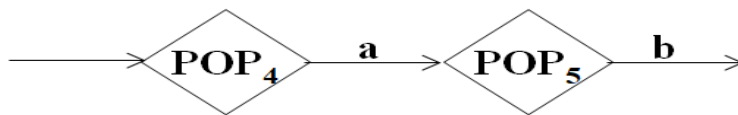
Even before we get to START, a “bottom of STACK” symbol \$ is placed on the STACK. If this symbol is ever popped in the processing it must be replaced immediately. The STACK is never popped beneath this symbol. Right before entering ACCEPTS this symbol is popped out and left. The PDA must begin with the sequence.



The entire input string must be read before the machine can accept the word. Different situations of a PDA to be converted into conversion form are discussed as follows



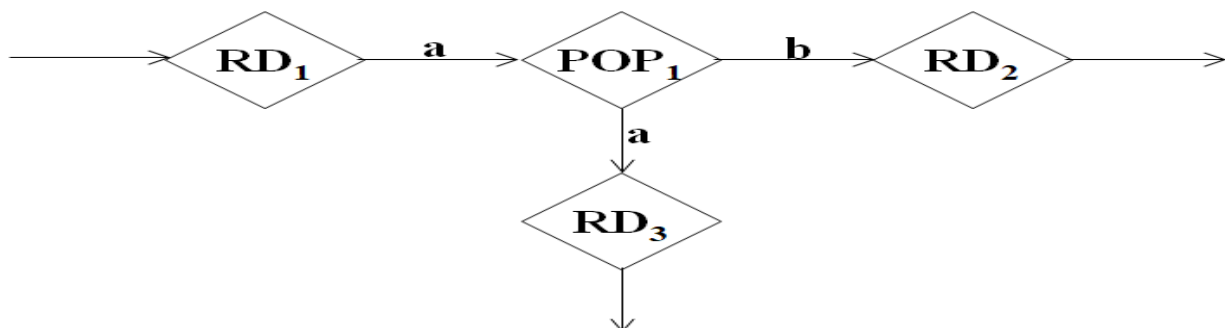
To satisfy condition 4,

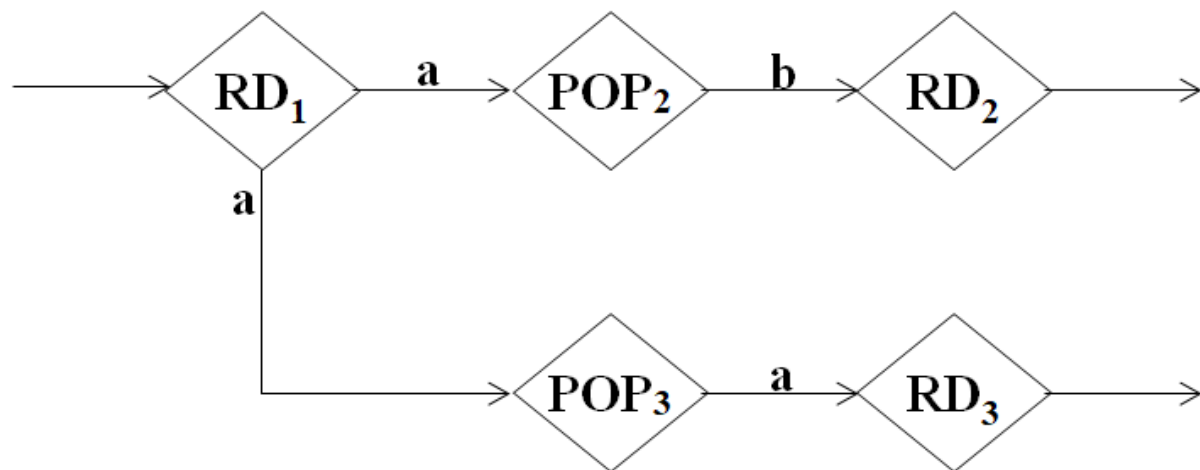


becomes

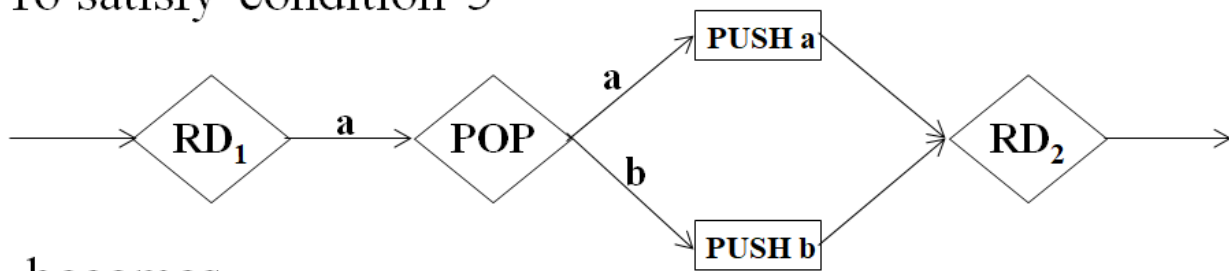


To satisfy condition 5

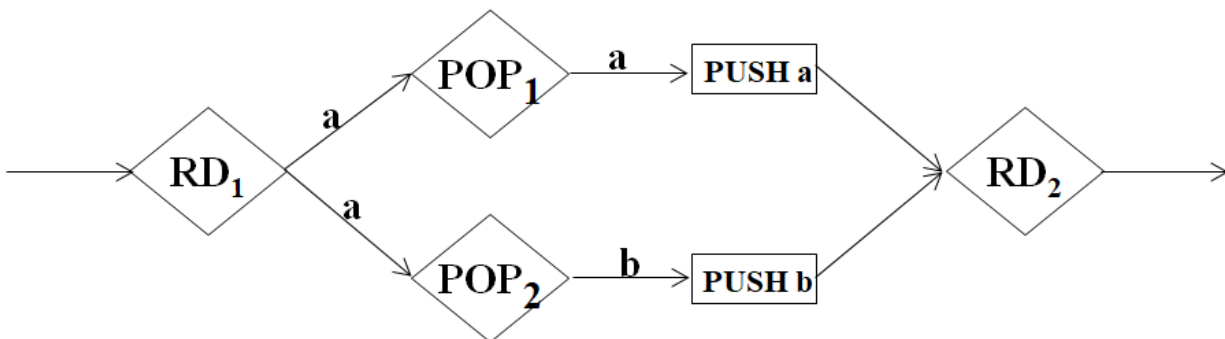




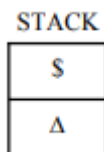
To satisfy condition 5



becomes



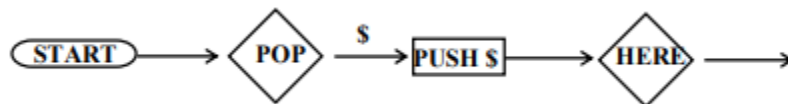
To satisfy condition 6, it is supposed that the STACK is initially in the position



shown below .

Conversion form of PDA

A PDA is in conversion form if it fulfills the following conditions: There is only one **ACCEPT** state. There are no **REJECT** states. Every **READ** or **HERE** is followed immediately by a **POP** i.e. every edge leading out of any **READ** or **HERE** state goes directly into a **POP** state. No two **POPs** exist in a row on the same path without a **READ** or **HERE** between them whether or not there are any intervening **PUSH** states (i.e. the **POP** states must be separated by **READs** or **HEREs**). All branching, deterministic or nondeterministic occurs at **READ** or **HERE** states, none at **POP** states and every edge has only one label. Even before we get to **START**, a “bottom of **STACK**” symbol **\$** is placed on the **STACK**. If this symbol is ever popped in the processing it must be replaced immediately. The **STACK** is never popped beneath this symbol. Right before entering **ACCEPT** this symbol is popped out and left. The PDA must begin with the sequence.



The entire input string must be read before the machine can accept the word.

STACK consistent.

It is very important to determine which sequences of rows do correspond to possible paths through the PDA, because the paths are directly related to the language accepted, e.g. Row4 cannot be immediately followed by Row6 because Row4 leaves in **HERE**, while Row6 begins in **Read2**. Some information must be kept about the **STACK** before rows are concatenated.

To represent a path, a sequence of rows must be joint-consistent (the rows meet up end to end) and **STACK** consistent (when a row pops a character it should be there at the top of the **STACK**). The next target is to define row language whose alphabet is $\Sigma = \{\text{Row1}, \text{Row2}, \dots, \text{Row7}\}$ i.e. the alphabet consists of the letters which are the names of the rows in the summary table.

