Context-Free Grammars

Tanjila Alam Sathi

CSE Department,IUT

"Too many Computers,

In too many Countries

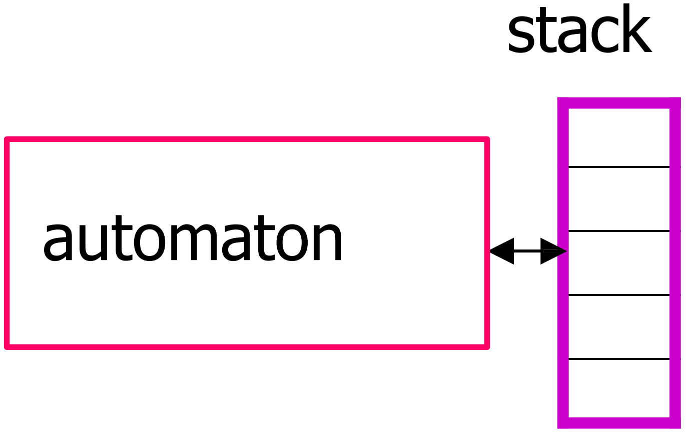
Recognize the same Language,

The language of CFG"

Context-Free Languages

Context-Free Grammars

Pushdown Automata



Introduction

- CFG have played a central role in compiler technology since the 1960's.
- They turned the implementation of parsers
- Parsers functions: discover the structure of a program.
- **Other Uses:** document-type (DTD), XML(extensible language) definition markup

CFG: Informal Example

- Palindromes
- A palindrome is a string that reads the same forward & backward, such as *otto*, *madamimadam* ("Madam, I am Adam",).
- Let's consider describing only the palindromes with alphabet {0,1}. EX: 0110,11011 etc.
- Basis: ε, 0 and 1 are palindromes

Induction: if w is a palindromes, so are 0w0 and 1w1. No string is a palindrome of 0's & 1's, unless it follows from this basis & induction rule

CFG for Palindromes

1. $P \rightarrow \epsilon$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

Only for binary strings.

Informal Comments

- A *context-free grammar* is a notation for describing languages.
- It is more powerful than FA/RE's, but still cannot define all possible languages.
- Useful for nested structures, e.g., parentheses in programming languages.

Informal Comments – (2)

- Basic idea is to use "variables" to stand for sets of strings (i.e., languages).
- These variables are defined recursively, in terms of one another.
- Recursive rules ("productions") involve only concatenation.
- Alternative rules for a variable allow union.

Example: CFG for $\{ 0^n1^n \mid n \geq 1\}$

- Productions:

  S -> 01

  S -> 0S1

- Basis: 01 is in the language.
- Induction: if w is in the language, then so is 0w1.

CFG Formalism

- *Terminals* = symbols of the alphabet of the language being defined.
- *Variables* = *nonterminals* = a finite set of other symbols, each of which represents a language.
- *Start symbol* = the variable whose language is the one being defined.

Productions/Rules

- A *production* has the form variable -> string of variables and terminals.
- Convention:
  - A, B, C,… are variables.
  - a, b, c,… are terminals.
  - …, X, Y, Z are either terminals or variables.
  - …, w, x, y, z are strings of terminals only.
  - α, β, γ,… are strings of terminals and/or variables.

Productions/Rules

- Each productions consists of:

a.the head of the production.

b.the production symbol →

c.The body of the production, a string of zero or more terminals and variables.

