Chapter 4 – Context Free Grammar and Context Free Languages

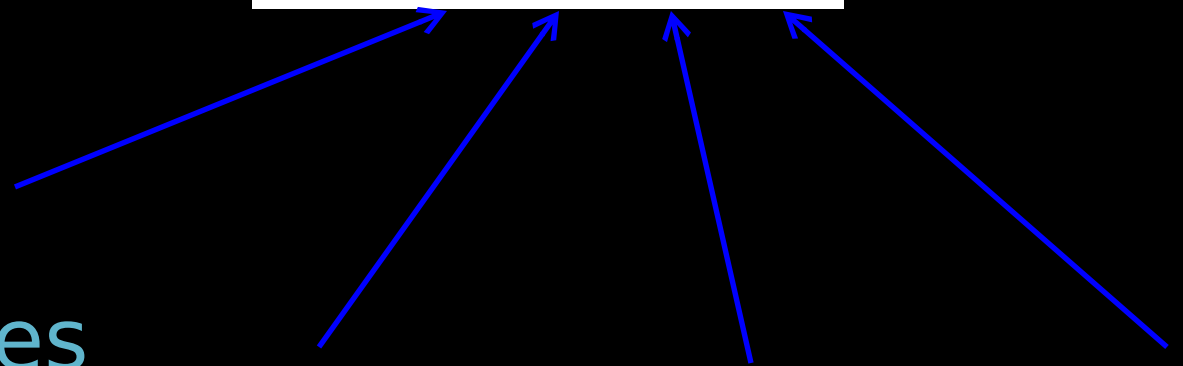
Outline

- CFGs and CFLs
- Parsing (or derivation) and Parse trees
- Ambiguity of grammar and language
- Simplification of CFGs
- Chmosky Normal Form (CNF)

Context-Free Grammars and Context-Free Languages

Formal Definitions

Grammar: $G = (V, T, P, S)$



Set of variables

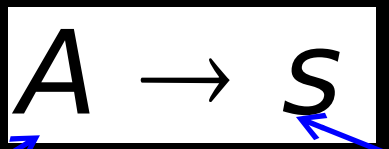
Set of terminal symbols

Set of productions

Start variable

Context-Free Grammar:

All productions in are of the form



Variable

String of variables and terminals

* The restriction on the right side of the productions made on regular grammar regarding the number and position of variables are relaxed.

Example of Context-Free Grammar

$$S \rightarrow aSb \mid \lambda$$

productions

$$P = \{S \rightarrow aSb,\ S \rightarrow \lambda\}$$

$$G = (V, T, P, S)$$

variables

terminals

start variable

Language of a Grammar:

For a grammar with start variable

$$L(G) = \{w: \ S \overset{*}{\Rightarrow} w, \quad w \in T^*\}$$

String of terminals or λ

Context-Free Language:

A language is context-free
if there is a context-free grammar
with

Example:

$$L = \{a^n b^n : \ n \geq 0\}$$

is a context-free language

since context-free grammar :

$$S \rightarrow aSb \mid \lambda$$

generates $L(G) = L$

Another Example

Context-free grammar :

$$S \rightarrow aSa \mid bSb \mid \lambda$$

Example derivations:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abb$$

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abaSaba \Rightarrow abaaab$$

$$L(G) = \{ww^R : \ w \in \{a,b\}^*\}$$

Palindromes of even length

Another Example

Context-free grammar :

$$S \rightarrow aSb \mid SS \mid \lambda$$

Example derivations:

$S \Rightarrow SS \Rightarrow aSbS \Rightarrow abS \Rightarrow ab$

$S \Rightarrow SS \Rightarrow aSbS \Rightarrow abS \Rightarrow abaSb \Rightarrow abab$

Describes balanced parentheses:

A core idea of most programming languages.

() ((( ))) (( ))

$a = (, \quad b =)$

Another Example

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

Is ab, baba, abbbaa in L?
How about a, bba,abbaba?
Show with the possible derivations.

Another Example

Construct a grammar that generates the language:

$L=\{a^n b^n c^i | n \geq 0, i \geq 0\}$

Answer:

$S \rightarrow S_1 S_2$

$S_1 \rightarrow aS_1b | \varepsilon$

$S_2 \rightarrow cS_2 | \varepsilon$

Check your answer by deriving abc, aabbc, aaabbbcccc

Another Example

Write the productions of CFG that accept the language:

$L = \{ab\ (bbaa)^n\ bba \mid n \geq 0\}$

Ans: $S \rightarrow S_1 S_2 S_3$

$S_1 \rightarrow ab$

$S_2 \rightarrow bbaaS_2$

$\mid \varepsilon$

$S_3 \rightarrow bba$

Ans: $S \rightarrow abABa$

$A \rightarrow bbaaA \mid$

ε

$B \rightarrow bb$

Derivation Order and Derivation Trees

Derivation Order

Consider the following example grammar with 5 productions:

1. $S \rightarrow AB$
2. $A \rightarrow aaA$
3. $A \rightarrow \lambda$
4. $B \rightarrow Bb$
5. $B \rightarrow \lambda$

1. $S \rightarrow AB$

2. $A \rightarrow aaA$

3. $A \rightarrow \lambda$

4. $B \rightarrow Bb$

5. $B \rightarrow \lambda$

Leftmost derivation order of string aab:

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} aaAB \overset{3}{\Rightarrow} aaB \overset{4}{\Rightarrow} aaBb \overset{5}{\Rightarrow} aab$$

At each step, we substitute the leftmost variable

1. $S \rightarrow AB$

2. $A \rightarrow aaA$

3. $A \rightarrow \lambda$

4. $B \rightarrow Bb$

5. $B \rightarrow \lambda$

Rightmost derivation order of string aab :

$$S \overset{1}{\Rightarrow} AB \overset{4}{\Rightarrow} ABb \overset{5}{\Rightarrow} Ab \overset{2}{\Rightarrow} aaAb \overset{3}{\Rightarrow} aab$$

At each step, we substitute the rightmost variable

1. $S \to AB$

2. $A \to aaA$

3. $A \to \lambda$

4. $B \to Bb$

5. $B \to \lambda$

Leftmost derivation of aab:

$$S \overset{1}{\Rightarrow} AB \overset{2}{\Rightarrow} aaAB \overset{3}{\Rightarrow} aaB \overset{4}{\Rightarrow} aaBb \overset{5}{\Rightarrow} aab$$

Rightmost derivation of aab:

$$S \overset{1}{\Rightarrow} AB \overset{4}{\Rightarrow} ABb \overset{5}{\Rightarrow} Ab \overset{2}{\Rightarrow} aaAb \overset{3}{\Rightarrow} aab$$

Derivation Trees

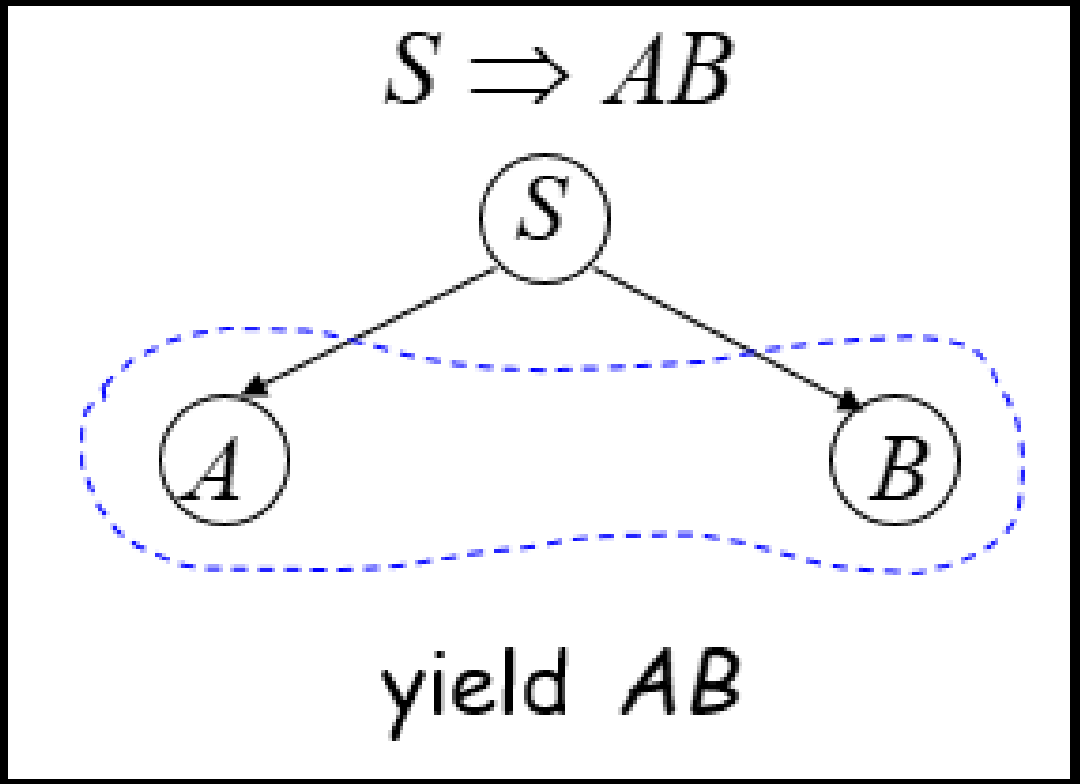
Consider the same example grammar:

$S \rightarrow AB$ $A \rightarrow aaA \mid \lambda$ $B \rightarrow Bb \mid \lambda$

