BITS Pilani
Hyderabad Campus

Theory of Computation (CS F351)

Prof.R.Gururaj
CS&IS Dept.

Context Free Grammars And Context Free Languages

(Chapter-3)

Concepts

Language Recognizer: A device that accepts valid strings.
Finite Automata are type of language Recognizers.

Language Generators: Devices that produce valid strings.
Ex: Regular Expressions.

Now we study certain types of formal language generators.

Language Generators

- That device begins when a signal to start is given to construct the string.
- It operation determined by a set of rules.
- Eventually this process halts and produces the completed string.
- The language defined by the device is set of all strings that it can produce.
- It is difficult to produce a recognizer for English language.

- We are interested in generators of artificial languages such as Regular Languages and CFL.
- Regular Expressions can be viewed as Language generators.

$$a(a^* \cup b^*)b$$

How to generate a string according to the above RE

- First output an a. Then do the following two.
- Either output a number of as or output number of bs.
- Finally output a b

- The language associated with this language generator is set of all strings that can be produced by the process above.

- Now we study more complex language generators called as *Context Free Grammars* (CFGs).

- CFGs are based on more complete understanding of the structure of the strings belonging to the language.

innovate achieve lead

Ex: $a \underbrace{(a^* \cup b^*)}_{M} b$

Start ↗ ↖ end

If S be the String in the Language and M be the Symbol for the middle part

$$S \rightarrow aMb$$

Read as 'can be'. We call such expression a <u>Rule</u>

What can 'M' be?

Obviously it is either String of a's or String of b's

We express this by adding a rule $M \rightarrow A$
$M \rightarrow B$

A } are new symbols that stand for
B } strings of a's or b's respectively.

What is a string of a's, it can be 'e' also.

$A \rightarrow e$

$A \rightarrow aA$

Similarly $B \rightarrow e$

$B \rightarrow bB$

Then the language denoted by RE $a(a^* \cup b^*)b$ can be defined alternatively by the following language generator —

$S \rightarrow aMb$

$M \rightarrow A$

$M \rightarrow B$

$A \rightarrow e$

$B \rightarrow e$

$A \rightarrow aA$

$B \rightarrow bB$

1. Start with the string containing the symbol S.
2. find a symbol in the current string that appear to the left of '$\longrightarrow$' in one of the rules above.
3. Replace an occurence of this symbol with the string that appears to the right of '$\longrightarrow$' in the same rule.

Repeat this process until no such symbol can be found.

CFG

- In CFG symbols that do not appear on the LHS of a production rule are known as terminal symbols.
- In the process of producing a string, by using CFG, we see only terminal symbols in the string. Then we stop further replacements and the result is a valid string according to the language.

CFG Definition

- A CFG $G = (V, \sum, R, S)$

V is an alphabet

$\sum$ is set of terminal symbols and subset of V

R set of rules $(V- \sum) X V^*$

S is the start symbol

and is an element of $(V- \sum)$

CFG Definition

CFG for L = $\{a^n\ b^n : n \geq 0\}$ // it is not a Regular Language

CFG G = (V, ∑ , R, S)

V = {S, a, b}

∑ =$\{a,b\}$

R =$\{ S \rightarrow aSb;\ S \rightarrow e\}$

S =is the start symbol

CFG Definition

- A derivation in G of w_n from w_0 may be any string in V^*, and n is the length of the derivation , may be any natural number including zero.

- We say that the derivation has n steps.

CFG Definition

CFG for L = {$a^n b^n$: $n \geq 0$}   CFG G = (V, ∑ , R, S)

V = {S, a, b}   ∑ ={a,b}

R ={ S→ aSb; S→e}   S =is the start symbol

One possible derivation:

S→ aSb → aaSbb → aaaSbbb → aaabbb

w_0 w_1 w_2 w_3 w_4

Here the length of derivation is 4

CFG and PL

- Computer programs written in any language must satisfy some rigid criteria in order to be syntactically correct, and therefore amenable for mechanical interpretation.
- The syntax of most of the languages can be captured by CFG
- If a programming language is described by CFG, it will be easy for parsing.
- Parsing is the process of analyzing a program to find the syntax.