Example: Formal CFG

- Here is a formal CFG for $\{ 0^n 1^n \mid n \geq 1 \}$.
- Terminals = {0, 1}.
- Variables = {S}.
- Start symbol = S.
- Productions =

  S -> 01

  S -> 0S1

Formal Definition of CFG

- The 04 components of CFG G can be represent as follows

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

Variables

terminals

productions

Start variable

Example of Context-Free Grammar

$$\boxed{S \to aSb \mid \lambda}$$

Productions

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

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

$$V = \{S\}$$

variables

$$T = \{a, b\}$$

terminals

start variable

A Context-free Grammar for Palindromes

- The grammar G_{pal} for the palindrome is represented by..

$$G_{pal} = (\{P\},\{0,1\},A,P)$$

where A represents the set of 05 productions:

- $P \rightarrow \epsilon$
- $P \rightarrow 0$
- $P \rightarrow 1$
- $P \rightarrow 0P0$
- $P \rightarrow 1P1$

Example of CFG

- A CFG for simple expressions with **'+'** & **'*'**.
- It allows only the letters 'a' & 'b' & the digits '0' & '1'.
- Every identifiers must begin with a & b which may be followed by any other string in {a,b,0,1}*

Productions:

1. E → I	6. I → b
2. E → E+E	7. I → Ia
3. E → E*E	8. I → Ib
4. E → (E)	9. I → I0
5. I → a	10 I → I1

- G=({E,I},T,P,E)
- T={0,1,a,b,+,*,(,)}

Derivation using Grammar

- We apply the productions of a CFG to infer the strings. There are two approaches:
  - Recursive Inferences
  - Derivation

  - **Recursive Inferences**: In this approach rules use from **body to head**.
  - **Derivation:** In this approach rules used from head to body.

Inferring string using grammar

	String Inferred	For language of	Production Used	String (s) Used
(i)	a	I	5	---
(ii)	b	I	6	---
(iii)	b0	I	9	(ii)
(iv)	b00	I	9	(iii)
(v)	a	E	1	(i)
(Vi)	b00	E	1	(iv)
(Vii)	a+b00	E	2	(v), (vi)
(Viii)	(a+b00)	E	4	(vii)
(ix)	a*(a+b00)	E	3	(v), (viii)

Productions:

1. E → I
2. E → E+E
3. E → E*E
4. E → (E)
5. I → a
6. I → b
7. I → Ia
8. I → Ib
9. I → I0
10. I → I1

Derivation using grammar

- **(ab+ab0)**

1. E→(E)--------------4
2. E→(E+E)-----------2
3. E→(I+E)------------1
4. E→(Ib+E)----------8
5. E→(ab+E)---------5
6. E→(ab+I)-----------1
7. E→(ab+I0)-----------9
8. E→(ab+Ib0)---------8
9. E→(ab+ab0)--------5

Productions:

1. E →I
2. E →E+E
3. E →E*E
4. E →(E)
5. I →a
6. I →b
7. I →Ia
8. I →Ib
9. I →I0

10 I →I1

An informal example

Word categories: Traditional parts of speech

Noun	Names of things	boy, cat, truth
Verb	Action or state	become, hit
Pronoun	Used for noun	I, you, we
Adverb	Modifies V, Adj, Adv	sadly, very
Adjective	Modifies noun	happy, clever
Conjunction	Joins things	and, but, while
Preposition	Relation of N	to, from, into
Interjection	An outcry	ouch, oh, alas, psst

An example of CFG

$G = \langle T, N, S, R \rangle$

$T = \{that,\ this,\ a,\ the,\ man,\ book,\ flight,\ meal,\ include,\ read,\ does\}$

$N = \{\text{S, NP, NOM, VP, Det, Noun, Verb, Aux}\}$

$S = S$

$R = \{$

$\text{S} \rightarrow \text{NP VP}$
$\text{S} \rightarrow \text{Aux NP VP}$
$\text{S} \rightarrow \text{VP}$
$\text{NP} \rightarrow \text{Det NOM}$
$\text{NOM} \rightarrow \text{Noun}$
$\text{NOM} \rightarrow \text{Noun NOM}$
$\text{VP} \rightarrow \text{Verb}$
$\text{VP} \rightarrow \text{Verb NP}$

$\text{Det} \rightarrow \textit{that} \mid \textit{this} \mid \textit{a} \mid \textit{the}$
$\text{Noun} \rightarrow \textit{book} \mid \textit{flight} \mid \textit{meal} \mid \textit{man}$
$\text{Verb} \rightarrow \textit{book} \mid \textit{include} \mid \textit{read}$
$\text{Aux} \rightarrow \textit{does}$

