Part II Chapter 4

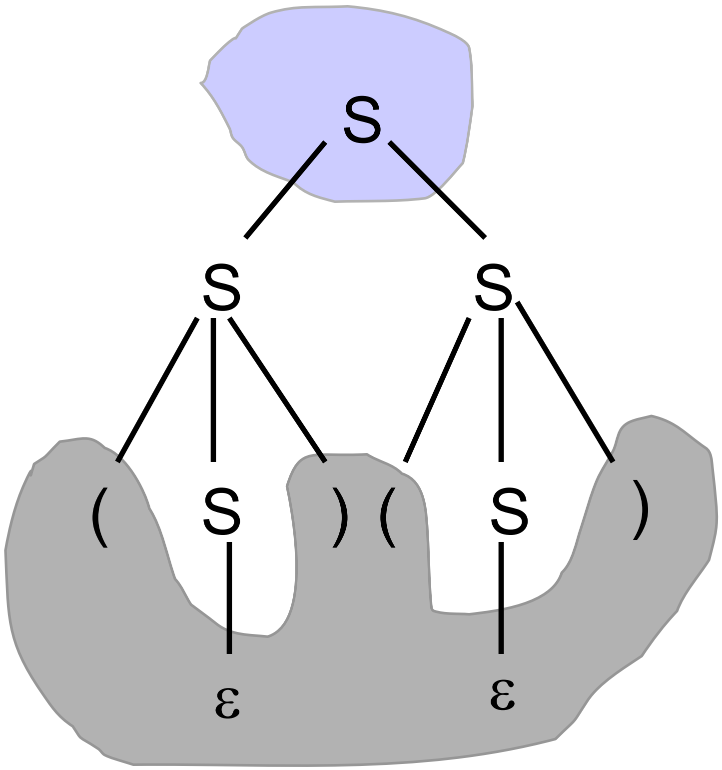
Parse Trees and Parsing

Derivations and Parse trees

- **G: S --> ε | SS | (S)   // L(G) = PAREN**
- **Now consider the derivations of the string: "()()".**
  - **D_1: S--> SS -->(S) S --> ()S --> ()(S) --> ()()**
  - **D_2: S-->SS -->S(S) -->(S)(S)-->(S)()-->()()**
  - **D_3: S-->SS -->S(S) --> S() -->(S)() --> ()()**
- Notes:
  - 1. D_1 is a leftmost derivation, D_3 is a rightmost derivation while D_2 is neither leftmost nor rightmost derivation.
  - 2. D_1 ~ D_3 are the same in the sense that:
    - ✪The rules used (and the number of times of their applications) are the same.
    - ✪All applications of each rule in all 3 derivations are applied to the same place in the string.
    - ✪More intuitively, they are equivalent in the sense that by reordering the applications of applied rules, they can be transformed to the same derivation.

Parse Trees

- D_1 ~ D_3 represent different ways of generating the following parse tree for the string "()()".