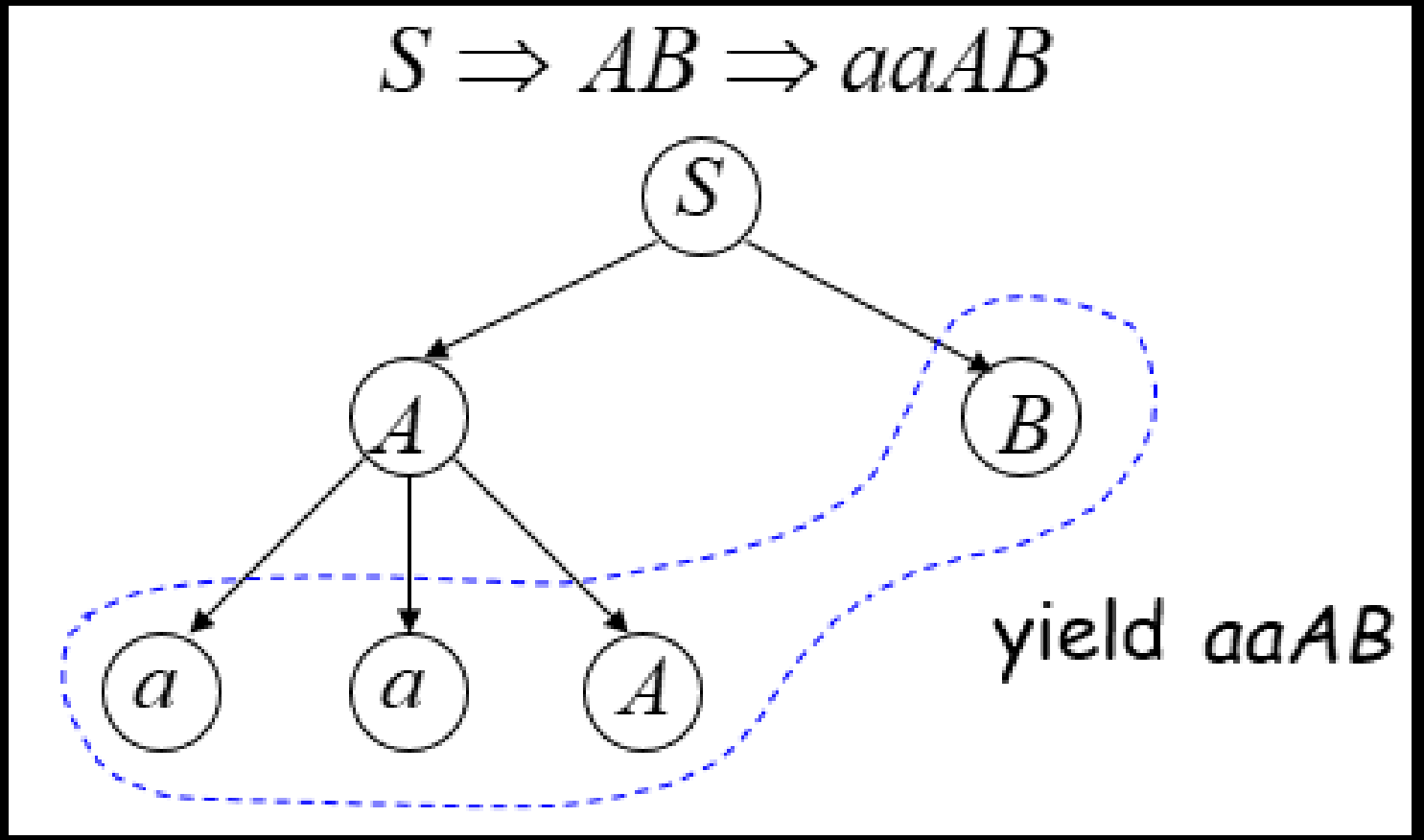
And a derivation of aab:

$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb \Rightarrow aaBb \Rightarrow aab$

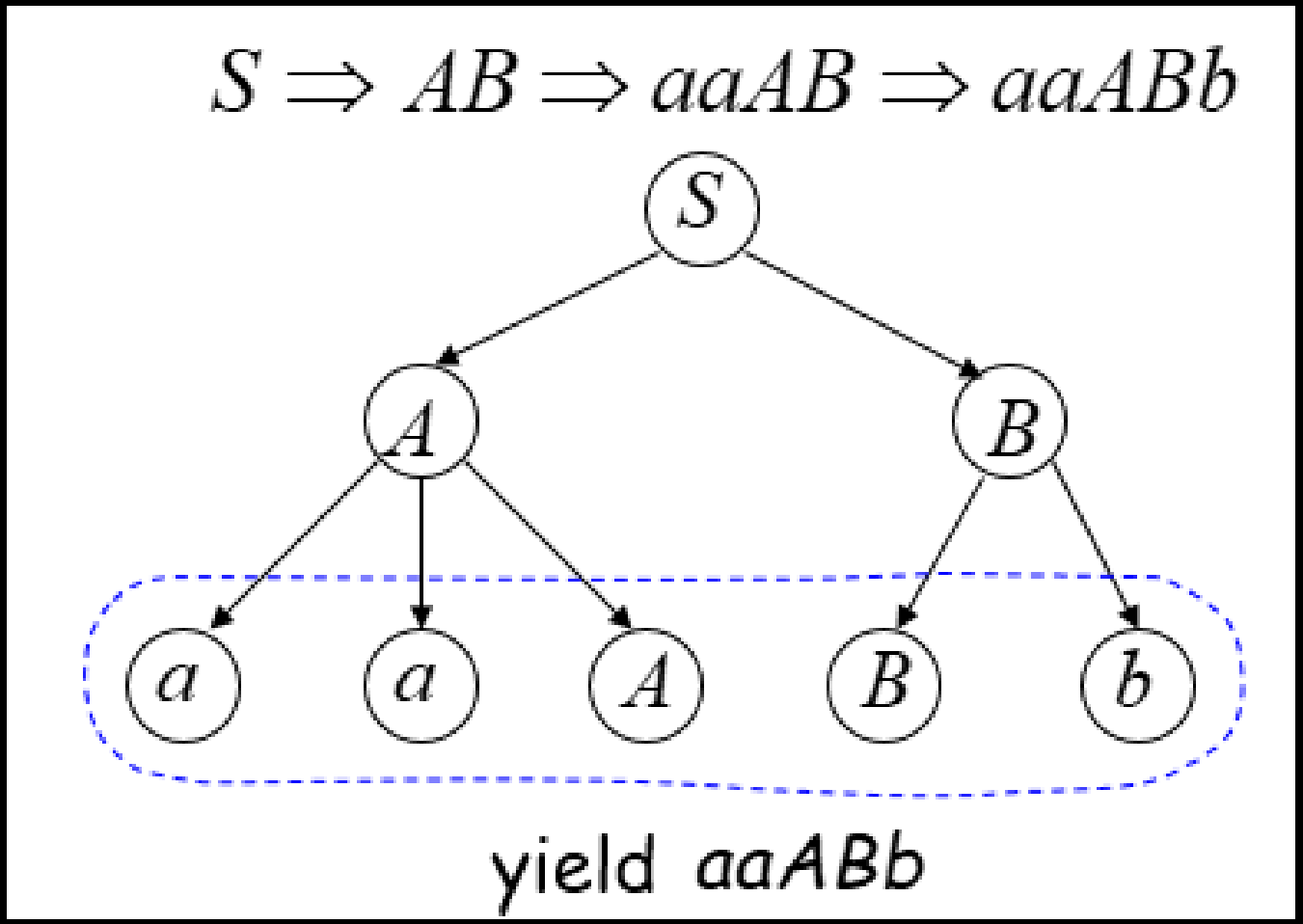
$S \rightarrow AB$ $A \rightarrow aaA|\lambda$ $B \rightarrow Bb|\lambda$