$\}$

An example of CFG

$S \longrightarrow NP\ VP$
$S \longrightarrow Aux\ NP\ VP$
$S \longrightarrow VP$
$NP \longrightarrow Det\ NOM$
$NOM \longrightarrow Noun$
$NOM \longrightarrow Noun\ NOM$
$VP \longrightarrow Verb$
$VP \longrightarrow Verb\ NP$

$Det \longrightarrow \textit{that} \mid \textit{this} \mid \textit{a} \mid \textit{the}$
$Noun \longrightarrow \textit{book} \mid \textit{flight} \mid \textit{meal} \mid \textit{man}$
$Verb \longrightarrow \textit{book} \mid \textit{include} \mid \textit{read}$
$Aux \longrightarrow \textit{does}$

$S \longrightarrow NP\ VP$
$\longrightarrow Det\ NOM\ VP$
$\longrightarrow \textit{The}\ NOM\ VP$
$\longrightarrow \textit{The}\ Noun\ VP$
$\longrightarrow \textit{The man}\ VP$
$\longrightarrow \textit{The man}\ Verb\ NP$
$\longrightarrow \textit{The man read}\ NP$
$\longrightarrow \textit{The man read}\ Det\ NOM$
$\longrightarrow \textit{The man read this}\ NOM$
$\longrightarrow \textit{The man read this}\ Noun$
$\longrightarrow \textit{The man read this book}$

Derivation Order

1. Left most derivation (LMD)
2. Right most derivation (RMD)

Consider the following example grammar with 05 productions:

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

$$1.\ S \to AB \qquad 2.\ A \to aaA \qquad 4.\ B \to Bb$$

$$3.\ A \to \lambda \qquad 5.\ B \to \lambda$$

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

$$1.\ S \to AB \qquad 2.\ A \to aaA \qquad 4.\ B \to Bb$$

$$3.\ A \to \lambda \qquad 5.\ B \to \lambda$$

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

$$
\begin{array}{lll}
1.\ S \to AB & 2.\ A \to aaA & 4.\ B \to Bb \\
 & 3.\ A \to \lambda & 5.\ B \to \lambda
\end{array}
$$

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

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

Another Example: LMD

- CFG: E → I | E+E | E*E| (E)
  I → a| B| Ia |Ib |I0 | I1
- a*(a+b00)
- E =>E*E
  lm=>I*E
  lm=>a*E
  lm=>a*(E)
  lm=>a*(E+E)
  lm=>a*(I+E)
  lm=>a * (a+E)
  lm=>a*(a+I)
  lm=>a*(a+I0)
  lm=>a*(a+I00)
  lm=>a*(a+b00)

Another Example: LMD

- CFG: E → I | E+E | E*E | (E)

  I → a | B | Ia | Ib | I0 | I1
- **a*(a+b00)**
- E =>**E*E**

  rm=>E*(E)

  rm=>E*(E+E)

  rm=>E*(E+I)

  rm=>E*(E+I0)

  rm=>E*(E+I00)

  rm=>E * (E+b00)

  rm=>E*(I+b00)

  rm=>E*(a+b00)

  rm=>I*(a+I00)

  rm=>a*(a+b00)

Derivation/Parse Tree

- Representation for derivations which shows clearly has the symbols of a terminal string are grouped into substrings.