Note

It may be noted that the words of the row language trace joint-to-joint and **STACK** consistent paths, which shows that all the words of this language begin with Row1 and end in Row7. Consider the following row sequence Row5 Row3 Row6. This is string of 4 letters, but not words of the row language because it does not represent a path starting from **START** and ending in **ACCEPT** state. It is not joint consistent. It is not **STACK** consistent. Before the CFG that generates the language accepted by the given PDA, is determined, the CFG that generates the row language is to be determined. For this purpose new nonterminal are to be introduced that contain the information needed to ensure joint and **STACK** consistency.

It is not needed to maintain any information about what characters are read from the **TAPE**.

Lecture N0. 42 ^{CS402}

Skip

Lecture N0. 43 ^{CS402}

Non-Context-Free language

Whether all languages are CFL?

Languages which are not Context-Free are called Non-CFL. To prove the claim that all languages are not Context-Free, the study of machines of word production from the grammar is needed

Live production: A production of the form nonterminal $\rightarrow$ string of two nonterminal is called a live production.

Dead production: A production of the form nonterminal $\rightarrow$ terminal is called a dead production. It may be noted that every CFG in CNF has only these types of productions.

Theorem

If a CFG is in CNF and if there is restriction to use the live production at most once each, then only the finite many words can be generated. It may be noted that every time a live production is applied during the derivation of a word it increases the number of nonterminals by one. Similarly applying dead production decreases the nonterminals by one. Which shows that to generate a word, one more dead production are applied than the live productions e.g.

$S \Rightarrow XY \Rightarrow aY \Rightarrow aa$

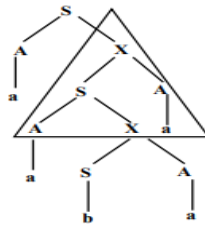
if a CFG in CNF has p live and q dead productions then all words generated without repeating any live production have at most $(p+1)$ letters.

Theorem

If a CFG is in CNF with p live and q dead productions and if w is word generated by the CFG, having more than $2p$ letters then any derivation tree for w has a nonterminal z which is used twice, where the second z is the descended from the first z . It can be observed from the above theorem that generation tree of word w has more than p rows.

Self-embedded nonterminal

A nonterminal is said to be self-embedded, if in a given derivation of a word, it ever occurs as a tree.



Here the nonterminal X is self-embedded.

Pumping lemma for CFLs Theorem

If G is any CFG in CNF with p live productions, then every word w of length more than $2p$ can be partitioned into five substrings as $w = uvxyz$, where x is not null string and v and y are not both null string. Then all the words of the form $uv^n xy^n z$, $n = 1, 2, 3, \dots$ can also be generated by G .

Example

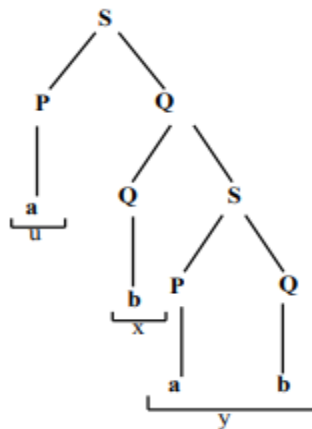
Consider the following CFG which is in CNF

$S \rightarrow PQ$

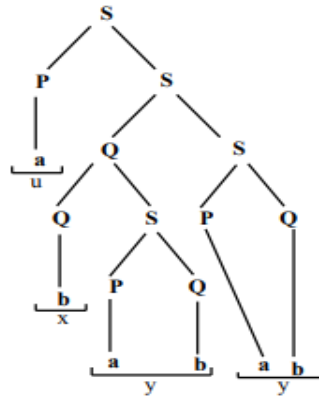
$Q \rightarrow QS \mid b$

$P \rightarrow a$

and a word $abab$ generated by the above CFG with the following derivation tree.



Then w can be broken up as $w = uvxyz$ where $u = a$, $v = \Lambda$, $x = b$, $y = ab$, $z = \Lambda$. Repeating the triangle from the second Q just as it descends from the first Q , the corresponding tree may be expressed as follows



Lecture NO. 44 ^{CS402}

Decidability

Following are the decidable problems w.r.t. CFG

Whether or not the given CFG generates any word?