$S \rightarrow AB$ $A \rightarrow aaA|\lambda$ $B \rightarrow Bb|\lambda$

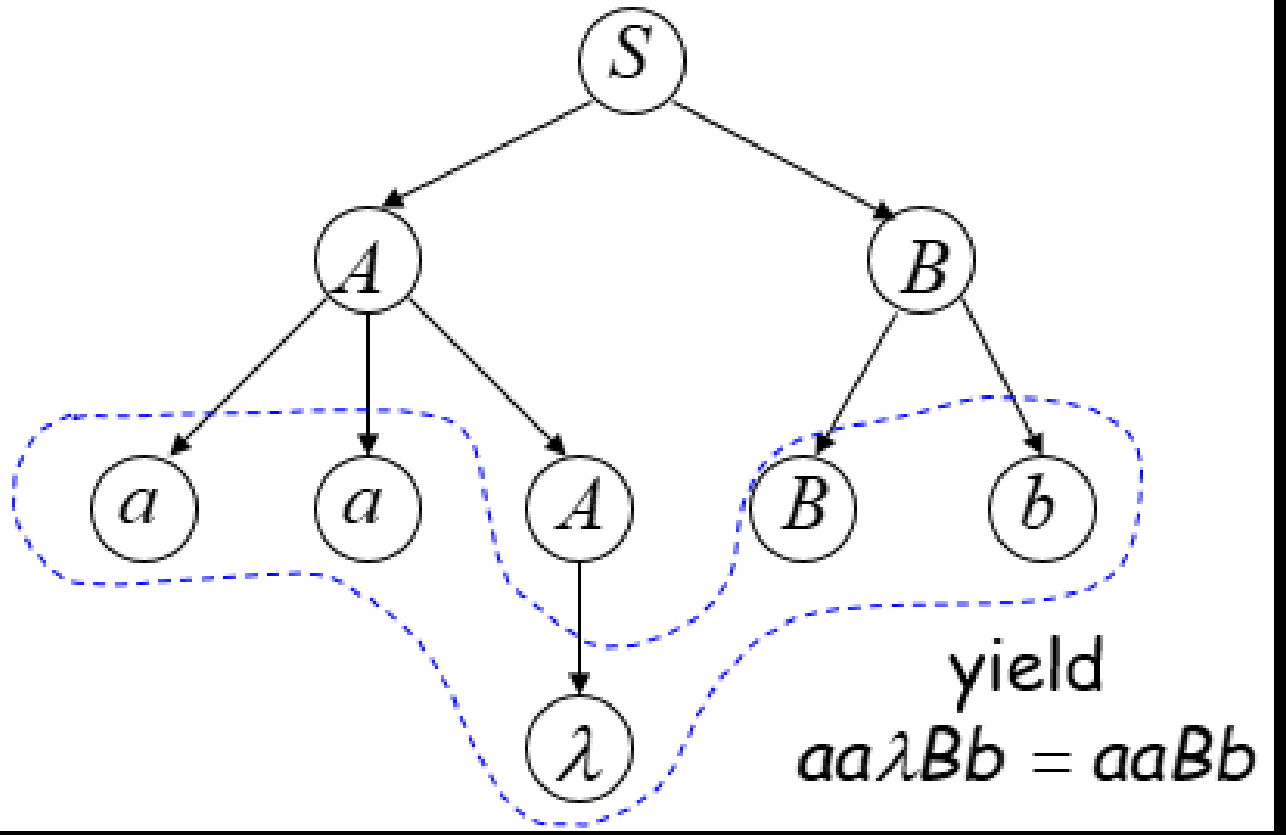


$S \rightarrow AB$ $A \rightarrow aaA|\lambda$ $B \rightarrow Bb|\lambda$



$S \rightarrow AB$ $A \rightarrow aaA|\lambda$ $B \rightarrow Bb|\lambda$

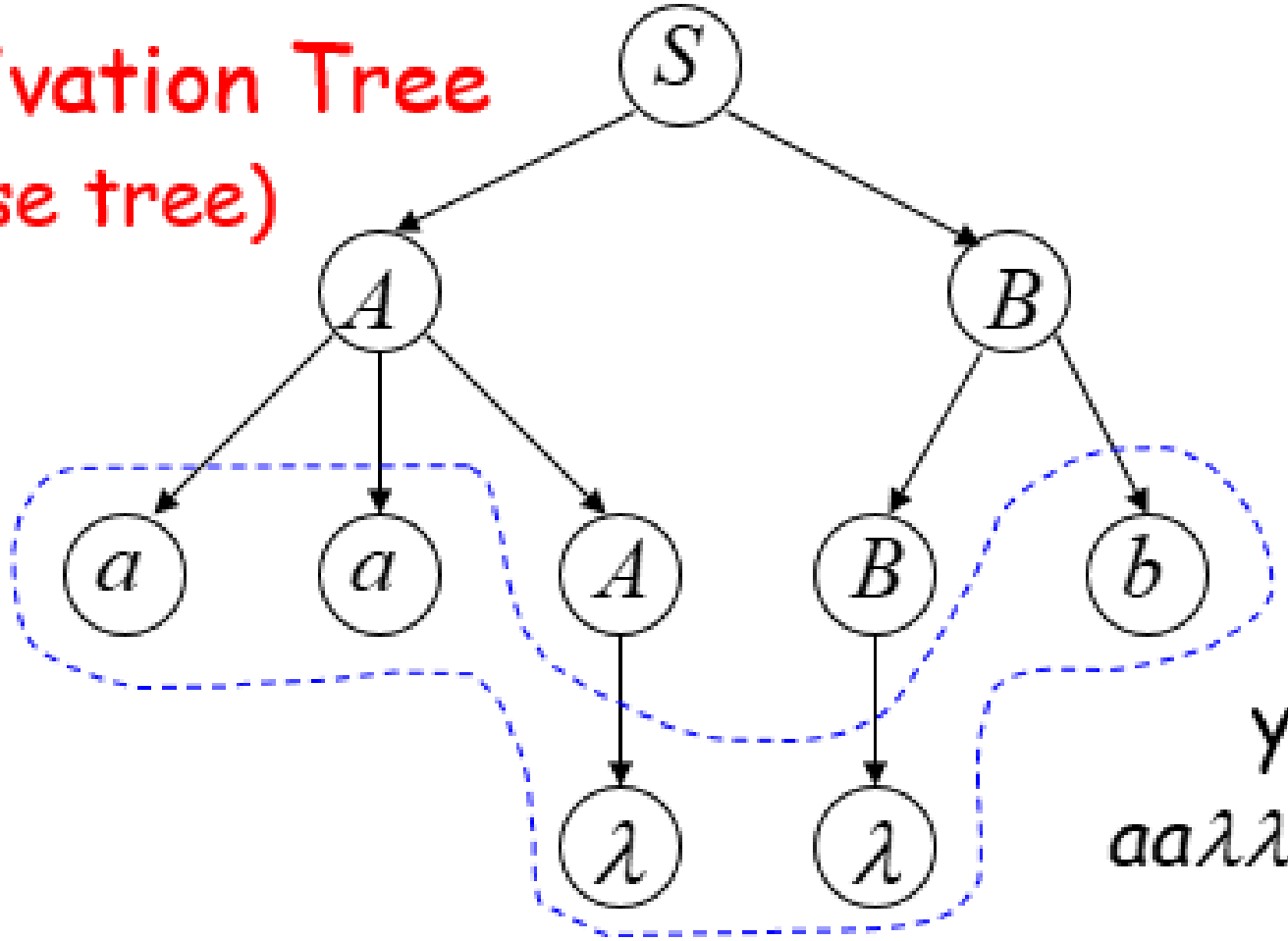
$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb \Rightarrow aaBb$



$S \rightarrow AB$ $A \rightarrow aaA|\lambda$ $B \rightarrow Bb|\lambda$

$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb \Rightarrow aaBb \Rightarrow aab$

Derivation Tree (parse tree)



yield

$aa\lambda\lambda b = aab$

$S \rightarrow aAS \mid a \quad A \rightarrow SbA \mid SS \mid ba$

Consider the CFG with the above production rules:

Find the Left most & Right most derivations and construct a parse tree (Generation tree) for generating the strings:

i) aabaaa

ii) abaabaa

iii) aabaabbaaaa

Sometimes, derivation order doesn't matter

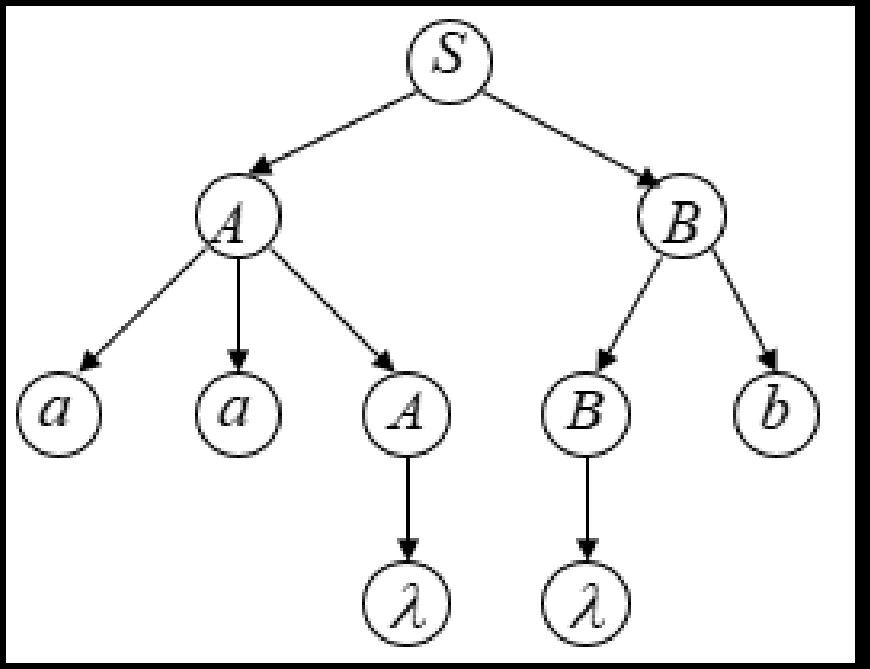
Leftmost derivation:

$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aaBb \Rightarrow aab$$

Rightmost derivation:

$$S \Rightarrow AB \Rightarrow ABb \Rightarrow Ab \Rightarrow aaAb \Rightarrow aab$$

Give same derivation tree



Ambiguity

Grammar for mathematical expressions

$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

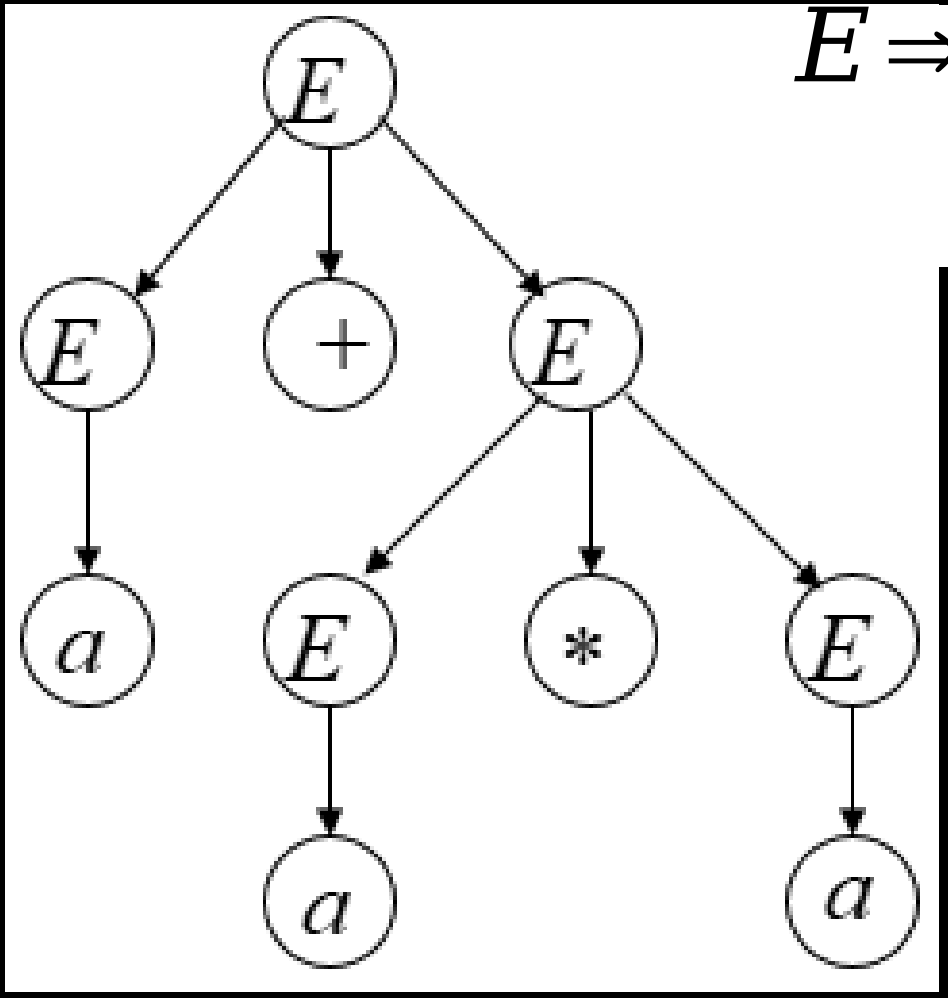
Example strings:

$$(a + a) * a + (a + a * (a + a))$$



Denotes any number

$$E \rightarrow E+E \mid E*E \mid (E) \mid a$$



$$E \Rightarrow E+E \Rightarrow a+E \Rightarrow a+E*E$$

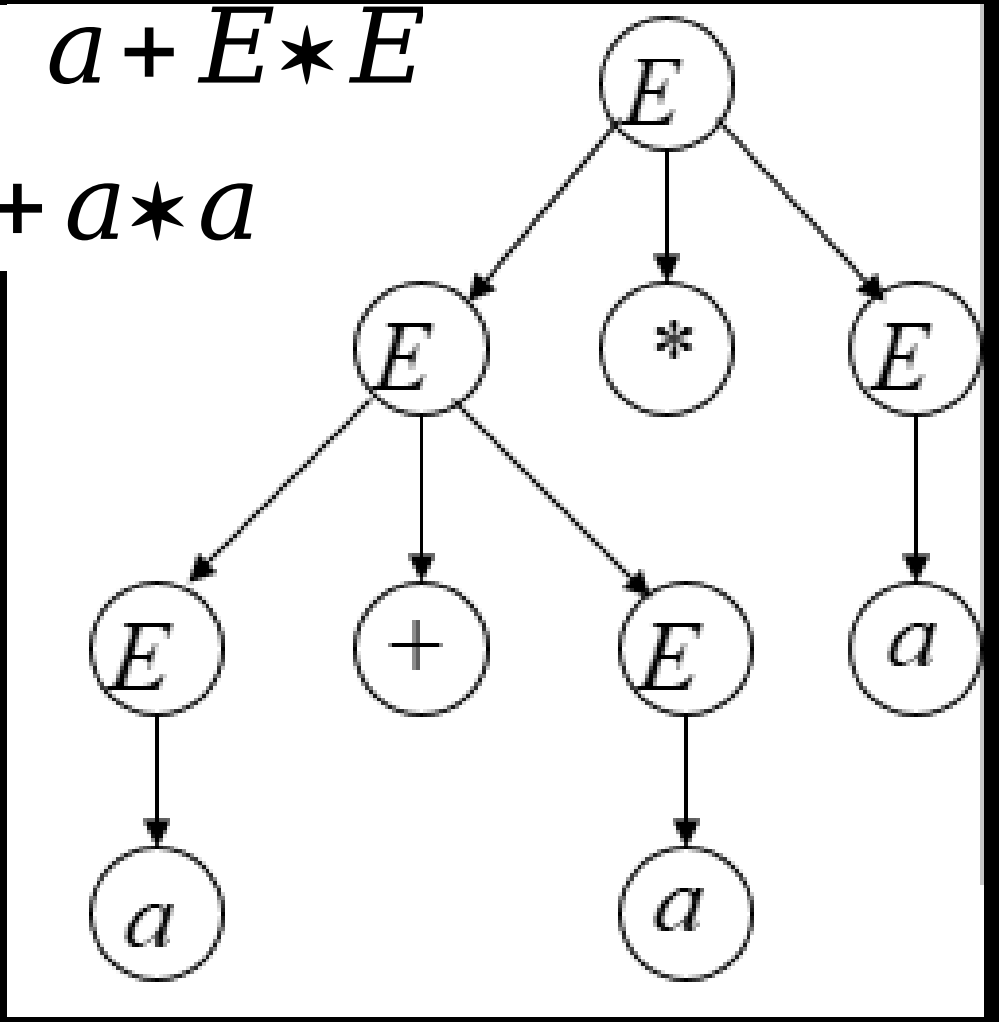
$$\Rightarrow a+a*E \Rightarrow a+a*a$$

A leftmost derivation for $a+a*a$

$$E \rightarrow E+E \mid E*E \mid (E) \mid a$$

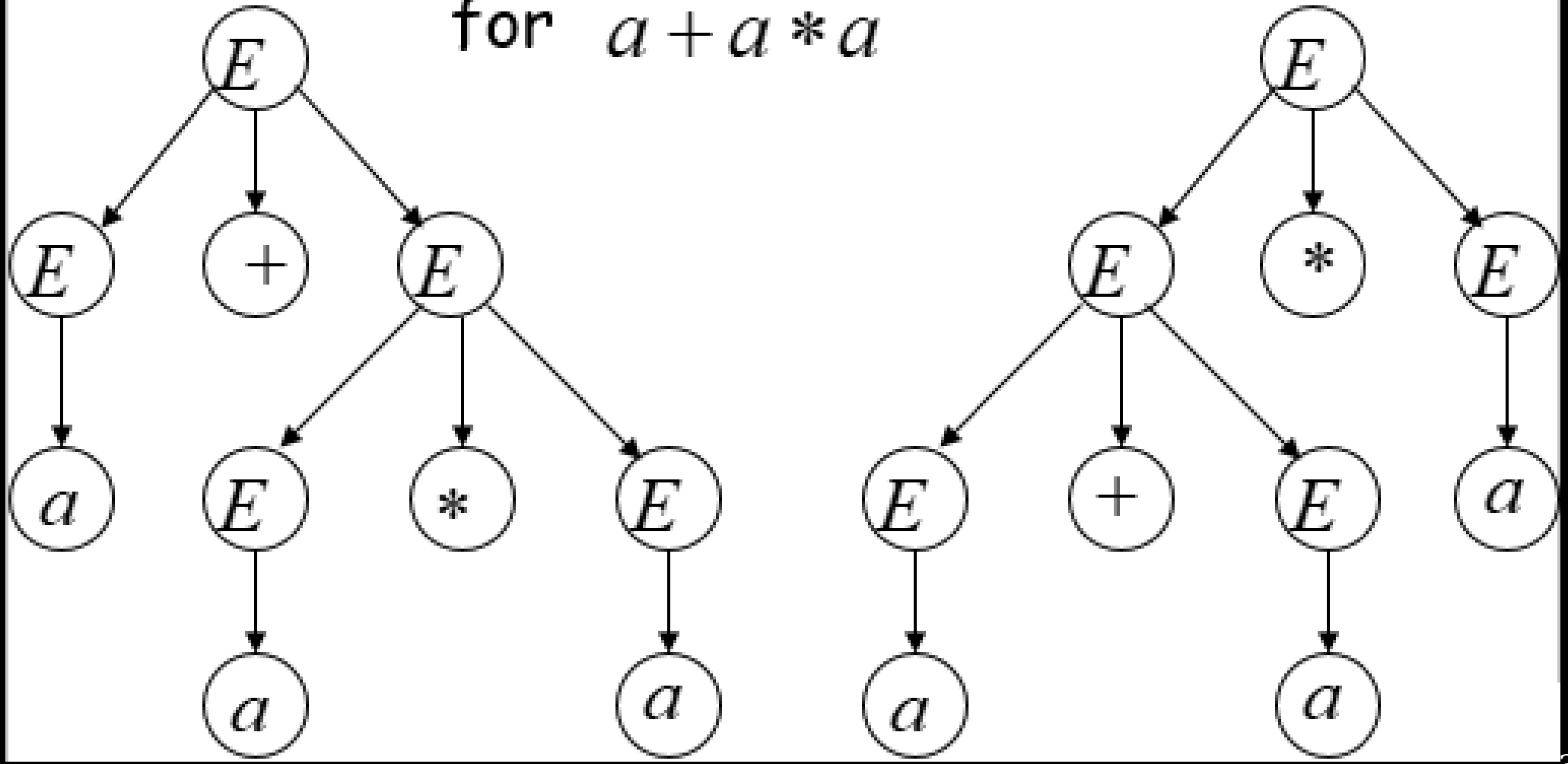
$$E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow a+E*E$$
$$\Rightarrow a+a*E \Rightarrow a+a*a$$

Another leftmost derivation for $a+a*a$



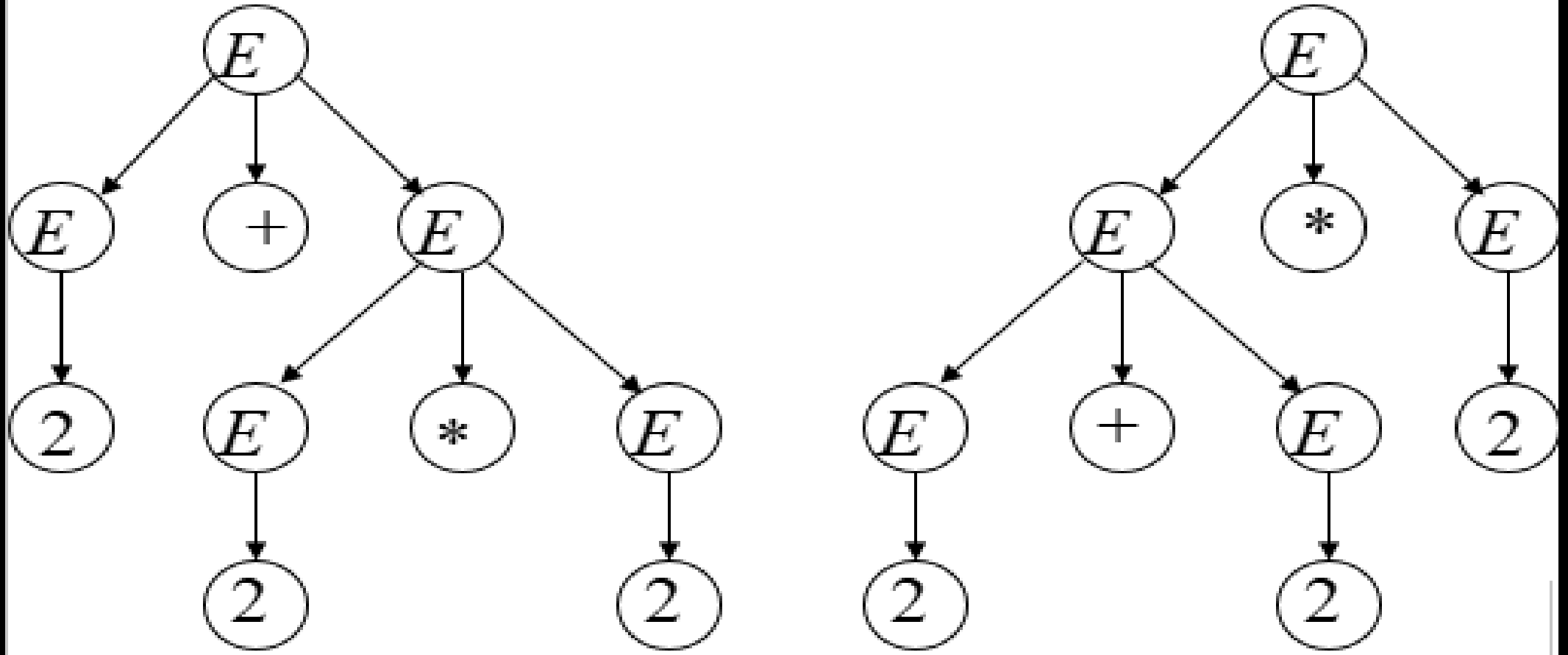
$$E \rightarrow E+E \mid E*E \mid (E) \mid a$$