- Graphical representation for a derivations

Properties of Parse Tree

- $G=(V,T,P,S)$.
- Conditions:

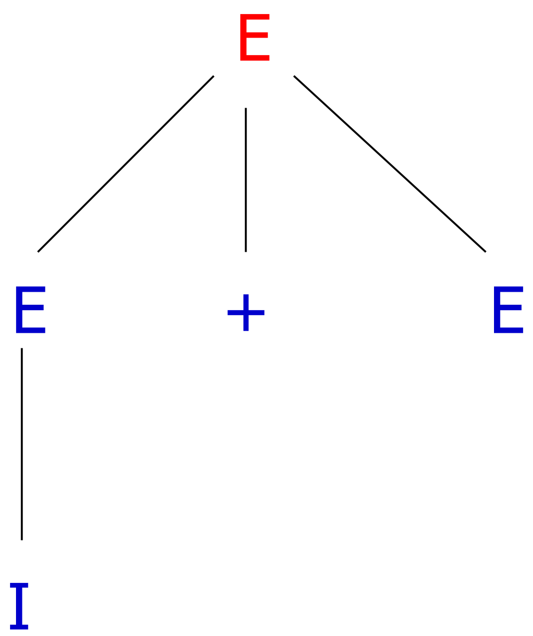
1. Each interior node is labeled by a variable V
2. Each leaf is labeled by either variable, a terminal or ϵ
3. If an interior node is labeled A, & its children are labeled

$$X_1, X_{2,}\ldots\ldots\ldots\ldots\ldots\ldots\ldots, X_k$$

respectively, from the left, then $A \rightarrow X_1X_2\ldots X_k$ is a production.

Example

- A parse tree showing the derivation of **$E \rightarrow I+E$**



$E \rightarrow I \mid E+E \mid E*E \mid (E)$
$I \rightarrow a \mid B \mid Ia \mid Ib \mid I0 \mid I1$

Example

Consider the same example grammar:

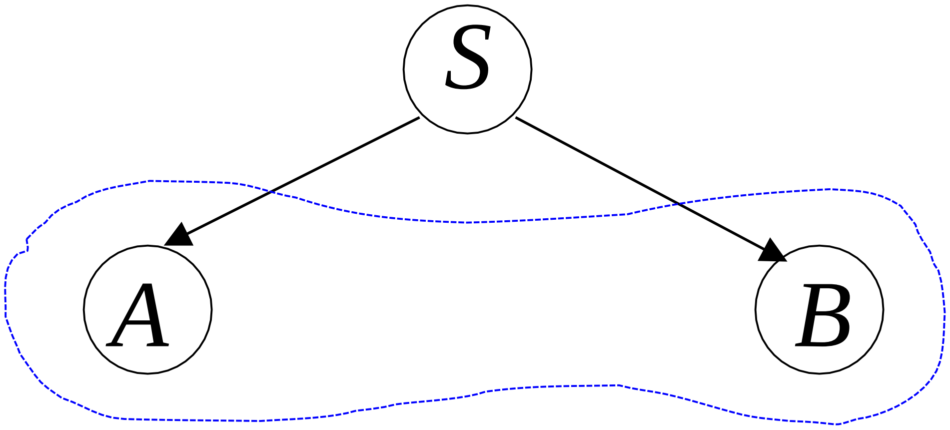
$$\boxed{S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda}$$

