Theory of computation

Prof. R Shivamallikarjun, Assistant Professor
Information Technology

CHAPTER-3

Grammars

What is Grammar ?

- **A grammar consists of one or more variables that represent classes of strings.**

Grammar (G) consists of four tuple and it is represented as G= (V, T, P, S) where,

V= Finite set of variables.

T= Set of terminal.

P= Set of production rules. It is denoted as $\rightarrow$

S= Start symbol that represents the language being defined.

Context Free Grammars

- A context-free grammar (CFG) is a set of recursive rewriting rules used to generate patterns of strings.
- It is more powerful than regular grammar, but still can not define all possible language.
- Production rule of CFG can be represented as L → R. L is a single nonterminal symbol, and R is Terminal or Non terminal or Empty String.

- **Example**

The grammar G = ({ S } , { a , b } , P , S) with productions

$S \to aSa$

$S \to bSb$

$S \to \varepsilon$

Context Free Grammars

- **CFG for Palindrome**

G= ({P}, {0, 1}, A, P)

Where A represents the production rules:

$P \rightarrow \epsilon$

$P \rightarrow 0$

$P \rightarrow 1$

$P \rightarrow 0P0$

$P \rightarrow 1P1$

We can also write: $P \rightarrow \epsilon|0|1|0P0|1P1$

Context Free Language (CFL)

In formal language theory, a CFL is a language generated by a context-free grammar.
It is denoted as L (G), where G is a CFG.

A model of CFL $\{L=a^nb^n:n\geq1\}$ represents the language of all non-empty even-length strings.

Chomsky Normal Form (CNF)

- **A context-free grammar G is said to be in Chomsky normal form if all of its production rules are of the form.**

$A \rightarrow BC$ or $A \rightarrow a$ or $S \rightarrow \varepsilon$

Where A, B, and C are nonterminal symbols, a is a terminal symbol (a symbol that represents a constant value), S is the start symbol, and ε denotes the empty string.

Eliminating Null Production

Null productions are of the form **A -> ϵ.**

Step 1: Find out nullable non-terminal variables which derive ε.

Step 2: For each production A → a, construct all productions A → x where x is obtained from 'a' by removing one or multiple non-terminals from Step 1.

Step 3: Combine the original productions with the result of step 2 and remove ε - productions.

Example

Remove the null productions from the following grammar.

S -> ABAC

A -> aA / ϵ

B -> bB / ϵ

C -> c

Example

Variable A and B derive empty string.

At first, we will remove A → ε.

S -> ABAC / ABC / BAC / BC

A -> aA / a

B -> bB / ϵ

C -> c

B → b

Now we will remove B → ε.

S -> ABAC / ABC / BAC / BC / AAC / AC / C

A -> aA / a

B -> bB / b

C -> c

Eliminating Unit Production

- **A unit production is one of the form A → B, where both A and B are single non-terminals.**

S -> AB

A -> a

B -> C / b

C -> D

D -> E

E -> a

Example

In the above example there are three unit production:

B -> C

C -> D

D -> E

For production D -> E there is E -> a so we add D ->a to the grammar and add D -> E from the grammar. Now we have C -> D so we add a production C -> a to the grammar and delete C -> D from the grammar. Similarly we have B -> C by adding B ->a and removing B -> C.

Example

S -> AB
A -> a
B -> a / b
C -> a
D -> a
E -> a

We can see that C, D and E are unreachable symbols so to get a completely reduced grammar we remove them from the CFG. The final CFG is

S -> AB

A -> a

B -> a / b

Eliminating Useless Production

- **Those symbols that do not participate in derivation of any string, i.e. the useless symbols, and remove the useless productions from the grammar.**
- **All terminals will be useful symbols.**

- **Example:**

S -> AB/a

A -> BC/b

B -> aB/C

C -> aC/B

Example

Useful Symbols: {a, b, S, A}.
Useless Symbol: {B, C}.
Remove the production B and C.
Now we left with production,
S -> a

A -> b.

Now A is not reachable from start symbol S. so we will remove production of A.
So finally we left with production.
S -> a.

Converting CFG to CNF

- **Steps to convert CFG to CNF**

Step 1: If the start symbol S occurs on some right side, create a new start symbol S’ and a new production S’→ S.

Step 2: Eliminate Null Productions.

Step 3: Eliminate Unit Productions.

Step 4: Restricting the right side of productions to single terminal or string of two or more non-terminals.

Step 5: Shorten the string of length to 2 (If n>2).