CFG and PL

- The CFG theory will neatly complement the study of automata, which recognize languages.
- It has a practical value in the specification and analysis of computer languages.
- CFGs are based on more complete understanding of the structure of strings belonging to languages.
- Why it is context free?
- L(G) is the language generated by G.

CFG and PL

- A language L is said to be CFG if L=L(G) for some G.
- All regular Languages are Context Free

RL and CFL

We show how RLs are CFL by direct construction.

If a Finite Automaton M=(*K*, ∑ , δ, s, *F*)

G(M) = (V, ∑ , R, S)

V= (K U ∑)

S= s

R= { q→ aP: δ(q, a)=P } U {q→e : q ϵ F }

Construct a CFG for all integers with sign.

Ex: -17; +23;   etc.

And show how -17 is derived.

CFG

$S \rightarrow$ <sign> <integer>

<sign> $\rightarrow$ – | +

< integer> $\rightarrow$ <digit> <integer> | <digit>

<digit> $\rightarrow$ 0 | 1 |2 |……|9

Ex: how to derive “ – 17 “

$S \rightarrow$ <sign> <integer> $\rightarrow$ – <integer> $\rightarrow$ – <digit><integer> $\rightarrow$
– 1 <integer> $\rightarrow$ – 1 <digit> $\rightarrow$ – 17

Derivation and corresponding Tree representation

- The same string may result from several derivations.
- A parse tree is a picture of derivation of a string from the G.
- The *yield of the parse tree*
- It is possible that the different derivations may result in same parse tree(identical).
- We have LMD and RMD
- A CFG is ambiguous if it produces two or more distinct parse trees for a string (2 or more LMDs).

Sentence: The language generated by G (denoted by L(G)) any w that belongs to L (G) is a sentence.

Sentential Form: any intermediate String in the process of derivation.

(ii) $S \rightarrow aSb \mid aaSb \mid e$ (here, Σ={a,b} and the symbol 'e' stands for null)

Give LMD and RMD for 'aaabb
Do we have two LMDs'

Precedence of Derivations

If two derivations d_1 and d_2 are identical except for two consecutive steps, during which the same Non-terminals are replaced by same two strings but in opposite order in two derivations d_1 and d_2 , then the derivation in which the Left-most of the two NTs is replaced is said to precede the other.

CFG Simplification, Classification and Normal Forms

(

Simplification

We try to Eliminate:

1. NTs not yielding any terminal string
2. NTs not appearing in any sentential form
3. Null productions
4. Unit productions

Source : Theory of Computer Sc Automata Languages and Computation, 3rd Ed., KLP Mishra and N Chandra Sekhar, PHI

1. Eliminating NTs not yielding any terminal string

Theorem 6.3 If G is a CFG such that $L(G) \neq \emptyset$, we can find an equivalent grammar G' such that each variable in G' derives some terminal string.

Proof *Let* $G = (V_N, \Sigma, P, S)$. We define $G' = (V'_N, \Sigma, G', S)$ as follows:

(a) *Construction of* V'_N:

We define $W_i \subseteq V_N$ by recursion:

$W_1 = \{A \in V_N \mid$ there exists a production $A \to w$ where $w \in \Sigma^*\}$. (If $W_1 = \emptyset$, some variable will remain after the application of any production, and so $L(G) = \emptyset$.)

$$W_{i+1} = W_i \cup \{A \in V_N \mid \text{there exists some production } A \to \alpha \text{ with } \alpha \in (\Sigma \cup W_i)^*\}$$

By the definition of W_i, $W_i \subseteq W_{i+1}$ for all i. As V_N has only a finite number of variables, $W_k = W_{k+1}$ for some $k \leq |V_N|$. Therefore, $W_k = W_{k+j}$ for $j \geq 1$. We define $V'_N = W_k$.

