Revision

Aditya Yadav

February 1, 2023

Contents

1 Grammer Type (Noam Chomsky Classification)

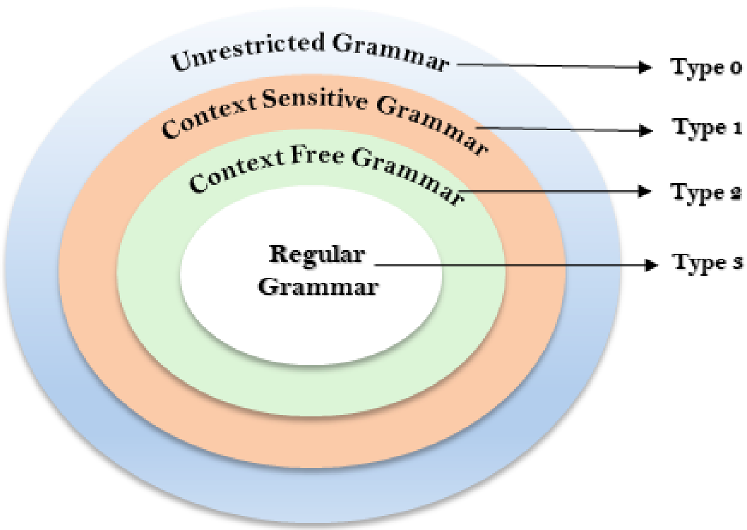


Fig: Chomsky Hierarchy

1.1 Type 0 (unrestricted Grammer)

1. α -> β
2. $\alpha \in (V+T)^+$
3. $\beta \in (V+T)^+$

Example -> AB -> ϵ Not a Example -> ϵ -> A

1.2 Type 1 (Context Sensitive Grammer) (CSL)

1. Must be a Type-0
2. $|\alpha| \leq |\beta|$

Not Example -> AB -> α

1.3 Type 2 (Context Free Grammer) (CSL)

1. Must be Type-1
2. V -> $(V+T)^+$

1.4 Type 3 (Regular Grammer)(RG)

1. Must be Type-2
2. V => T^* × V / T^* -> Right linear Grammer OR V => V × T^* / T^* -> Left linear Grammer

2 Ambigious grammer

2.1 Defination

Ambigious grammer -> A grammer which can derive multiple parse trees for an input string it is called a ambigious grammer.

2.2 Conversion from ambigious to unambigious grammer

- Converting a ambigious grammer to unambigious grammer is not always possible.
- There is no algorithm for converting unambigious grammer to ambigious grammer.
- Ambigious grammers from which ambigity can't be remvoed are known as inherently ambigious grammer.
- A language is said to be ambigious if all the grammers that generate that language are ambigious.

2.2.1 By Asssigning priority

S -> id/S+S/SxS -> Removal Ambiguity Priority S -> S+T/T T -> TxF/F F -> id The bottom productions has the highest priority and the higher we go the less priority that productions get.

2.2.2 Left factoring

in the below case the first grammer will cause the compiler to be confused as if a string starting with 'a' comes it will not be sure which of the productions to use. Ex-> W = ad S -> ab/ac/ad to S -> aS' S' -> b/c/d

1. Example S -> a/b/c/iEtS/iEtSeS to S -> a/b/c/iEtsP/iEtsSP P -> eS/ϵ

2.2.3 Recursion

1. Types recursion If the grammer contains left recursion then top down parser will go to infinite loop.

Left Recursion	Right Recursion
S -> Sa/b	S -> aS/b

2. Conversion A -> Aa/B ——- It has a recusion part Aa and a non recursion part and we try to seperate them to A -> BA' A' -> aA'/ϵ

3. Example E -> E+T/T T -> T*F/F F -> id/(E) to E -> TE' E' -> +TE'ϵ *T -> FT' T' -> *FT'*ϵ F -> id/(E)

3 Compiler

1. Processor can directly take high level language program also but it will take a lot of time, hence to make it faster we use a compiler as the processor will compile or covert the program each time it runs the program but the compiler only converts it only once.