Two derivation trees for $a+a*a$



take $a = 2$

$$a + a * a = 2 + 2 * 2$$



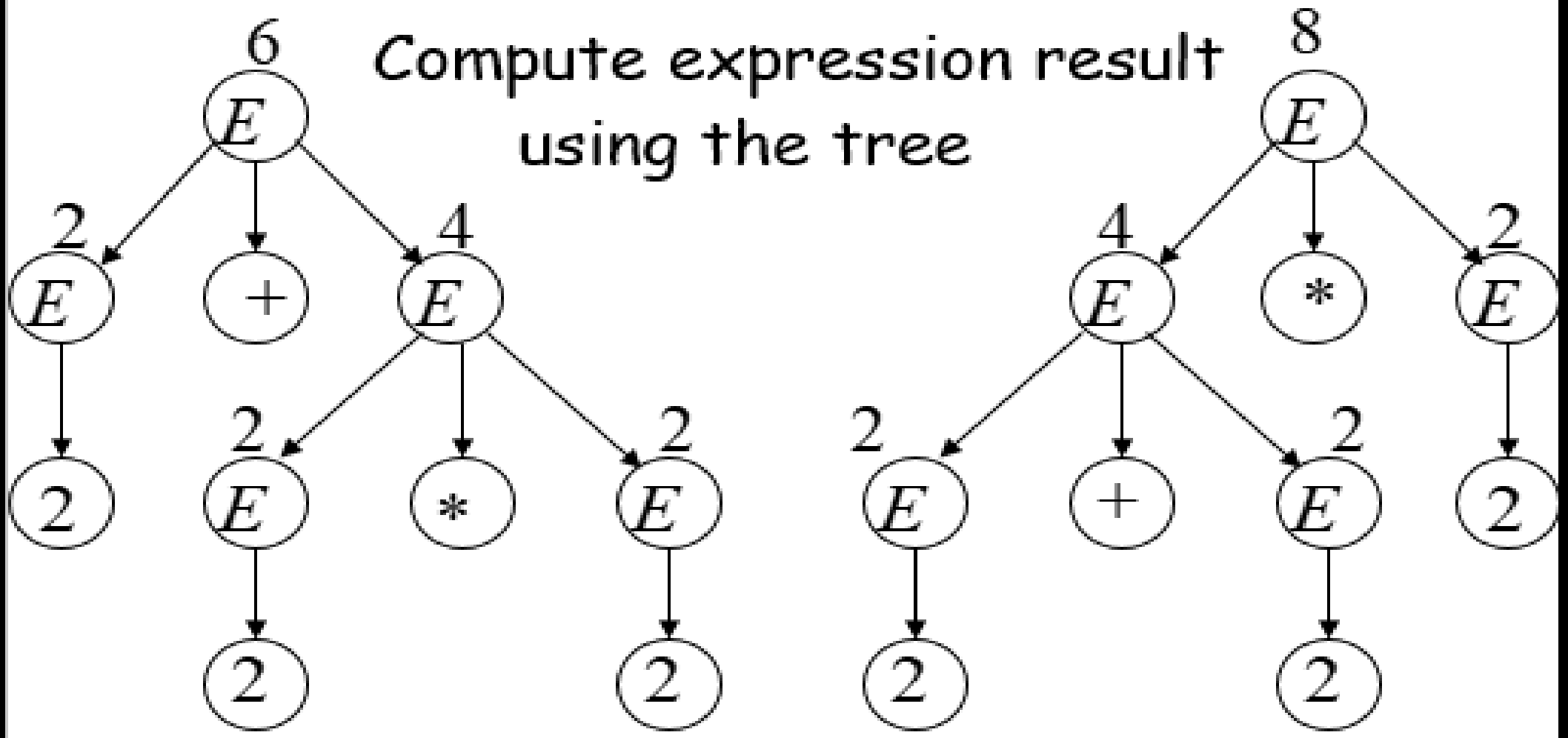
Good Tree

$2+2*2=6$

Bad Tree

$2+2*2=8$

Compute expression result using the tree



Two different derivation trees
may cause problems in applications which
use the derivation trees:

- Evaluating expressions

- In general, in compilers
  for programming languages

Ambiguous Grammar:

A context-free grammar G is ambiguous if there is a string $w \in L(G)$ which has:

two different derivation trees

or

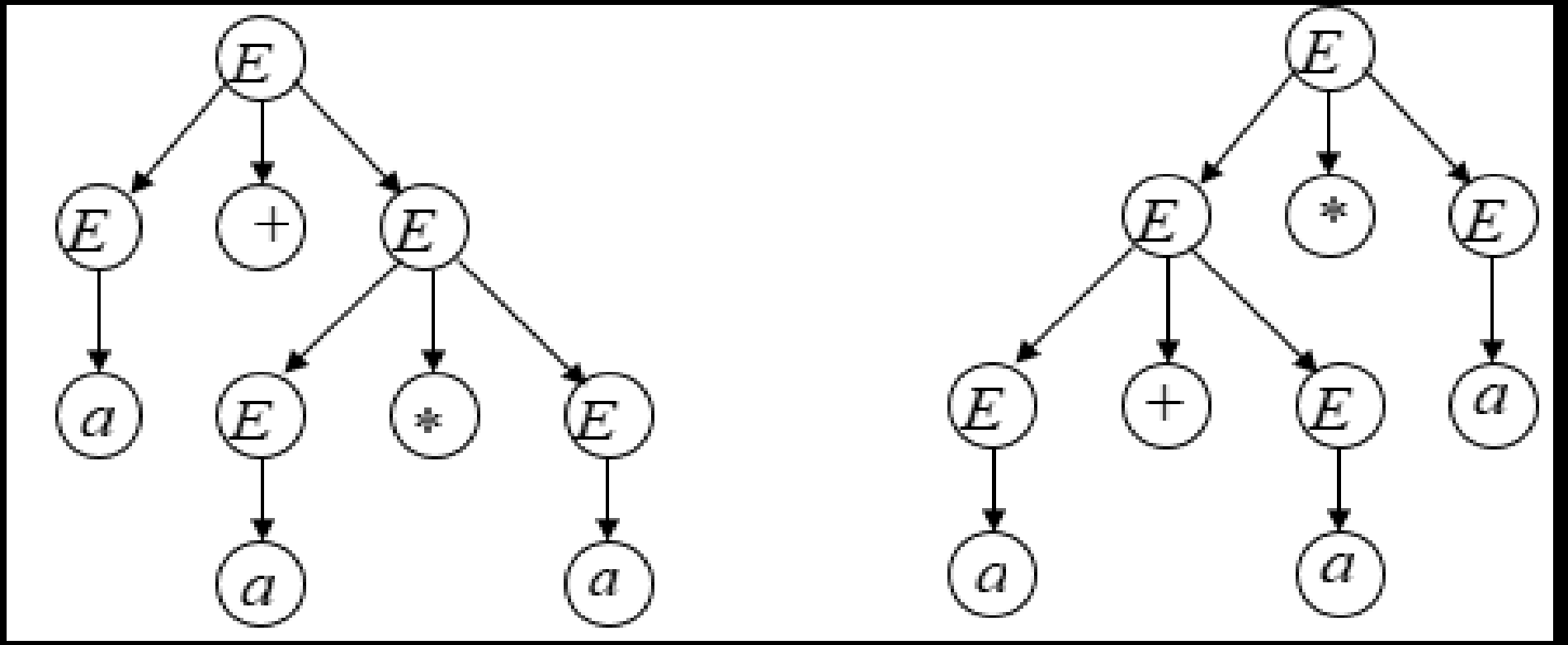
two leftmost derivations

(Two different derivation trees give two different leftmost derivations and vice-versa)

Example: $E \rightarrow E+E \mid E*E \mid (E) \mid a$

this grammar is ambiguous since

string $a+a*a$ has two derivation trees



$$E \rightarrow E+E \mid E*E \mid (E) \mid a$$

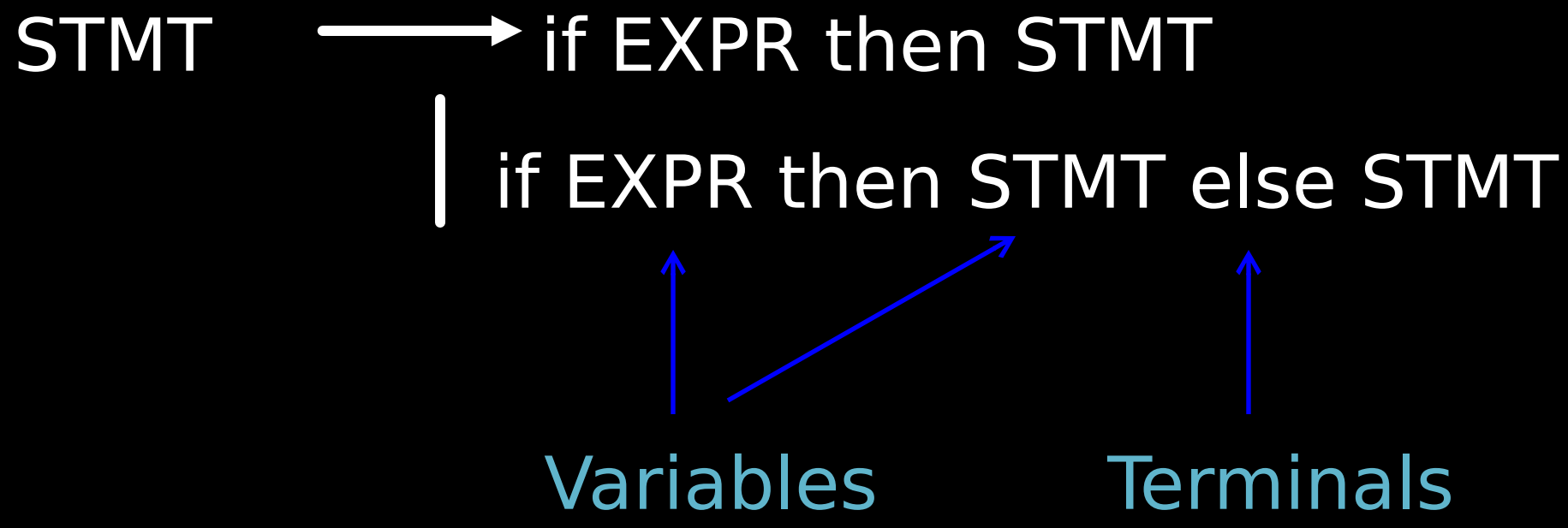
this grammar is also ambiguous because

string $a+a*a$ has two leftmost derivations

$$E \Rightarrow E+E \Rightarrow a+E \Rightarrow a+E*E$$
$$\Rightarrow a+a*E \Rightarrow a+a*a$$

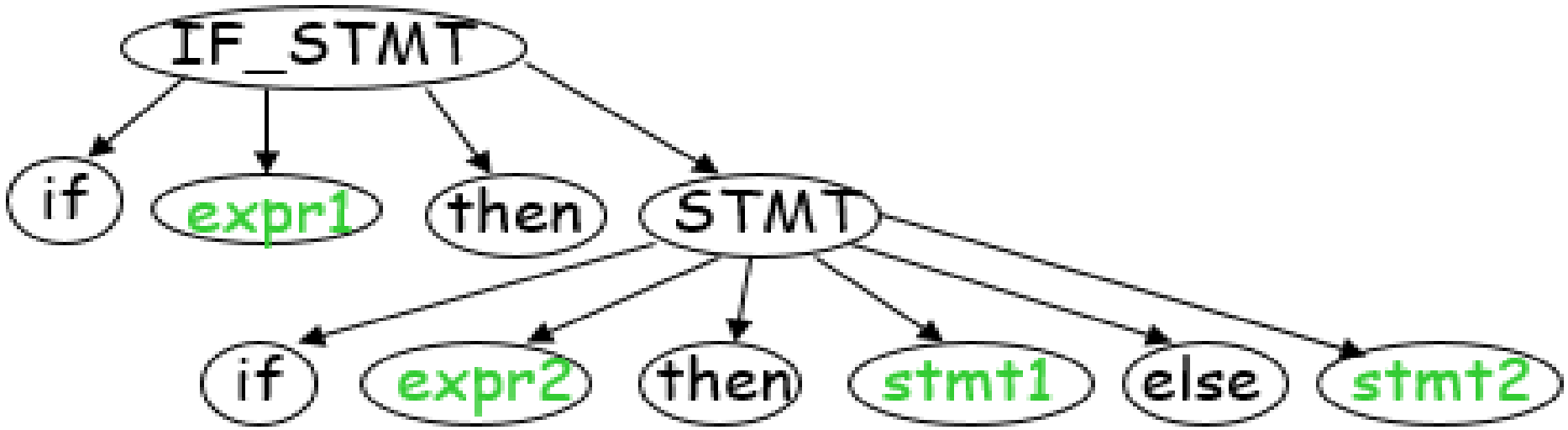
$$E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow a+E*E$$
$$\Rightarrow a+a*E \Rightarrow a+a*a$$