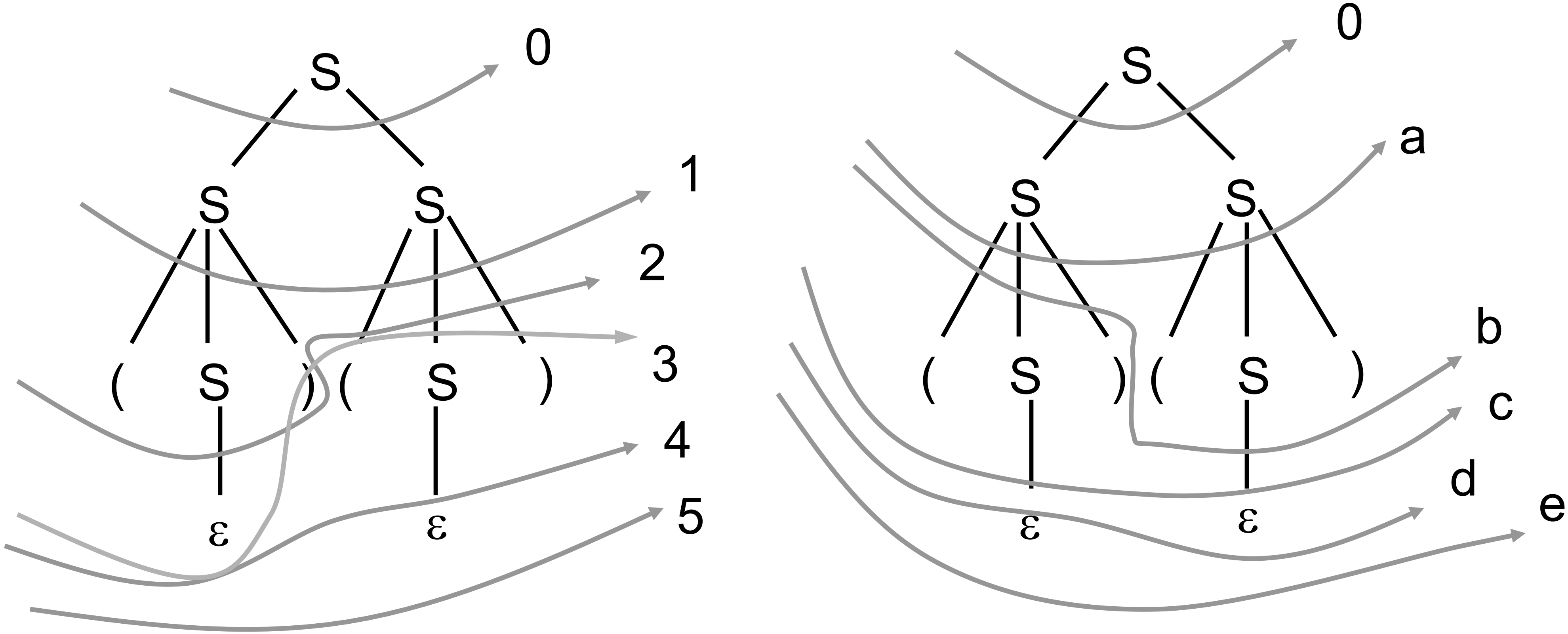
Features of the parse tree:

1. The root node is [labeled by] the start symbol: S
2. The left to right traversal of all leaves corresponds to the input string : ( ) ( ).
3. If X is an internal node and Y_1 Y_2 … Y_K are an left-to-right listing of all its children in the tree, then X --> Y_1Y_2… Y_k is a rule of G.
4. Every step of derivation corresponds to one-level growth of an internal node

A parse tree for the string "( ) ( )".