(b) *Construction of* P':

$$P' = \{A \to \alpha \mid A, \alpha \in (V'_N \cup \Sigma)^*\}$$

We can define $G' = (V'_N, \Sigma, P', S)$. S is in V_N. (We are going to prove that every variable in V_N derives some terminal string. So if $S \notin V_N$, $L(G) = \emptyset$. But $L(G) \neq \emptyset$.)

2. Eliminating symbols not appearing in any sentential form

Theorem 6.4 For every CFG $G = (V_N, \Sigma, P, S)$, we can construct an equivalent grammar $G' = (V_N, \Sigma', P', S)$ such that every symbol in $V_N \cup \Sigma'$ appears in some sentential form (i.e. for every X in $V'_N \cup \Sigma'$ there exists α such that $S \overset{*}{\underset{G'}{\Rightarrow}} \alpha$ and X is a symbol in the string α).

Proof We construct $G' = (V'_N, \Sigma', P', S)$ as follows:

(a) *Construction of W_i for $i \geq 1$:*

(i) $W_1 = \{S\}$.

(ii) $W_{i+1} = W_i \cup \{X \in V_N \cup \Sigma \mid \text{there exists a production } A \rightarrow \alpha \text{ with } A \in W_i \text{ and } \alpha \text{ containing the symbol } X\}$.

We may note that $W_i \subseteq V_N \cup \Sigma$ and $W_i \subseteq W_{i+1}$. As we have only a finite number of elements in $V_N \cup \Sigma$, $W_k = W_{k+1}$ for some k. This means that $W_k = W_{k+j}$ for all $j \geq 0$.

(b) *Construction of V'_N, Σ' and P':*

We define

$$V'_N = V_N \cap W_k, \qquad \Sigma' = \Sigma \cup W_k$$

$$P' = \{A \rightarrow \alpha \mid A \in W_k\}.$$

3. Eliminating null productions

Theorem 6.6 If $G = (V_N, \Sigma, P, S)$ is a context-free grammar, then we can find a context-free grammar G_1 having no null prodctions such that $L(G_1) = L(G) - \{\Lambda\}$.

Proof We construct $G_1 = (V_N, \Sigma, P', S)$ as follows:

Step 1 *Construction of the set of nullable variables:*

We find the nullable variables recursively:

(i) $W_1 = \{A \in V_N \mid A \to \Lambda \text{ is in } P\}$
(ii) $W_{i+1} = W_i \cup \{A \in V_N \mid \text{there exists a production } A \to \alpha \text{ with } \alpha \in W_i^*\}$.

By definition of W_i, $W_i \subseteq W_{i+1}$ for all i. As V_N is finite, $W_{k+1} = W_k$ for some $k \leq |V_N|$. So, $W_{k+j} = W_k$ for all j. Let $W = W_k$. W is the set of all nullable variables.

Step 2 (i) *Construction of P':*

Any production whose R.H.S. does not have any nullable variable is included in P'.

(ii) If $A \to X_1X_2 \ldots X_k$ is in P, the productions of the form $A \to \alpha_1\alpha_2 \ldots \alpha_k$ are included in P', where $\alpha_j = X_i$ if $X_i \notin W$. $\alpha_i = X_i$ or Λ if $X_i \in W$ and $\alpha_1\alpha_2 \ldots \alpha_k \neq \Lambda$. Actually, (ii) gives several productions in P'. The productions are obtained either by not erasing any nullable variable on the R.H.S. of $A \to X_1X_2 \ldots X_k$ or by erasing some or all nullable variables provided some symbol appears on the R.H.S. after erasing.

Let $G_1 = (V_N, \Sigma, P', S)$. G_1 has no null productions.

Before proving that G_1 is the required grammar, we apply the construction to an example.

4. Eliminating unit productions

Definition 6.10 A unit production (or a chain rule) in a context-free grammar G is a production of the form $A \to B$, where A and B are variables in G.

Theorem 6.7 If G is a context-free grammar,we can find a context-free grammar G_1 which has no null productions or unit productions such that $L(G_1) = L(G)$.