Another ambiguous grammar:

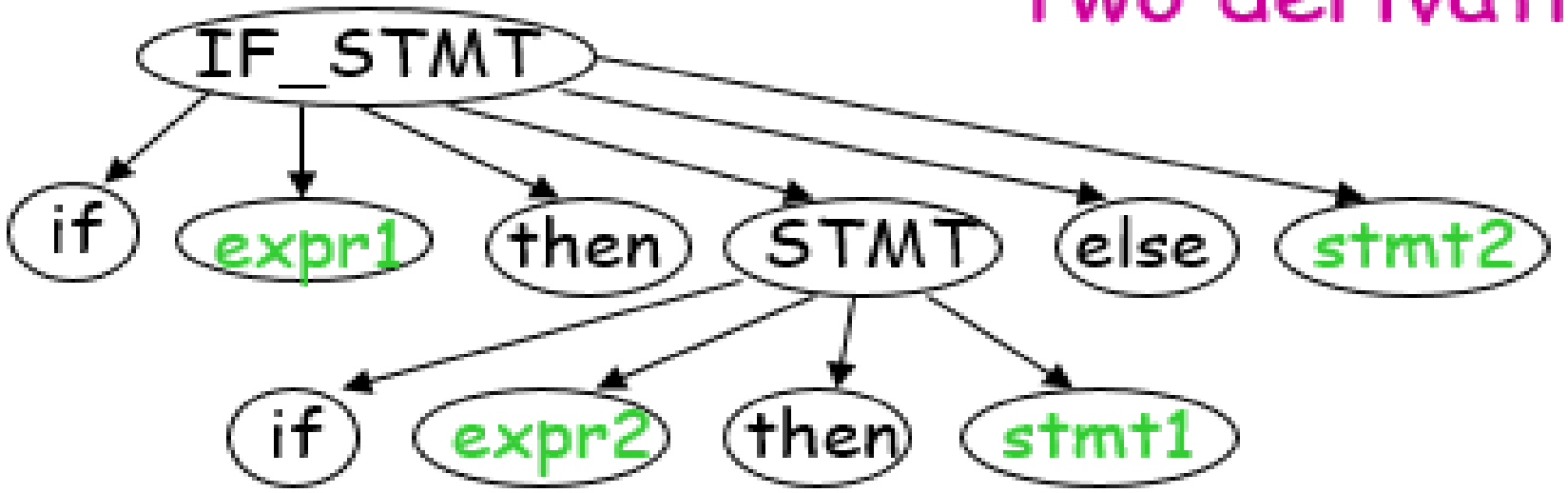


Very common piece of grammar
in programming languages

If expr1 then if expr2 then stmt1 else stmt2



Two derivation trees



In general, ambiguity is bad
and we want to remove it

Sometimes it is possible to find
a non-ambiguous grammar for a language

But, in general we cannot do so

Disambiguation

- Can we always disambiguate a grammar?
- No, for two reasons
  - There exists an inherently ambiguous context-free L:
    Every CFG for this language is ambiguous
  - There is no general procedure that can tell if a grammar is ambiguous
- However, grammars used in programming languages can typically be disambiguated

A successful example:

Ambiguous Grammar

$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow (E)$$
$$E \rightarrow a$$

Equivalent Non-Ambiguous Grammar

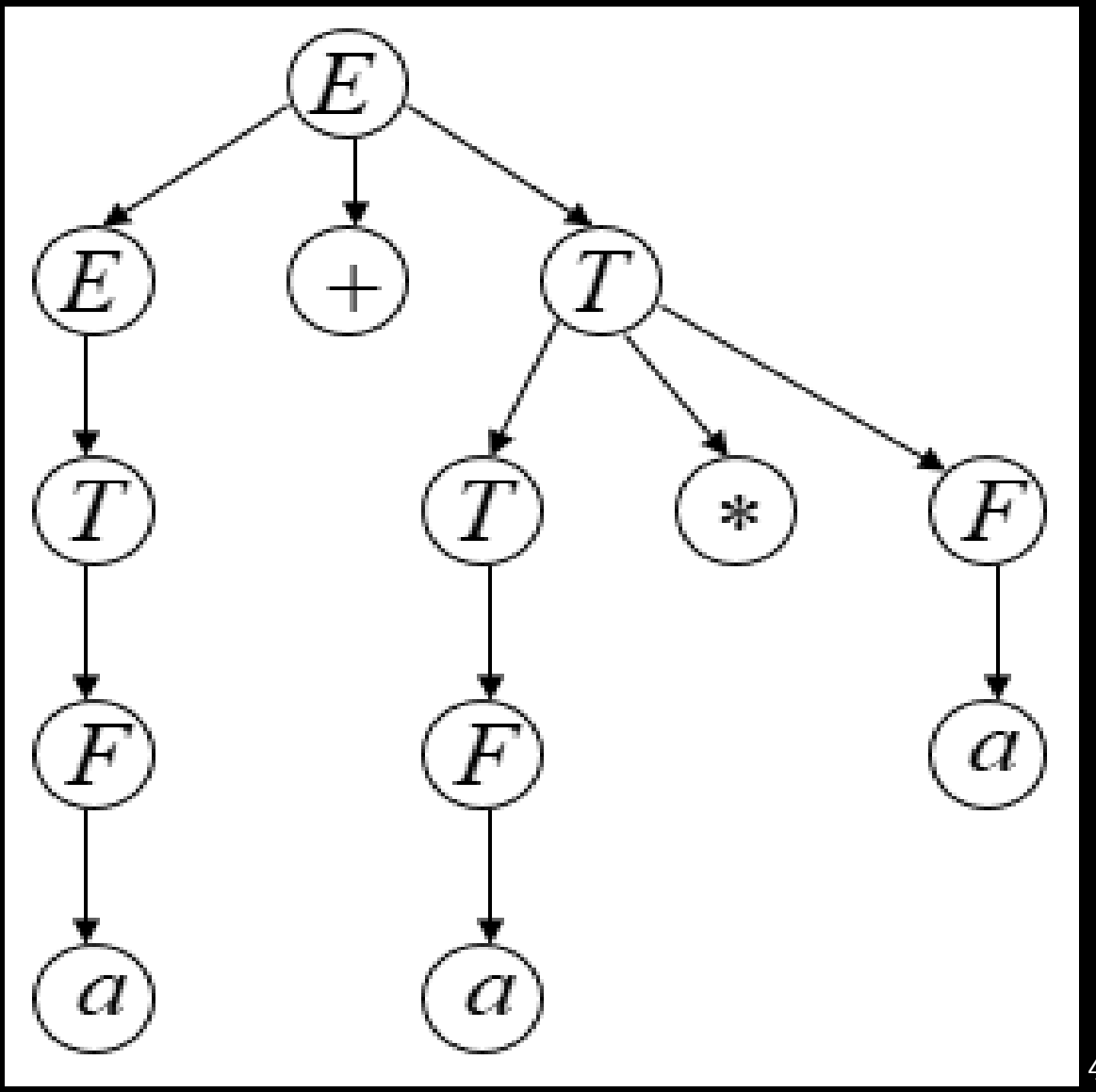
$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid a$$

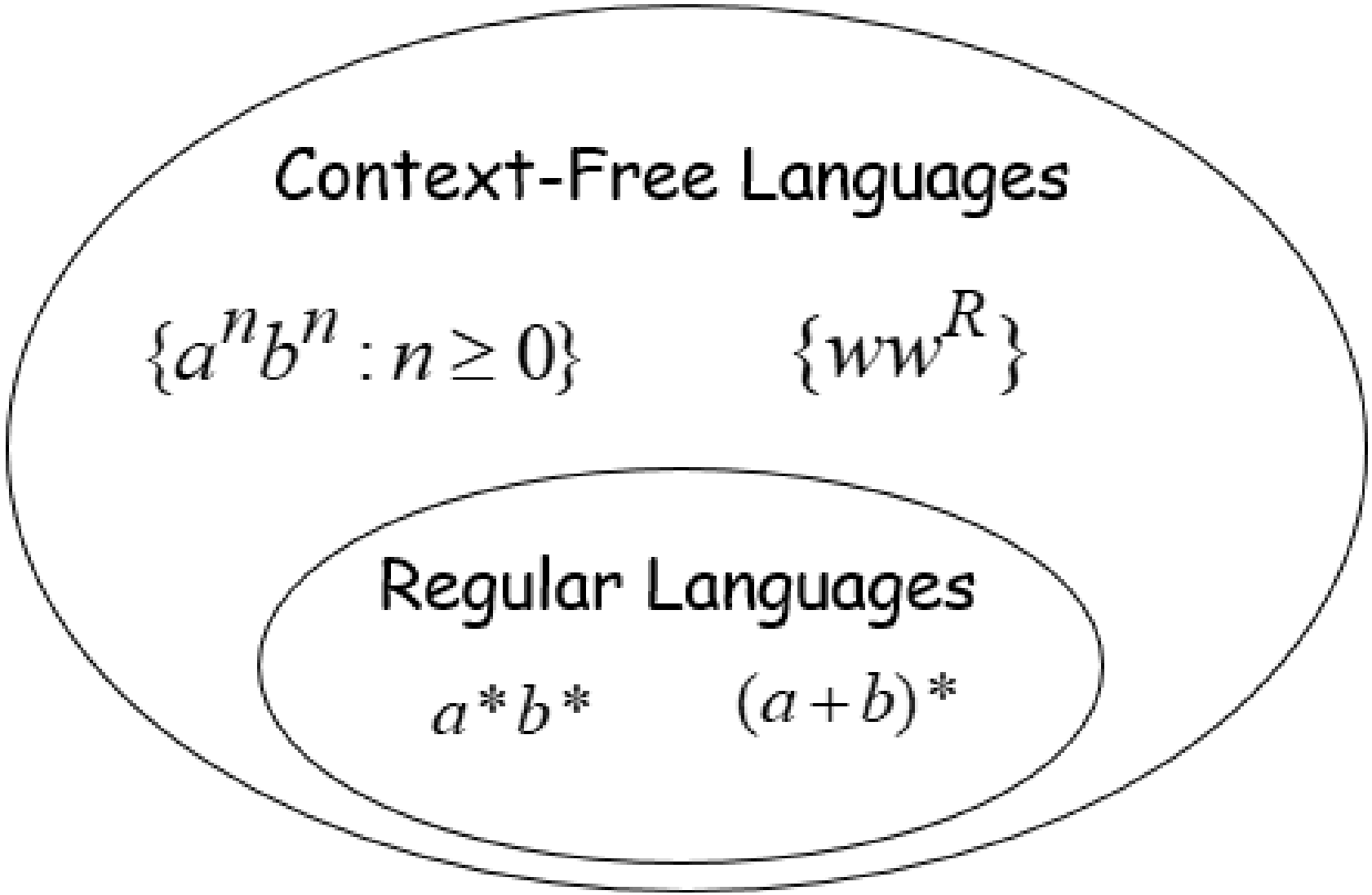
generates the same language

$$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow a+T \Rightarrow a+T*F$$
$$\Rightarrow a+F*F \Rightarrow a+a*F \Rightarrow a+a*a$$

$$E \rightarrow E+T \mid T$$
$$T \rightarrow T*F \mid F$$
$$F \rightarrow (E) \mid a$$