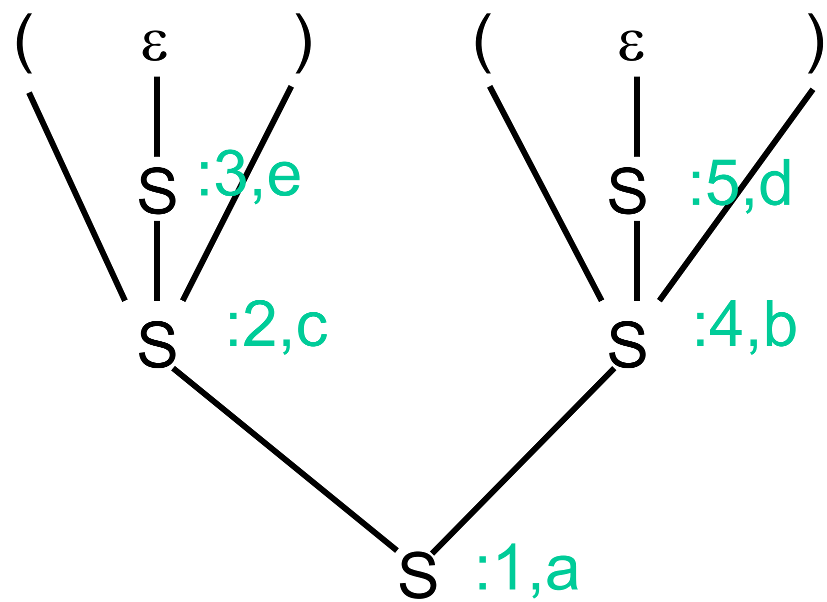
Mapping derivations to parse tree

- **How was a parse tree generated from a derivation ?**



Top-down view of D_1: S -->* ()() and D_2: S -->* ()().

Bottom-up view of the generation of the parse tree



Remarks:

1. Every derivation describes completely how a parse tree grows up.
2. In practical applications (e.g., compiler), we need to know not only if an input string w $\in$L(G), but also the parse tree (corresponding to S $\rightarrow^*$ w)
3. A grammar is said to be ***ambiguous*** if there exists some string which has more than one parse tree.
4. In the above example, '()()' has at least three derivations which correspond to the same parse tree and hence does not show that G is ambiguous.
5. Non-uniqueness of derivations is a necessary ***but not sufficient*** condition for the ambiguity of a grammar.
6. A CFL is said to be ambiguous if every CFG generating it is ambiguous.

An ambiguous context free language

- *Let L* = $\{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$
- It can be proved that the above language is inherently ambiguous. Namely, all context free grammars for it are ambiguous.

Parse trees and partial parse trees for a CFG

- G= (N,Σ,P,S) : a CFG

PT(G) $=_{def}$ the set of all parse trees of G, is the set of all trees corresponding to complete derivations (I.e., A -->* w where w ∈ Σ*) .

PPT(G) $=_{def}$ the set of all partial parse tree of G is the set of all trees corresponding to all possible derivations (i.e., A -->* α , where A ∈ N and α ∈ (NUΣ)*).

Inductive defintion of PPT(G) and PT(G)