Example of CFG to CNF Conversion

S□AAC

A□aAb|ϵ

C□aC|a

Step1: Eliminate Null production

S□AAC | AC |C

A□aAb|ab

C□aC|a

Step-2: Eliminate Unit Production

S□AAC | AC |aC |a

A□aAb|ab

C□aC|a

Example of CFG to CNF Conversion

Step 3:Replace non-terminals

S□AAC | AC |C

A□aAb|ab

C□aC|a

P□a

Q□b

Step 4:Shorten the non-terminal to length 2

S□AX_1

S□AC|PC|a

A□PY_1

A□PQ

C□PC|a

P□a

Q□b

X_1□AC

Y_1□AQ

Greibach Normal Forms (GNF)

- **In GNF if the right-hand sides of all production rules start with a terminal symbol, optionally followed by some variables.**
- **Example:**

A → b

A → bA1…An. A non-terminal generating a terminal which is followed by any number of non-terminals.

S → ε

Where,

S is Start symbol.

A, A1... An are non-terminals and b is a terminal.

Deterministic pushdown automata (DPDA)

- Context-free grammars generate context-free language and push down automata recognize all of the strings in the language.
- A deterministic context-free language is a language that can be recognized by a DPDA.
- The deterministic context-free languages are a proper subset of the context-free languages.
- Adding Memory to Automata. We can augment a finite automaton by adding in a memory device for the automaton to store extra information.

Deterministic pushdown automata (DPDA)

- **Pushdown automaton machine describe by the 7-tuple (Q, Σ, Γ, δ, q0, Z0, F)**

where,

Q is a finite set of states.

Σ is an alphabet.

Γ is the stack alphabet of symbols that can be pushed on the stack

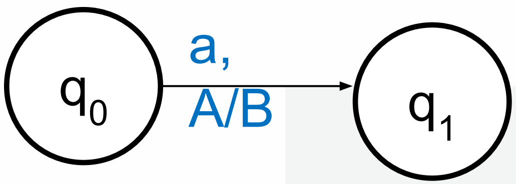
δ: is a transition function

q0 is the start state.

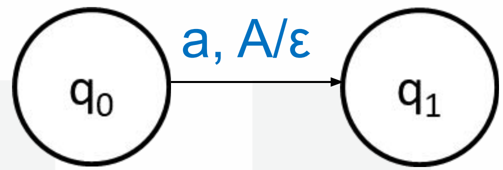
Z0 Bottom of the stack.

F Final states.

Deterministic pushdown automata (DPDA)



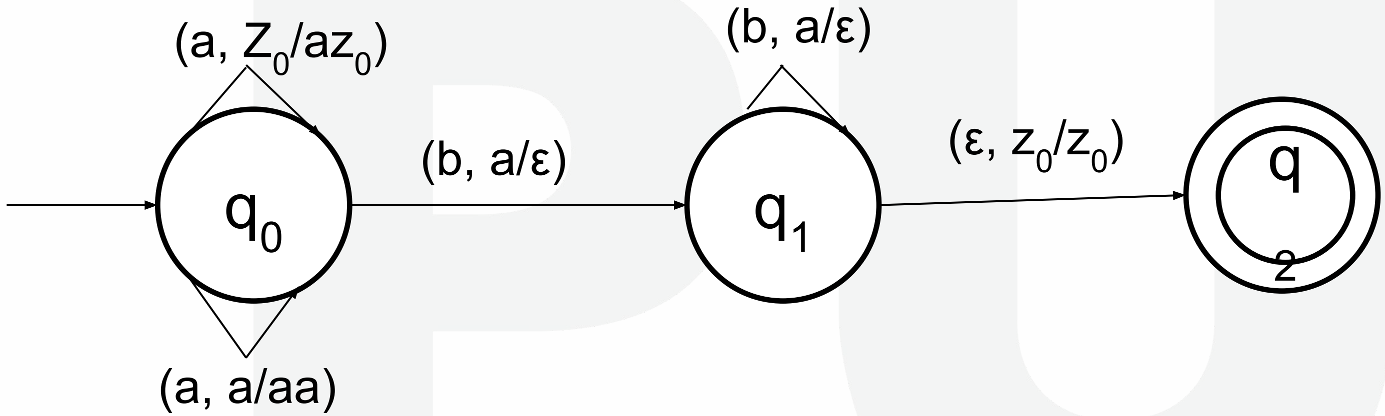
From the above transition diagram:
If next input symbol is a, stack top is A then read a from word, pop A, push B.



From the above transition diagram:
If next input symbol is a, stack top is A then read a from word, pop A.
Here is nothing to push.