Unique derivation tree for $a+a*a$

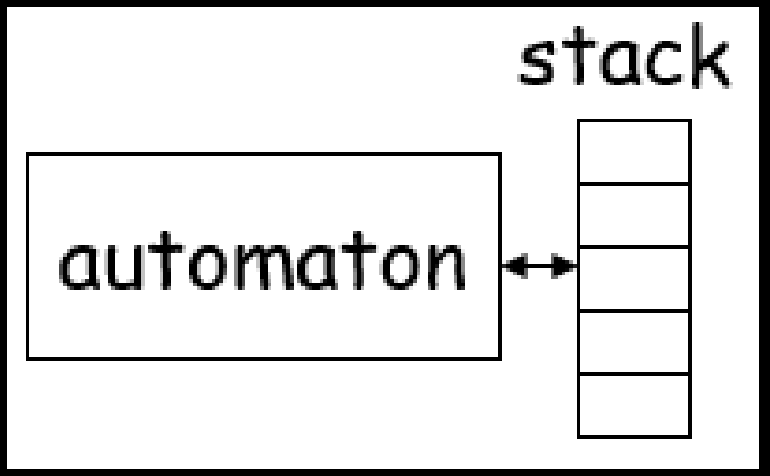



Context-Free Languages
$\{a^n b^n : n \geq 0\}$
$\{ww^R\}$
Regular Languages
$a^* b^*$
$(a+b)^*$

Context-Free Languages

Context-Free Grammars

Pushdown Automata



Simplification of CFG and Chomsky Normal Form

Simplifying CFGs

- There are several ways in which context-free grammars can be simplified.
- One natural way is to eliminate useless symbols
  - those that cannot be part of a derivation (or parse tree)

- Symbols may be useless in one of two ways.
  - they may not be reachable from the start symbol.
  - or they may be variables that cannot derive a string of terminals

Example of a useless symbol

- Consider the CFG G with rules
  - S → aBC, B → b|Cb, C → c|cC, D → d
- Here the symbols S, B, C, a, b, and c are reachable, but D is not.
- D may be removed without changing L(G)

Reachable symbols

- In a CFG, a symbol is reachable iff it is S or
  - it appears in α, where $A \rightarrow \alpha$ is a rule of the grammar, and A is reachable
- So in the grammar above
  - we first find that S is reachable
  - then a, B, and C are
  - and finally b and c are.
- A symbol that is unreachable cannot be part of a derivation. It may be eliminated along with all of its rules.

Another reachability example

- Suppose the grammar G instead had rules
  - S → aB, B → b|Cb, C → c|cC, D → d,
- Then we would first see that S is reachable
  - then that a and B are
  - then that b and C are
  - and finally that c is
- We might say in this case that
  - S is reachable at level 0,
  - a and B at level 1,
  - b and C at level 2,
  - and c at level 3.

A second kind of useless symbol

- Two simple inductions show that X is reachable iff S =*> $\alpha X \beta$ for some strings α and β of symbols.

- A symbol X is also useless iff it cannot derive a string of terminals
  - that is, iff there's no string w of terminals such that X =*> w.

Another simplification example

- In the grammar with rules
  - S → aB, B → b|BD|cC, C → cC, D → d
- the symbol C cannot derive a string of terminals.
- So it and all rules that contain it may be eliminated to get just
- S → aB, B → b|BD, D → d

Generating strings of terminals

- A simple induction shows that the only symbols that can generate strings of terminals are
  - terminal symbols
  - variable A for which A → α is a rule of the grammar and every symbol of α generates a string of terminals

Our example revisited

$$S \rightarrow aB,\ \ B \rightarrow b|BD|cC,\ \ C \rightarrow cC,\ \ D \rightarrow d$$

- In the grammar above, we would observe
  - first that a, b, c, and d generate strings of terminals (at level 0),
  - then that B and D do (at level 1),
  - and finally that S does (at level 2)

Removing the two kinds of useless symbols

- The characterizations of the two kinds of useless symbols are similar, except that
  - To find reachable symbols, we work top down
  - To find generating symbols, we work bottom up.
- When removing useless symbols, it's important to remove unreachable symbols last
  - since only this order will leave only useful symbols at the end of the process.

Example of removing useless symbols

- Using the algorithms implicit in the above characterizations, suppose a CFG has rules
  - S → aB, B → b|bB|CD, C → cC, D → d
- We first observe that a, b, c, and d generate strings of terminals (at level 0)
  - then that B and D do (at level 1)
  - and finally that S does (at level 2).
- But removing the rule B → CD from this grammar makes the symbol D unreachable.

Exercise on removing useless symbols

- Eliminate the useless variables from the grammar given by the production rules:

1) S → aS|A|C
   A → a
   B → aa
   C → aCb

Answer:

S → aS|A
A → a

2) S → aB|bC
   A → Bac|bSC|a
   B → aSB|bBC
   C → SBc|aBC|ac

Answer:

S → bC
C → ac

Eliminating λ-rules

- Sometimes it is desirable to eliminate λ-rules from a grammar G.
- This cannot be done if λ is in L(G),
- But it's always possible to eliminate λ-rules from a CFG and get a grammar that generates L(G) - {λ}.

Nullable symbols

- Eliminating λ-rules is like eliminating useless symbols.
- We first define a nullable symbol A to be one such that A =*> λ.
- Then for every rule that contains a nullable symbol on the RHS, we add a version of the rule that doesn't contain this symbol.
- Finally we remove all λ-productions from the resulting grammar.

Nullability

- Note that λ is in L(G) iff S is nullable.
- In this case a CFG with $S \rightarrow \lambda$ as its only λ-rule can be obtained by removing all other λ-rules and then adding this rule.
- Otherwise, removing λ-rules gives a CFG that generates L(G) = L(G) - {λ}
- By a simple induction, A is nullable iff
  - G has a rule $A \rightarrow \lambda$, or
  - G has a rule $A \rightarrow \alpha$, where every symbol in α is nullable

Example: removing nullable symbols

- Suppose G has the 9 rules

  S → ABC | CDB, A → λ|aA, B → b|bB,

  C → λ|cC, D → AC
- Then A and C are nullable, as is D.
- Optionally deleting nullable symbols adds:
  - S → BC | AB | B, S → DB | CB | B,
  - A → a, C → c, D → A | C
- Removing A → λ and C → λ (and not adding D → λ) gives a CFG with 16 distinct rules

Exercise on Eliminating λ-Productions

- Remove Empty Productions from the following production rule of a CFG

S → AB
A → aAA | λ
B → bBB | λ

Answer:

S → AB | A | B
A → aAA | aA |a
B → bBB | bB | b

Observations on the previous example

$S \rightarrow ABC \mid CDB,\ A \rightarrow \lambda \mid aA,\ B \rightarrow b \mid bB,$

$C \rightarrow \lambda \mid cC,\ D \rightarrow AC$

- Note that if the rule $S \rightarrow ABC$ had been replaced by $S \rightarrow AC$, then λ would be in L(G).
- We'd then have to allow the rule $S \rightarrow \lambda$ into the simplified grammar to generate all of L(G).
- Our algorithm for eliminating λ-rules has the annoying property that it introduces rules

Unit productions

- Productions of the form A → B are called unit productions.
- Unit productions can be eliminated from a CFG
- In all cases where A =*> B => α, a rule must be added of the form A → α

Eliminating unit productions -- example

- Consider the familiar grammar with rules
  - E → E+T | T, T → T*F | F, F → x | (E)
- Here we have that
  - E =*> T and T =*> F (at level 0), and
  - E =*> F (at level 1)
- Eliminating unit productions gives new rules
  - E → E+T | T*F | x | (E)
  - T → T*F | x | (E)
  - F → x | (E)

Exercise on Eliminating unit Productions

- Remove Unit Productions from the following production rule of a CFG

S → Aa|A
A → a|bc|B
B → A|bb

Answer:

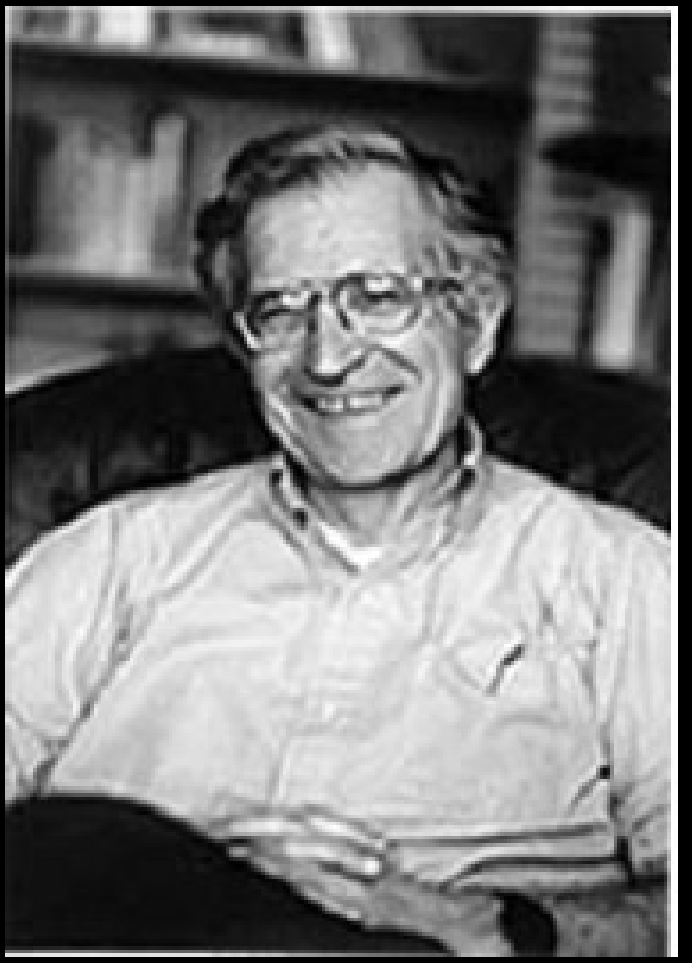
S → Aa |a|bc|bb
A → a| bc |bb
B → a| bc |bb

Order of steps when simplifying

- To eliminate useless symbols, λ-productions, and unit productions safely from a CFG, we need to apply the simplification algorithms in an appropriate order.
- A safe order is:
  - λ-productions
  - unit productions
  - useless symbols
    - nongenerating symbols
    - unreachable symbols

Chomsky normal form

Noam Chomsky
– The Grammar Guy
– 1928 –
– b. Philadelphia, PA
– PhD – UPenn (1955)
• Linguistics
– Prof at MIT
(Linguistics)
(1955 - present)



Chomsky normal form

- One additional way to simplify a CFG is to simplify each RHS
- A CFG is in Chomsky normal form (CNF) iff each production has a RHS that consists of
  - a single terminal symbol, or
  - two variable symbols
- For any CFG G, with λ not in L(G), there is an equivalent grammar G_1 in CNF.
- Must begin the transformation *after simplifying* the grammar (removing λ, all unit productions, & useless variables)

Converting to Chomsky normal form

- A CFG that doesn't generate λ may be converted to CNF by first eliminating all λ-moves and unit productions.
- This will give a grammar where each RHS of length less than 2 consists of a lone terminal.
- Any RHS of length $k > 2$ may be broken up by introducing $k-2$ new variables.
- For any terminal a that remains on a RHS, we add a new variable and new rule $C_a \rightarrow a$.