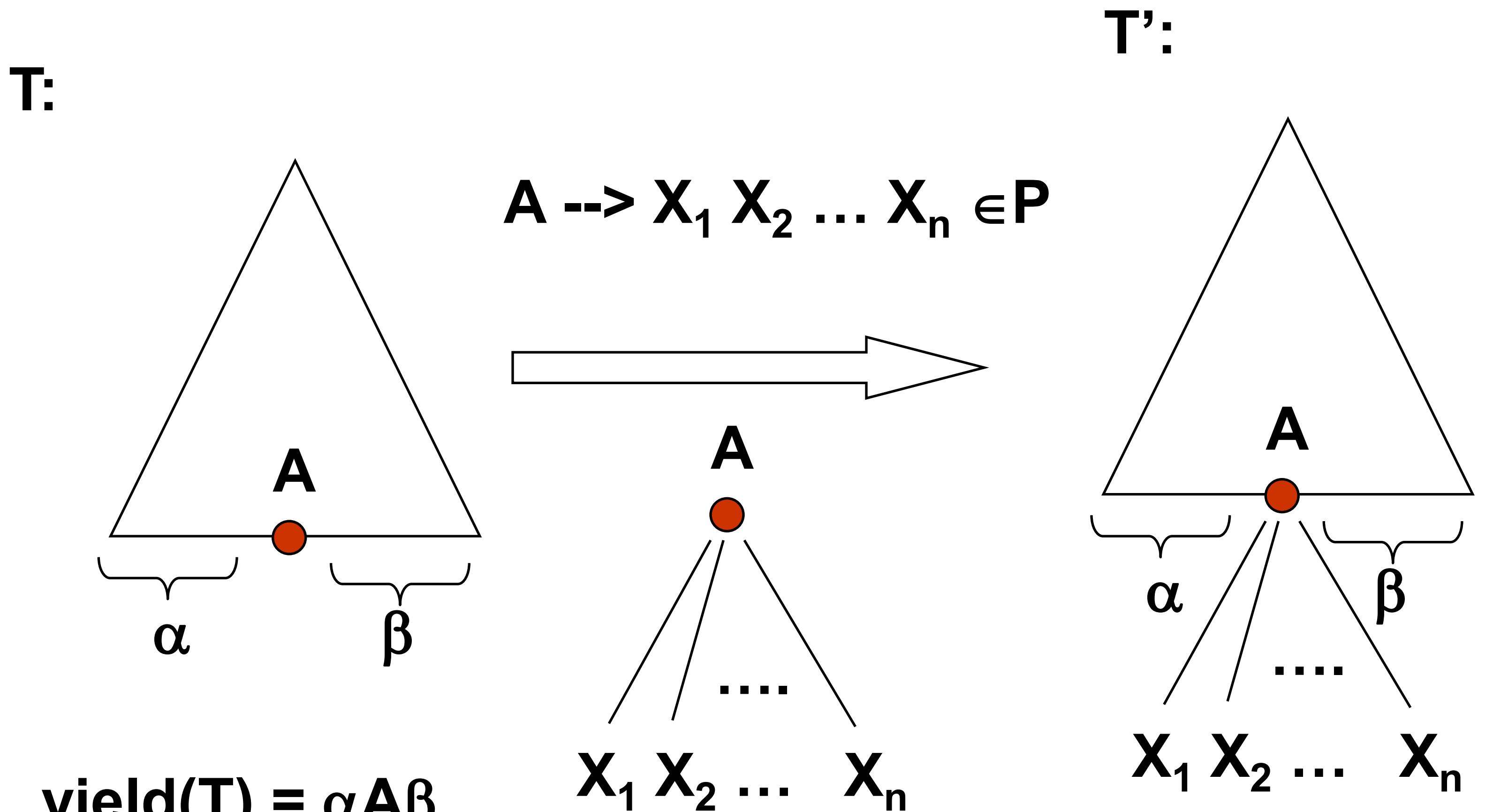
- The set PPT(G) and PT(G) as well as two functions root and yield on PPT(G) are defined inductively as follows:

1. Every nonterminal A is a PPT (with root A and yield A)

2. If **T = (… A …)** is a PPT where A is a nonterminal leaf and T has yield $\alpha A \beta$. and A --> $X_1X_2…X_n$ ($n \geq 0$) is a production, then the tree **T' = (…. (A X_1 X_2 …X_n) …)** obtained from T by appending $X_1…X_n$ to the leaf A as children of A is a PPT with yield α $X_1…X_n$ β. (See the figure in next slide)

- A PPT is called a partial X-tree if its root is labeled X.
- A PPT is a parse tree (PT) if its yield is a terminal string.

How a new PPT is defined in terms of old one



$yield(T) = \alpha A \beta$

$yield(T') = \alpha X_1 X_2 \ldots X_n \beta.$

Relations between parse trees and derivations

Lemma 4.1: If T is a partial X-tree with yield α, then X -->^*_G α.

Pf: proved by ind. on the structure(or number of nodes) of T.

Basis: T = X is a single-node PPT. Then α = X. Hence X -->^0_G α.

Ind: T = (… (A β) …) can be generated from T' = (…. A …) with yield $\mu A \nu$ by appending β to A. Then

X -->^*_G $\mu A \nu$ // by ind. hyp. on T'

-->_G $\mu \beta \nu$ // by def. A --> β in P QED.

- **Let D : X --> α_1 --> α_2 --> … --> α_n be a derivation.**

The partial X-tree generated from D, denoted T_D, which has yield(T_D) = α_n, can be defined inductively on n:

1. n = 0 : (i.e., D = X). Then T_D = X is a single-node PPT.
2. n = k+1 > 0: let D = [X --> α_1 --> … --> α_k = $\alpha A \beta$ -->α $X_1 \ldots X_m$ β] = [D' --> α $X_1 \ldots X_m$ β]

then T_D = $T_{D'}$ with leaf A replaced by the PPT (A $X_1 \ldots X_m$)

Relations between parse trees and derivations (cont'd)

Lemma 4.2: D = [X --> α_1 --> α_2 --> … --> α_n] : a derivation. Then T_D is a partial X-tree with yield α_n.