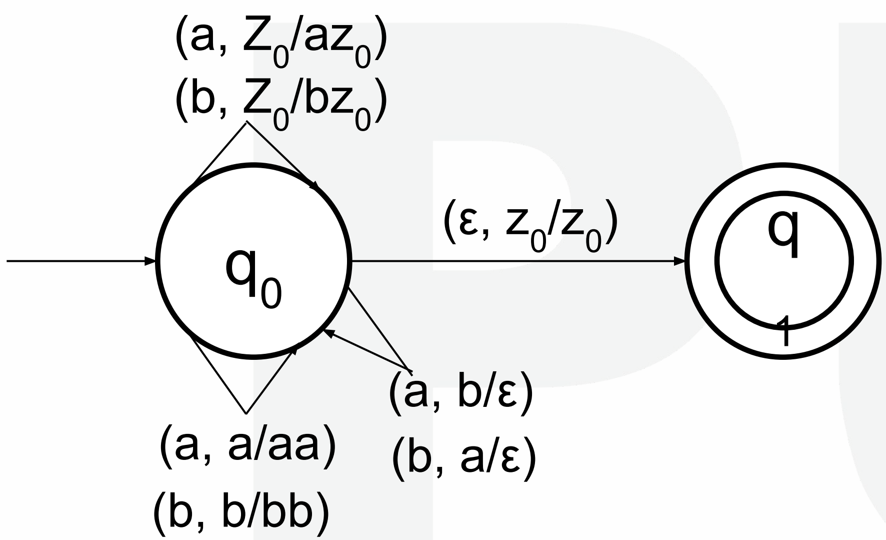
Deterministic pushdown automata (DPDA)

Example: $\{L=a^nb^n: n>=1\}$



Deterministic pushdown automata (DPDA)

Example: {na(w)=nb(w)}



Deterministic pushdown automata (DPDA)

Example: {L=$a^n b^n$: n>=1}

Transition Function:

δ(q0,a,z0)□δ(q0,az0)

δ(q0,a,a)□δ(q0,aa)

δ(q0,b,a)□δ(q1, ε)

δ(q1,b,a)□δ(q1, ε)

δ(q1, ε,z0)□δ(q2, z0)

Nondeterministic pushdown automata (NPDA)

- A nondeterministic pushdown automaton differs from a deterministic pushdown automaton (DPDA) in almost the same ways: The transition function delta is at most single-valued for a DPDA, multi-valued for an NPDA.
- NPDA is very similar to NFA.
- NPDA is more powerful then DPDA.
- Some languages are not accepted by DPDA but that is accepted by NPDA.

Equivalence with CFG

- While dealing with finite automata, there is no difference in the power of NFAs and DFAs. The power of NFA and DFA is equivalent.
- However, while dealing with PDA, there are CFLs that can be recognized by NPDAs that cannot be recognized by DPDA.
- If Language L is accepted by DPDA then L is also accepted by NPDA, but vice versa is not true.
- Power of NPDA is not equivalent to power of DPDA.

Equivalence with CFG

- **Examples of deterministic CFLs:**

1. $\{a^n b^n \mid n \geq 0\}$
2. $\{w\ c\ w^R \mid w \in \{a, b\}^*\}$

- **Examples of CFLs that are not deterministic:**

1. $\{ww^R \mid w \in \{a,b\}^*\}$
2. $\{w \in \{a,b\}^* \mid w=w^R\}$(Palindromes)

Note: L is deterministic CFL implies L is unambiguous, but there are unambiguous CFLs that are not deterministic

Parse trees

- **A parse tree is an entity which represents the structure of the derivation of a terminal string from some non-terminal.**

Derivation: The process of deriving a string is called as derivation. There are two types of derivation, first one is left most derivation and second is right most derivation.

Left most derivation (LMD)

- **The process of deriving a string by expanding the leftmost non-terminal at each step is called as leftmost derivation.**

Example: E → E + E | E * E | (E) | id
Derive Statement A * B + C using LMD.

- E → E * E
- E → A * E
- E → A * E + E
- E → A * B + E
- E → A * B + C

Right most derivation (RMD)

- **The process of deriving a string by expanding the rightmost non-terminal at each step is called as rightmost derivation.**

Example: E → E + E | E * E | (E) | id
Derive Statement A * B + C using RMD.