Converting to CNF: an example

- For example, the rule S → AbCD in a CFG G can be replaced by
  - S → AX, X → bY, Y → CD
- Here we don't change L(G)
- After the remaining steps, the new rules would be
  - S → AX, X → C_bY, Y → CD, C_b → b
- Again we don't change L(G)

A more complete example

- Consider the grammar with rules
  - $E \rightarrow E + T \mid T * F \mid x, \quad T \rightarrow T * F \mid x, \quad F \rightarrow x$
- The last rule for each symbol is legal in CNF.
- We may replace
  - $E \rightarrow E + T$ by $E \rightarrow EX,\ X \rightarrow C_{+}T,\ C_{+} \rightarrow +$
  - $E \rightarrow T * F$ by $E \rightarrow TY,\ Y \rightarrow C_{*}F,\ C_{*} \rightarrow *$
  - $T \rightarrow T * F$ by $T \rightarrow TZ,\ Z \rightarrow C_{*}F,\ C_{*} \rightarrow *$

The resulting grammar

- The resulting CFG is in CNF, with rules
  - $E \rightarrow EX \mid TY \mid x$
  - $T \rightarrow TZ \mid x$
  - $F \rightarrow x$
  - $X \rightarrow C_{+}T$
  - $Y \rightarrow C_{*}F$
  - $Z \rightarrow C_{*}F$ (or Z could be replaced by Y)
  - $C_{+} \rightarrow +$
  - $C_{*} \rightarrow *$

Exercise

- Convert the CFGs into Chomsky Normal Forms

1) $S \rightarrow \varepsilon \mid ADDA$
$A \rightarrow a$
$C \rightarrow c$
$D \rightarrow bCb$

2) $S \rightarrow bA \mid aB$
$A \rightarrow bAA \mid aS \mid a$
$B \rightarrow aBB \mid bS \mid b$

3) $S \rightarrow abAB$
$A \rightarrow bAB \mid \lambda$
$B \rightarrow BAa \mid A \mid \lambda$

4) $S \rightarrow aS \mid bS \mid B$
$B \rightarrow bb \mid C \mid \lambda$
$C \rightarrow cC \mid \lambda$

Greibach Normal Form

- A CFG G is in Greibach Normal Form if
  - every production is of the form $A \to a\alpha$, where $\alpha \in V^*$ and $a \in \Sigma$ (α may be λ) and
  - $S \to \lambda$ is in G if $\lambda \in L(G)$.

- When $\lambda \in L(G)$, we assume that S does not appear on the R.H.S. of any production.
- For example, G given by

  $S \to aAB \mid A, \quad A \to bC, \quad B \to b, \quad C \to c$

  is in GNF.

Greibach Normal Form

- For converting any CFG G into GNF, we use this two technical lemmas:

Lemma 1:

- Let G = (V, T, P, S) be a CFG, and $A \rightarrow B\gamma$ be an *A*-production in *P*. Let the *B*-productions be $B \rightarrow \beta 1 \mid \beta 2 \mid \ldots \mid \beta s$. Define $P1 = (P - \{A \rightarrow B\gamma\}) \cup \{A \rightarrow \beta_i \gamma \mid 1 \leq I \leq s\}$. Then, G1 = (V, T, P1, S) is a CFG equivalent to G.

Lemma 2:

- Let G = (V, T, P, S) be a CFG, and let the set of *A*-productions be

  $A \rightarrow A\alpha_1 \mid \ldots \mid A\alpha_r \mid \beta 1 \mid \ldots \mid \beta s$ (β_i's do not start with A).

- Let *Z* be a new variable and G1 = (V∪{Z}, T, *P1*, S), where *P1* is defined as follows:

(i) The set of *A*-productions in *P1* are $A \rightarrow \beta 1 \mid \ldots \mid \beta s \mid \beta 1 Z \mid \ldots \mid \beta s Z$

(ii) The set of *Z*-productions in P1 are $Z \rightarrow \alpha_1 \mid \ldots \mid \alpha_r \mid \alpha_1 Z \mid \ldots \mid \alpha_r Z$

(iii) The productions for the other variables are as in *P*. Then G1 is a CFG and equivalent to G.

Conversion to GNF

Step 1:

- Eliminate null productions and then construct a grammar G in Chomsky normal form generating ***L***.
- Rename the variables as A1, A2, . . . , An with S = A1.
- We write G as ({A1, A2, . . . , An}, Σ, P, A1).

Step 2:

- To get the productions in the form $A_i \rightarrow a\gamma$ or $A_i \rightarrow A_j\gamma$, where $j > i$, convert the A_i-*productions* (i = 1, 2, . . . , n - 1) to the form $A_i \rightarrow A_i\gamma$ such that j > I, using lemma 2.

Step 3:

- Convert A_n-productions to the form $A_n \rightarrow a\gamma$. Here, the productions of the form $A_n \rightarrow A_n\gamma$ are eliminated using Lemma 2.
- The resulting A_n-productions are of the form $A_n \rightarrow a\gamma$.

Conversion to GNF

Step 4:

- Modify the A_i-productions to the form $A_i \rightarrow a\gamma$ for i = 1, 2, . . . , n – 1.
- At the end of step 3, the A_n-productions are of the form $A_n \rightarrow a\gamma$.
- The A_{n-1}-productions are of the form $A_n \rightarrow a\gamma$ ' or $A_{n-1} \rightarrow A_n\gamma$.
- By applying Lemma 1, we eliminate productions of the form $A_{n-1} \rightarrow A_n\gamma$.
- The resulting A_{n-1}-productions are in the required form. We repeat the construction by considering A_{n-2}, A_{n-3}, . . . , A^1.

Conversion to GNF

Step 5:

- Modify Z_i-productions. Every time we apply Lemma 2, we get a new variable. (We take it as Z_i when we apply the Lemma 2 for A_i-Productions.)
- The Z_i-productlons are of the form $Z_i \rightarrow \alpha Z_i$ or $Z_i \rightarrow \alpha$ (where α is obtained from $A_i \rightarrow A_i \alpha$), and hence of the form $Z_i \rightarrow a\gamma$ or $Z_i \rightarrow A_k \gamma$ for some k.
- At the end of step 4, the R.H.S. of any A_k-production starts with a terminal.
- So we can apply Lemma 1 to eliminate $Z_i \rightarrow A_k \gamma$.
- Thus at the end of step 5, we get an equivalent grammar G1 in GNF.

Conversion to GNF

Example: Construct a grammar in Greibach normal form equivalent to the grammar
$S \rightarrow AA \mid a,\ \ A \rightarrow SS \mid b$
Solution:
Step1:

- Eliminate null productions and convert *G* to CNF.
- The given grammar is in CNF.
- S and A are renamed as A1 and A2, respectively.
- So the productions are A1 → A2A2 | a, and A2 → A1A1 | b.
- As the given grammar has no null productions and is in CNF we need not carry out step 1.
- So we proceed to step 2.

Conversion to GNF

Example: Cont....

Step2:

- Convert the A_i-productions (i = 1, 2, . . . , n - 1)

$A_i \rightarrow a\gamma$ or $A_i \rightarrow A_j\gamma$ such that $j > i$.

(i) *A1*-productions are in the required form.

- They are $A1 \rightarrow A2A2 \mid a$.

(ii) $A2 \rightarrow b$ is in the required form.

- We apply ***Lemma 1*** to $A2 \rightarrow A1A1$.
- The resulting productions are $A2 \rightarrow A2A2A1, A2 \rightarrow aA1$.
- Thus the *A2*-productions are $A2 \rightarrow A2A2A1$, $A2 \rightarrow aA1$, $A2 \rightarrow b$

Conversion to GNF

Example: Cont....

Step3:

- Convert A_n-productions of form $A_n \rightarrow A_n\gamma$ to the form $A_n \rightarrow a\gamma$, by applying lemma 2.
- We have to apply Lemma 2 to *A2*-productions as we have $A2 \rightarrow A2A2A1$, and let *Z2* to be the new variable.
- The resulting productions are:

$A2 \rightarrow aA1$ $\quad$ $A2 \rightarrow b$

$A2 \rightarrow aA1Z2$ $\quad$ $A2 \rightarrow bZ2$

$Z2 \rightarrow A2A1$ $\quad$ $Z2 \rightarrow A2A1Z2$

Conversion to GNF

Example: Cont....

Step4:

- Modify the A_i-productions to the form $A_i \rightarrow a\gamma$ *for* $i = 1,2, \ldots, n-1$ by applying lemma 1.

(i) The *A2*-productions are $A2 \rightarrow aA1 \mid b \mid aA1Z2 \mid bZ2$.

(ii) Among the *A*1-productions we retain $A1 \rightarrow a$ and eliminate $A1 \rightarrow A2A2$ using Lemma 1.

- The resulting productions are $A1 \rightarrow aA1A2 \mid bA2$, $A1 \rightarrow aA1Z2A2 \mid bZ2A2$
- The set of all (modified) A1-productions is

$A1 \rightarrow a \mid aA1A2 \mid bA2 \mid aA1Z2A2 \mid bZ2A2$

Conversion to GNF

Example: Cont....

Step5:

- Modify Z_i-productions, which are of the form $Z_i \rightarrow a\gamma$ *or* $Z_i \rightarrow A_k\gamma$ for some k.
- The *Z2*-productions to be modified are:

$Z2 \rightarrow A2A1, \quad Z2 \rightarrow A2A1Z2$

- We apply Lemma 1 and get

$Z2 \rightarrow aA1A1 \mid bA1 \mid aA1Z2A1 \mid bZ2A1$

$Z2 \rightarrow aA1A1Z2 \mid bA1Z2 \mid aA1Z2A1Z2 \mid bZ2A1Z2$

Conversion to GNF

Example: Cont....

- Hence the equivalent grammar is

G' = ({A1, A2, Z2}, {a, b}, P1, A1)

where *P1* consists of

A1 → *a* | *aA1A2* | *bA2* | *aA1Z2A2* | *bZ2A2*
A2 → *aA1* | *b* | *aA1Z2* | *bZ2*
Z2 → *aA1A1* | *bA1* | *aA1Z2A1* | *bZ2A1*
Z2 → *aA1A1Z2* | *bA1Z2* | *aA1Z2A1Z2*| *bZ2A1Z2*

- **G'** is the required equivalent grammar in **GNF**.