And a derivation of : aab

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

$$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$$

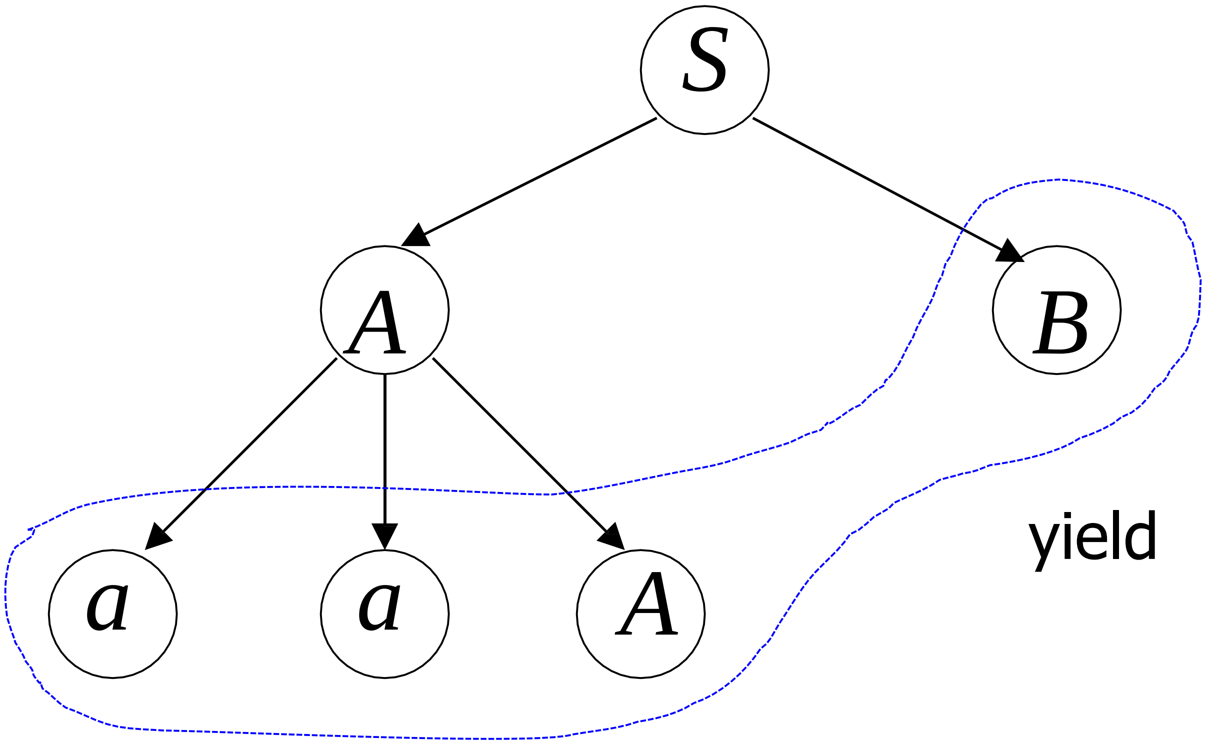
$$S \Rightarrow AB$$



yield AB

$$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$$