Proof We can apply Corollary 2 of Theorem 6.6 to grammar G to get a grammar $G' = (V_N, \Sigma, P, S)$ without null productions such that $L(G') = L(G)$. Let A be any variable in V_N.

Step 1 *Construction of the set of variables derivable from A:*

Define $W_i(A)$ recursively as follows:

$$W_0(A) = \{A\}$$

$$W_{i+1}(A) = W_i(A) \cup \{B \in V_N \mid C \to B \text{ is in } P \text{ with } C \in W_i(A)\}$$

By definition of $W_i(A)$, $W_i(A) \subseteq W_{i+1}(A)$. As V_N is finite, $W_{k+1}(A) = W_k(A)$ for some $k \le |V_N|$. So, $W_{k+j}(A) = W_i(A)$ for all $j \ge 0$. Let $W(A) = W_k(A)$. Then $W(A)$ is the set of all variables derivable from A.

Step 2 *Construction of A-productions* in G_1:

The A-productions in G_1 are either (i) the nonunit production in G' or (ii) $A \to \alpha$ whenever $B \to \alpha$ is in G with $B \in W(A)$ and $\alpha \notin V_N$.

(Actually, (ii) covers (i) as $A \in W(A)$). Now, we define $G_1 = (V_N, \Sigma, P_1, S)$, where P_1 is constructed using step 2 for every $A \in V_N$.

Before proving that G_1 is the required grammar, we apply the construction to an example.

Classification of Grammars

According to Noam Chomosky, there are four types of grammars – Type 0, Type 1, Type 2, and Type 3. The following table shows how they differ from each other –

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type 0	Unrestricted grammar	Recursively enumerable language	Turing Machine
Type 1	Context-sensitive grammar	Context-sensitive language	Linear-bounded automaton
Type 2	Context-free grammar	Context-free language	Pushdown automaton
Type 3	Regular grammar	Regular language	Finite state automaton

Chomsky Normal Form (CNF)

The CNF has restriction on length of LHS and nature of symbols in LHS. A CFG G is said to be in CNF

NT→ (singleTerminal) | (NT)(NT))

(1) If every production is of the form A→ a OR

(2) A→BC and

(3) S→ e if e is in L(G)

When *e* is in L(G) we assume that S does not appear on the RHS of any production.

Ex:

(1) S→ AB; S→e ; A→a; B→b is in CNF

(2) S→ABC; S→aC, A→a, B→b, C→c is not in CNF

Some properties of CFL

CFLs are closed under union, concatenation, and Kleene star.

The intersection of a CFL and RL is a CFL.

CFLs are not closed under intersection and Complementation.

Other Properties of CFL

The Context-Free languages are closed under *union*, *concatenation*, and *Kleene* star.

The intersection of a Context-Free language with Regular Language is a Context Free Language.

CFLs are not closed under *intersection* or *complementation*.

There is a polynomial algorithm which, given a CFG, construct an equivalent PDA, and vice versa.

Let *G= (V, ∑, R, S)* be a Context-free Language.

The *fanout* of *G*, denoted by *Ø(G)*, is the largest number of symbols on the right-hand side of any rule in R.

A *path* in a parse-tree is a sequence of distinct nodes, each connected to the previous node by a line segment. The first node is the Root of the tree, and last node is the leaf.

The *length of the path* is the number of line segments in it.

The *height of the parse tree* is the length of the longest path in it.

The yield of any parse tree of G of height h has length at most $\emptyset(G)^h$

Pumping theorem for CFL

Let $G = (V, \Sigma, R, S)$ be a Context-free Grammar.

Then $w \in L(G)$ of length greater than $\varnothing(G)^{|V-\Sigma|}$ can be written as $w=uvxyz$ in such a way that v and y are non empty and uv^nxy^nz is in $L(G)$ for every $n \geq 0$

Then it is CFL otherwise not.

EX: $a^n b^n c^n$

Is not CFL

Conclusion to CFG/CFL

1. CFL
2. CFG
3. Grammar simplification
4. CNF and GNF
5. Regular Grammars
6. Properties
7. Pumping lemma for CFG