Pf: Simple induction on n. left as an exercise.

- **Leftmost and rightmost derivations:**
- **G: a CFG. Two relations**
  - **^{L}-->$_G$ (leftmost derivation),**
  - **^{R}-->$_G$ (rightmost derivation) $\subseteq$ $(N \cup \Sigma)^+ \times (N \cup \Sigma)^*$ are defined as follows: For $\alpha,\beta \in (N \cup \Sigma)^*$**

1. α ^{L}-->$_G$ β iff $\exists x \in \Sigma^*$, $A \in N$, $\gamma \in (N \cup \Sigma)^*$ and A --> $\delta \in$ P s.t.
   $\alpha = xA\gamma$ and $\beta = x\delta\gamma$. // $xA\gamma \rightarrow x\delta\gamma$
2. α ^{R}-->$_G$ β iff $\exists x \in \Sigma^*$, $A \in N$, $\gamma \in (N \cup \Sigma)^*$ and A --> $\delta \in$ P s.t.
   $\alpha = \gamma Ax$ and $\beta = \gamma\delta x$. // $\gamma Ax \rightarrow \gamma\delta x$
3. define ^{L}-->$^*_G$ (resp., ^{R}-->$^*_G$) as the ref. & trans. closure of ^{L}-->$_G$ (^{R}-->$_G$) .

parse tree and leftmost/rightmost derivations

- Ex: S --> SS | (S) | ε. Then

  (SSS) -->$_G$ ((S) SS) leftmost

  -->$_G$ (SS(S)) rightmost

  -->$_G$ (S (S) S) neither leftmost nor rightmost

Theorem 3 : G; a CFG, A ∈ N, w ∈ Σ*. Then the following statements are equivalent:

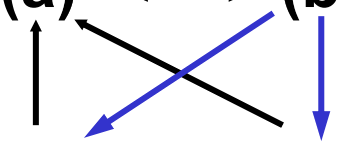
(a) A -->*$_G$ w.

(b) ∃ a parse tree with root A and yield w.

(c) ∃ a leftmost derivation A ^{L}-->*$_G$ w

(d) ∃ a rightmost derivation A ^{R}-->*$_G$ w

pf: (a) <==> (b) // (a) <==> (b) direct from Lemma 4.1 & 4.2.