- E → E * E
- E → E * E+E
- E → E* E + C
- E → E * B + C
- E → A * B + C

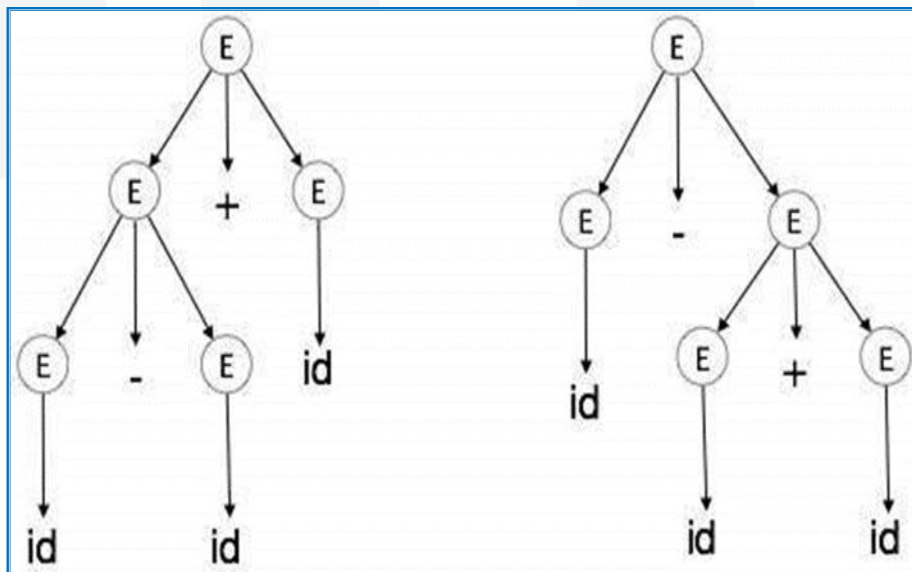
Ambiguity

- A Grammar is said to be ambiguous if it produces more than one parse tree for same sentence.
- In contrast, unambiguous grammars generate only one parse tree.
- For unambiguous grammars, Left most derivation and Rightmost derivation represents the same parse tree.
- For ambiguous grammars, Left most derivation and Rightmost derivation represents different parse trees.

Ambiguity

Example: The below grammar is ambiguous because there are two distinct parse trees for the same statement id - id + id.

- Suppose we want to evaluate expression 5-2+3.
- After evaluation of first parse tree we get (5-2) +3=3+3=6
- After evaluation of second parse



Pumping lemma

For every CFL there exists a constant n such that if z is a string in L of length at least n, then we can write z=uvwxy such that–

- $|vwx| \leq n$
- $vx \neq \varepsilon$
- For all $i \geq 0$ the string $uv^iwx^iy \in L$.

Proof that the given string $0^n10^n10^n$ is not a CFL by using pumping lemma.

Consider $z = 0^n10^n10^n$.

Pumping lemma

We can write z = uvwxy, where $|vwx| < n$, and $|vx| > 1$.

Case 1: vx has no 0's.Then at least one of them is a 1, and uwy has at most one 1, which no string in L does.

Case 2: vx has at least one 0. vwx is too short (length $<n$) to extend to all three blocks of 0's in 0n10n10n. Thus, uwy has at least one block of n 0's, and at least one block with fewer than n 0's. Thus, uwy is not in L.

Closure properties of CFLs

- **The class of CFLs is closed under the following operations:**

1. The union L ∪ P of L and P
2. The reversal of L
3. The concatenation L of L and P
4. The Kleene star L* of L.

Non closure under intersection, complement, and difference.

DCFL does not closed under union but closed under complement.

Context-sensitive languages (CSL)

- A context-sensitive grammar (CSG) is a formal grammar in which the left-hand sides and right-hand sides of any production rules may be surrounded by a context of terminal and nonterminal symbols.
- Language that can be defined by CSG is called CSL.
- It is less restricted then context-free grammar.
- It is called Type 1 grammar according to Chomsky hierarchy.

Context-sensitive languages (CSL)

All rules in P are in the form of $\alpha1\ A\ \alpha2 \rightarrow \alpha1\ \beta\ \alpha2$

Suppose that left side of production is denoted as u and right side of production is denoted as v. then must ensure that length of u is always less than equal to length of v.

$$|u|<=|v|.$$

Example:

AB → AbBc

A → bcA

B → b

Closure properties of CSL

- Union
- Intersection
- Concatenation
- Complement
- Kleene closure
- Set difference
- Reversal

Linear bounded automata and equivalence

- A linear bounded automaton (LBA) is a multi-track non-deterministic Turing machine with a tape of some bounded finite length.
- LBA is a Turing machine that cannot use extra working space.
- It can only use the space taken up by the input string tape space.
- The acceptance problem for linear bounded automata is decidable.
- The membership problems for sets accepted by linear bounded automata are solvable.

Linear bounded automata and equivalence

- The emptiness problem is unsolvable for linear bounded.
- Example: Language accepted by LBA $\{L=a^n b^n c^n : n>=0\}$
- Above language is neither accepted by finite automata nor by push down automata.
- LBA have more power than NPDA but less power than Turing machine.
- Also Non deterministic LBA have same power as deterministic LBA.

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.i