Problem of emptiness of CFL.

Whether or not the given CFG generates the finite language?

Problem of finiteness.

Whether or not the given string w can be generated by the given CFG?

Problem of membership.

Following are algorithms showing that the answers to the above three questions are yes.

Algorithm 1 (Emptiness)

If the given CFG contains a production of the form $S \rightarrow \Lambda$, then obviously the corresponding CFL is not empty. If the CFG contains the production of the form $S \rightarrow t$, where t is a terminal or string of terminal then t is a word of the corresponding CFL and CFL is not empty.

Algorithm 2 (Uselessness)

Find all unproductive nonterminals (the nonterminal is unproductive if it cannot produce a string of terminals). Eliminate all productions involving unproductive nonterminals.

Algorithm 3 (Finiteness)

- I. Determine all useless nonterminals and eliminate all productions involving these nonterminals.
- II. For each of the remaining nonterminals, determine whether they are self-embedded (using the following steps). Stop if a self-embedded nonterminal is discovered.
- III. If Y is any nonterminal on the left side of the production with X in the right side, then paint Y blue.

Parsing Techniques

Parsing Techniques Recall the CFG for arithmetic expression

$S \rightarrow S+S | S*S | \text{number}$

Parsing of word

The process of finding the derivation of word generated by particular grammar is called parsing. There are different parsing techniques, containing the following three

- Top down parsing
- Bottom up parsing
- Parsing technique

For particular grammar of arithmetic expression

It can be noted that Bottom Up Parsing can be determined similar to that of Top Down Parsing with the change that in this case, the process is started with the given string and the tree is extended till S is obtained.

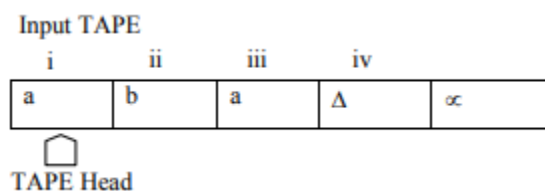
Lecture NO. 45 ^{CS402}

Turing machine

The mathematical models (FAs, TGs, PDAs) that have been discussed so far can decide whether a string is accepted or not by them.

Definition

A Turing machine (TM) consists of the following An alphabet Σ of input letters. An input TAPE partitioned into cells, having infinite many locations in one direction. The input string is placed on the TAPE starting its first letter on the cell i, the rest of the TAPE is initially filled with blanks (Δ 's).

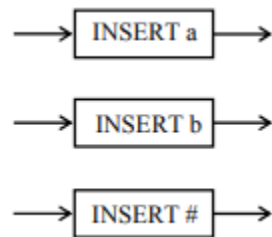


A tape Head can read the contents of cell on the TAPE in one step. It can replace the character at any cell and can reposition itself to the next cell to the right or to the left of that it has just read. An alphabet Γ of characters that can be printed on the TAPE by the TAPE Head. Γ may include the letters of Σ . Even the TAPE Head can print blank Δ , which means to erase some character from the TAPE. Finite set of states containing exactly one START state and some (may be none) HALT states that cause execution to

terminate when the HALT states are entered. A program which is the set of rules, which show that which state is to be entered when a letter is read from the TAPE and what character is to be printed. This program is shown by the states connected by directed edges labeled by triplet (letter, letter, direction).

It may be noted that there may not be any outgoing edge at certain state for certain letter to be read from the TAPE, which creates nondeterminism in Turing machines. It may also be noted that at certain state, there can't be more than one outgoing edges for certain letter to be read from the TAPE. The machine crashes if there is not path for a letter to be read from the TAPE and the corresponding string is supposed to be rejected. To terminate execution of certain input string successfully, a HALT state must be entered and the corresponding string is supposed to be accepted by the TM.

INSERT subprogram Sometimes, a character is required to be inserted on the TAPE exactly at the spot where the TAPE Head is pointing, so that the character occupies the required cell and the other characters on the TAPE are moved one cell right. The characters to the left of the pointed cell are also required to remain as such. The subprogram of insertion is independent and can be incorporated at any time with any TM program specifying what character to be inserted at what location. The subprogram of insertion can be expressed as



The above diagrams show that the characters a,b and # are to be inserted, respectively. Following is an example showing how the subprogram INSERT performs its function.