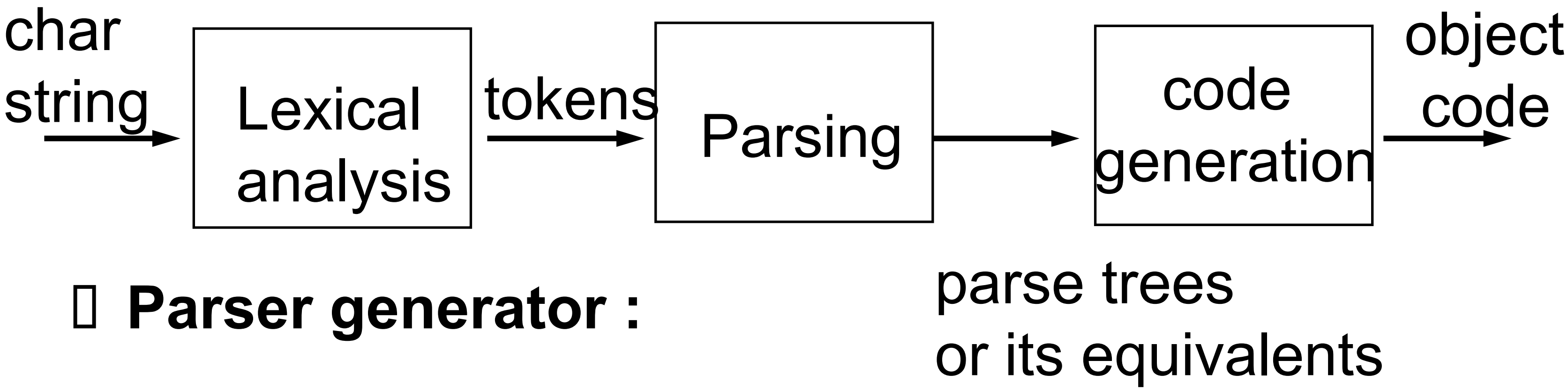
// (c),(d) ==> (a) : by definition

(c) (d) // need to prove (b) ==>(c),(d) only.

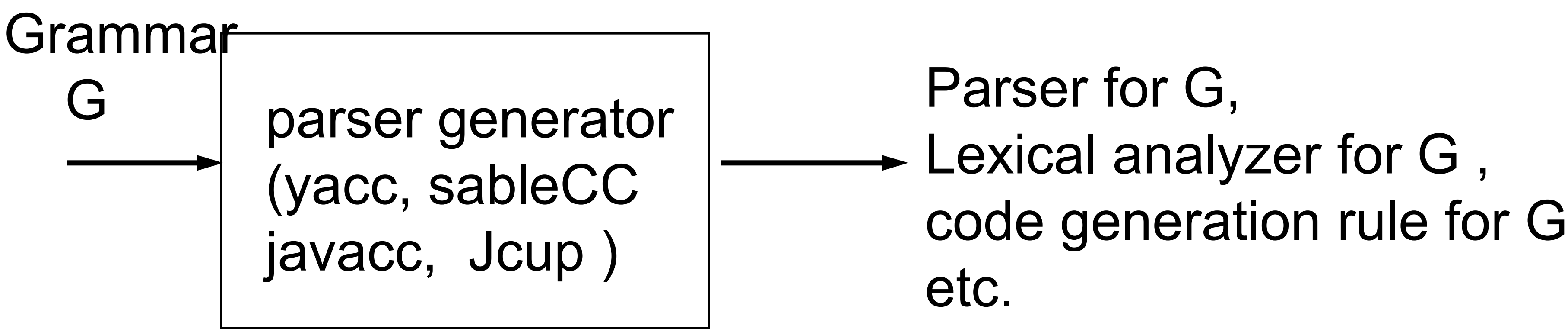
// left as an exercise.

Parsing

- **Major application of CFG & PDAs:**
  - **Natural Language Processing(NLP)**
  - **Programming language, Compiler:**
  - **Software engineering : text 2 structure**



  - **Parser generator :**

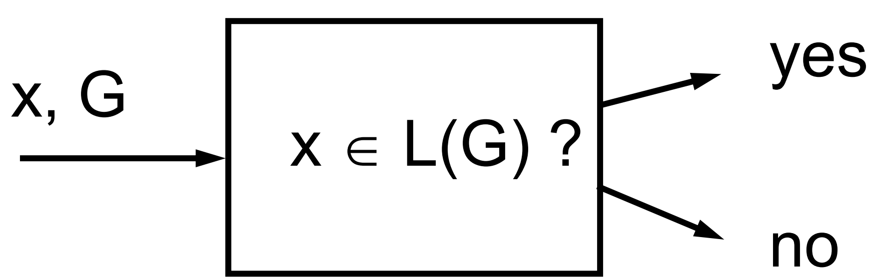


Parsing (cont'd)

- **Parsing is the process of generating a parse tree (or its equivalents) corresponding to a given input string w and grammar G.**

Note: In formal language we are only concerned with if $w \in L(G)$, but in compiler , we also need to know how w is derived from S (i.e., we need to know the parse tree if it exists).