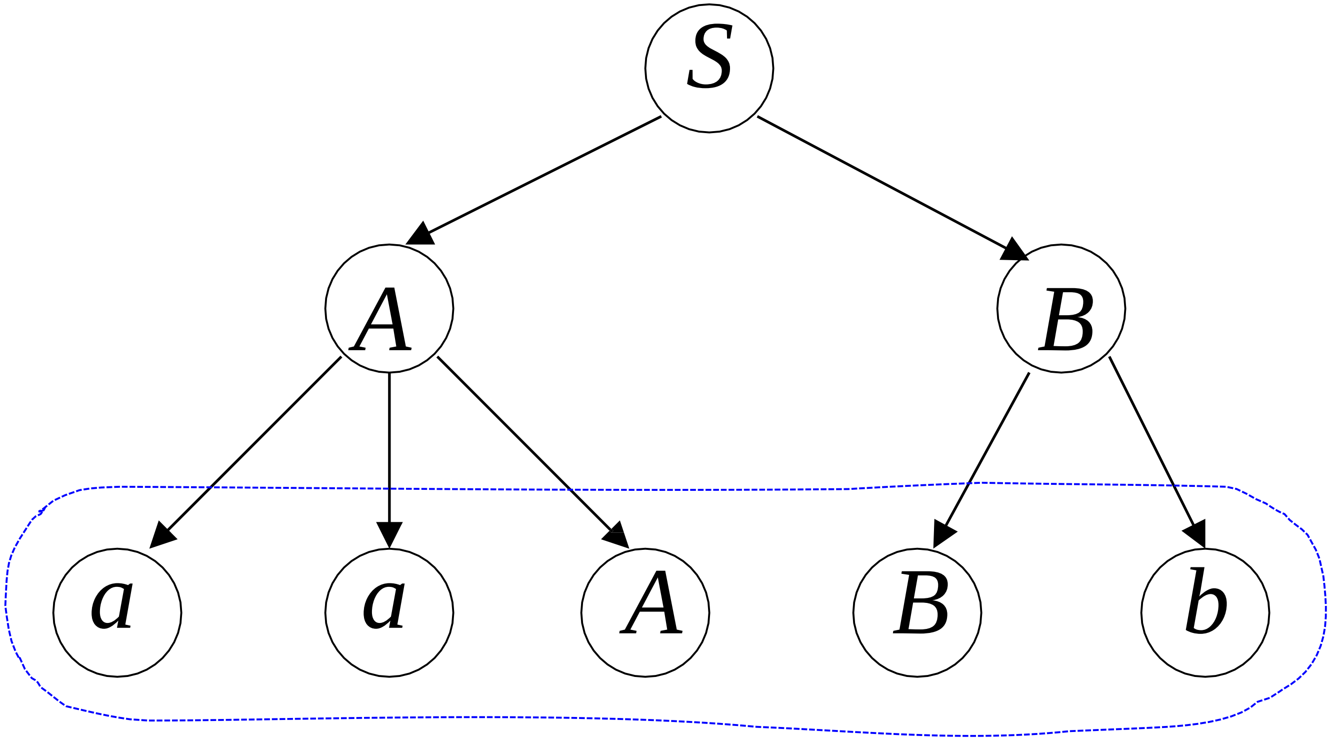
$$S \Rightarrow AB \Rightarrow aaAB$$



yield $aaAB$

$$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$$

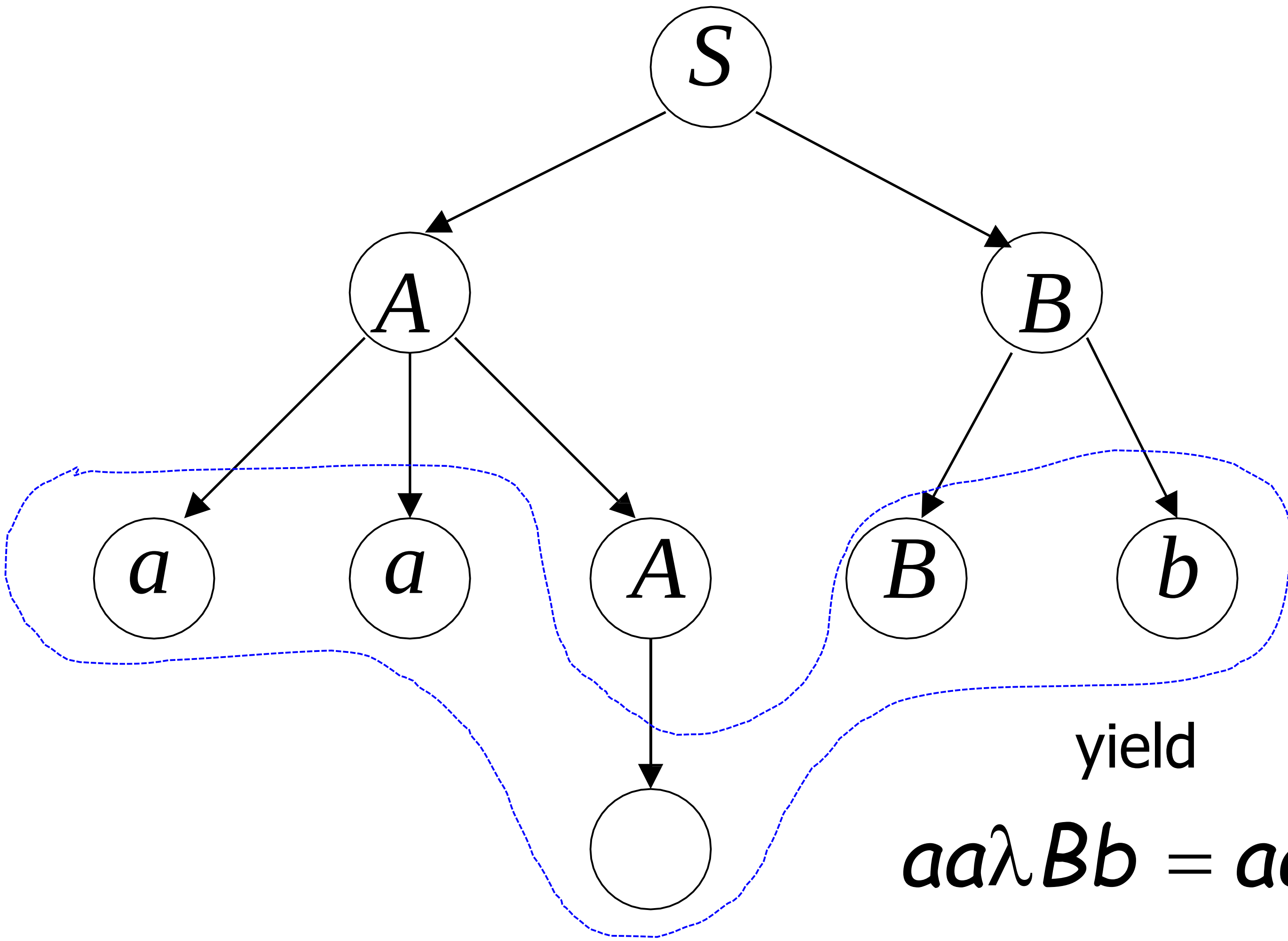
$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb$$