2. Compilation in java in compiler is faster than c-compiler as it does coversion up to the intermediate code generation stage only hence each time the program runs these last two stages are executed making to slower than c execution.

Language Processing	–HLL–>	Pre Processor	—Pure HHL—>	Compiler	——Multi Assembly

3.1 Lexical Analyzer

1. Lexical analyzer will generate a token only when parser asks for it.

2. First phase of the compiler is called lexical analyzer. It is also called scanner.It will divide the given program into meaningful strings know as token.

3.1.1 Functions

1. Dividing the program into tokens

2. It will eliminate the comment lines

3. It will eliminate the whitespace chracters(tab,\, ,"").

4. It will help in giving error message by providing the line number.

3.2 Parser

The process of deriving string from a given grammer is called derivation or parsing

3.2.1 Types of parser

1. Top down Parser These are also called LL(Left to Right,Left most derivation) parsers. It start with root or starting symbol and proceeds to children that is String.

    (a) Top down parsers use leftmost derivation.

    (b) All the parsers perform left most derivation only and none of them gives right of most derivation.

    (c) Difficulty with top down parsers is that when a variable has more than 1 choice it has to choose the correct production by backtracing.

    (a) Recursive descent Parser (LL(0))

        i. In Recursion descent Parser we use leftmost Derivation.

        ii. In Recursion descent Parser we will write one function for every variable

        iii. If the grammer contains left Recursion the parser will go into infinite loop.

        iv. If the grammer contains sometimes we will get parsing error.

        v. Lot of time is wasted in back tracing so time complexity of Recursion descent Parser is $O(2^n)$.

```
S(){
    choose any production S -> x1,x2,x3,x4.....xk of S
    for(i=1 to k){
       if (xi is variable) { x1() ;}
       else if (xi == look ahead symbol) { then increment input pointer;}
       else { Error {back track} }
    }
}
```

   (b) Predective Parser (LL(1))

2. Bottom Up parser

   (a) It start from the children or the string and proceeds to root or the starting symbol.
   (b) It uses reverse of right most derivation.
   (c) The Different with bottom up parser is identifying a substring or handle which will give a required variable to get to the start symbol.

4 Question

4.1 Q1

Write a CFG for Language L = { a^m | m $\geq$ 1 } Ans. S -> aS | a

4.2 Q2

Write a CFG for Language L = { a^m b^n | m,n $\geq$ 1} Ans. S-> AB A-> Aa | a B-> Bb | b

4.3 Q3

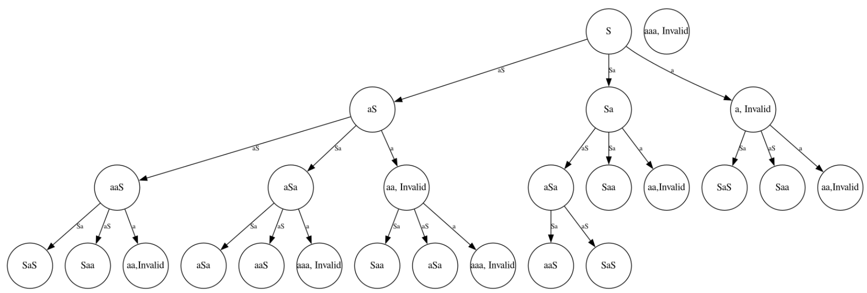
How many possible DFA's are there with 2 states X and Y, where X is an initial state over alphabet {a,b}?

N/NF	X	Y
X	a/b	a/b
Y	a/b	a/b

Here we can set either X or Y or neither or both as final states so we have to multiple the table result with 4 ans = 4 * 4 * 4 = 64

4.4 Q4

TO Complete/ Draw the parse tree of this statement? S -> aS/Sa/a W -> aaa


S
aaa, Invalid
aS
Sa
a, Invalid
aaS
aSa
aa, Invalid
aSa
Saa
aa,Invalid
SaS
Saa
aa,Invalid
SaS
Saa
aa,Invalid
aSa
aaS
aaa, Invalid
Saa
aSa
aaa, Invalid
aaS
SaS