- **A general CFG parser:**
  - **a program that can solve the problem:**
  - **x: any input string; G: a CFG**



The CYK algorithm

- **A general CFG parsing algorithm**
  - **run in time O(|x|^3).**
  - **using dynamic programming (DP) technique.**
  - **applicable to general CFG**
  - **but our demo version requires the grammar in Chomsky normal form.**
- Example : G =

S --> AB | BA | SS | AC | BD

A --> a B --> b C --> SB D --> SA

Let x = aabbab, n = |x| = 6.

Steps: 1. Draw n+1 vertical bars separating the symbols of x and number them 0 to n:

```
| a | a | b | b | a | b |
0   1   3   3   4   5   6
```

The CYK algorithm (cont'd)

For each $0 \le i < j \le n$, Let

x_{ij} = the substring of x between bar i and bar j.

$T(i,j) = \{ X \in N \mid X \dashrightarrow_G x_{ij} \}$.

I.e., T(i,j) is the set of nonterminal symbols that can derive the substring x_{ij} .

note: $x \in L(G)$ iff $S \in T(0,n)$.

- The spirit of the algorithm is that the value T(0,n) can be computed by applying DP technique.

2. Build a table with C(n,2) entries as shown in next slide:

The CYK chart

- **The goal is to fill in the table with cell(i,j) = T(i,j).**
  Problem: how to proceed ?
  ==> diagonal entries can be filled in immediately !! (why ?)

S --> AB | BA | SS | AC | BD
A --> a B --> b
C--> SB D --> SA

i \ j	1	2	3	4	5	6
5						b
4					a	
3				b		
2			b			
1		a				
0	a					

Fill in the CYK chart:

- **Why C(4,5) = { A } ? since A --> a = x_{45}.**
- **Once the main diagonal entries were filled in, the next lower diagonal entries can be filled in. (why ?)**

S --> AB | BA | SS | AC | BD
A --> a B --> b
C--> SB D --> SA

i \ j	1	2	3	4	5	6
5						B
4					A	
3				B		
2			B			
1		A				
0	A					

how to fill in the CYK chart

- $T(3,5) = S$ since $x_{35} = x_{34}\, x_{45}$ <-- $T(3,4)\ T(4,5) = B\ A$ <-- S
- In general $T(i,j) = \bigcup_{i < k < j} \{ X \mid X \to Y Z \mid Y \in T(i,k), Z \in T(k,j)\}$

S --> AB | BA | SS | AC | BD
A --> a B --> b
C--> SB D --> SA

i \ j	1	2	3	4	5	6
5						B
4					A	S
3				B	S	
2			B	-		
1		A	S			
0	A	-				

the demo CYK version generalized

- Let $P_k = \{ X \rightarrow \alpha \mid X \rightarrow \alpha \in P \text{ and } |\alpha| = k \}$.
- Then $T(i,j) = \bigcup_{k>0} \bigcup_{i = t0 < t1 < t2 < \ldots < tk < j = t(k+1)} \{ X \mid X \rightarrow X_1 X_2 \ldots X_k \in P_k \text{ and for all } m < k+1 \; X_m \in T(t_m, t_{m+1}) \}$

S --> AB | BA | SS | AC | BD
A --> a   B --> b
C--> SB   D --> SA

i \ j	1	2	3	4	5	6
5						B
4					A	S
3				B	S	C
2			B	-	-	-
1		A	S	C	S	C
0	A	-	-	S	D	S^2

The CYK algorithm

```
// input grammar is in Chomsky normal form
1. for i = 0 to n-1  do {        // first do substring of length 1
    T(i,i+1) = {A | A → x_{i,i+1} ∈ P };

2. for m = 2 to n do                 // for each length m > 1
     for i = 0 to n - m do{          // for each substring of length m
        T(i, i + m ) = {};
        for j = i + 1 to i + m -1 do{   // for each break of the string
          for each rule of the form A --> BC do
           If B ∈ T(i,j) and C ∈ T(j,i+m) then
               T(i,i+m) = T(i,i+m) U {A}
  }}
```