yield $aaABb$

$$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$$

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



yield

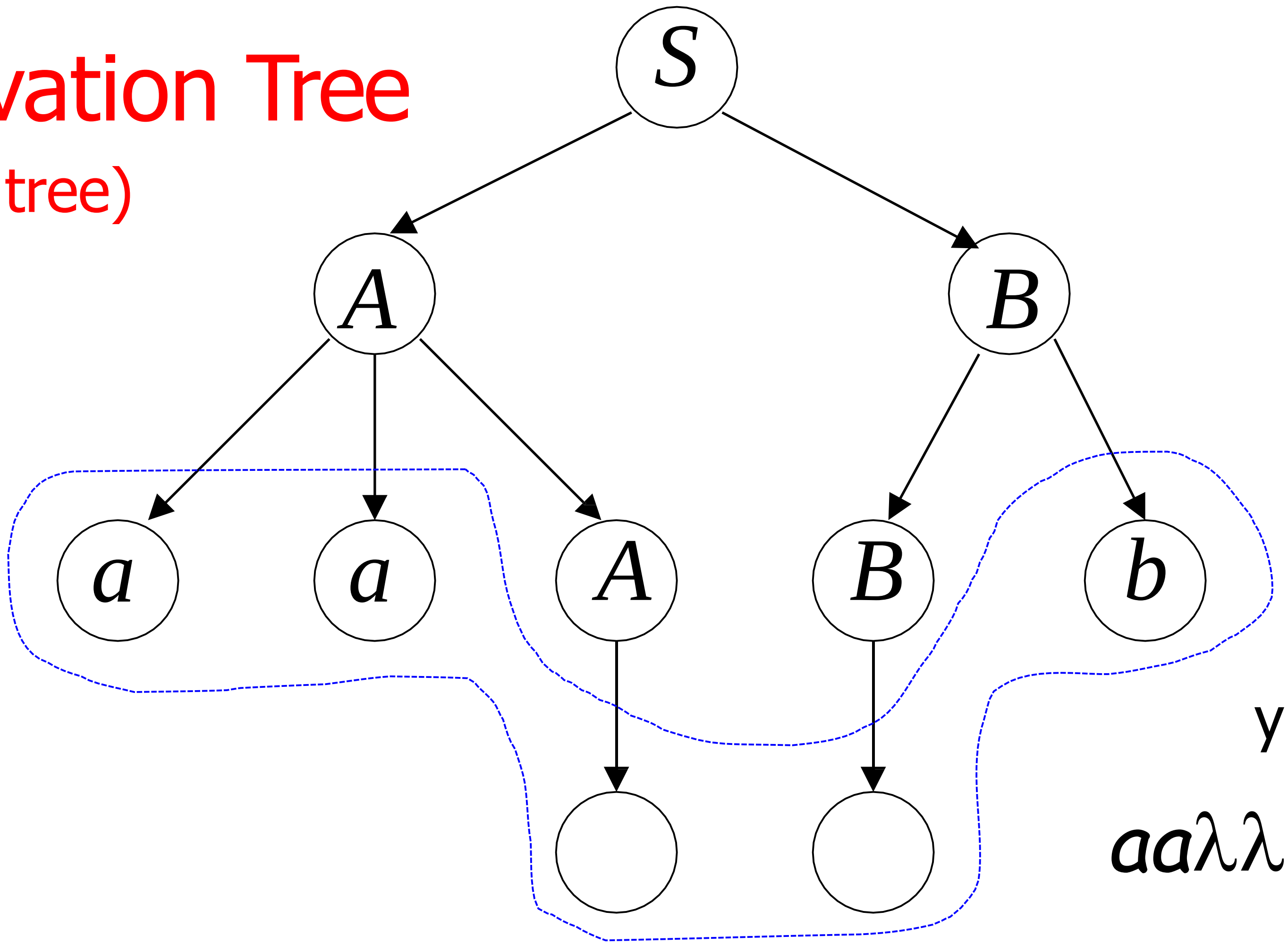
$$aa\lambda Bb = aaBb$$

$$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$$

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

Derivation Tree

(parse tree)



yield

$$aa\lambda\lambda b = aab$$

Sometimes, derivation order doesn't matter

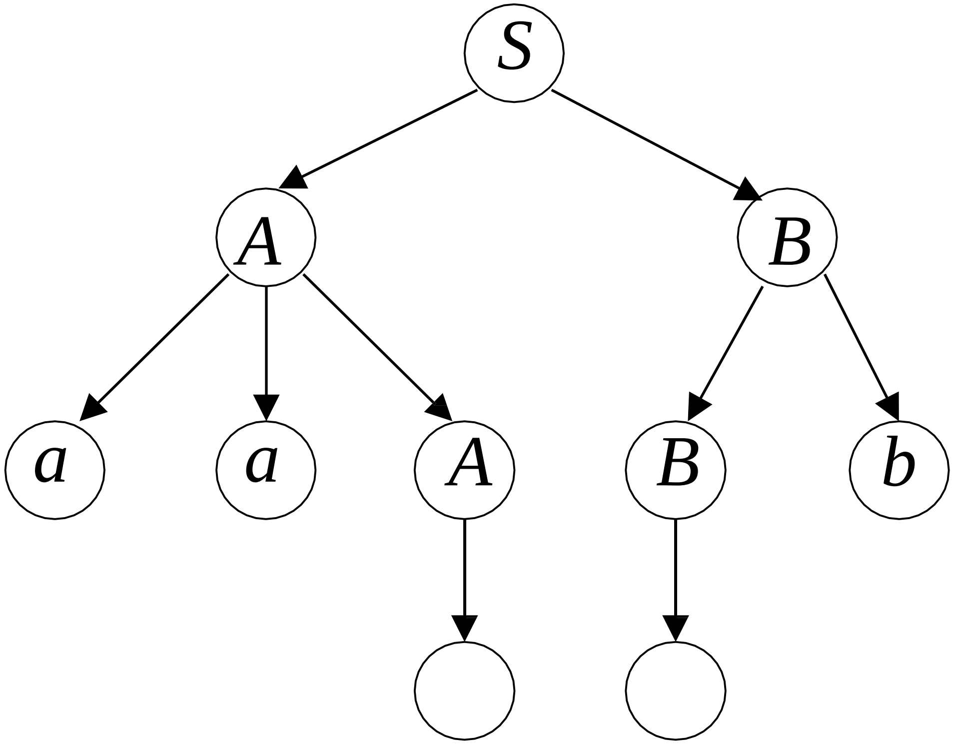
Leftmost derivation:

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

Rightmost derivation:

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

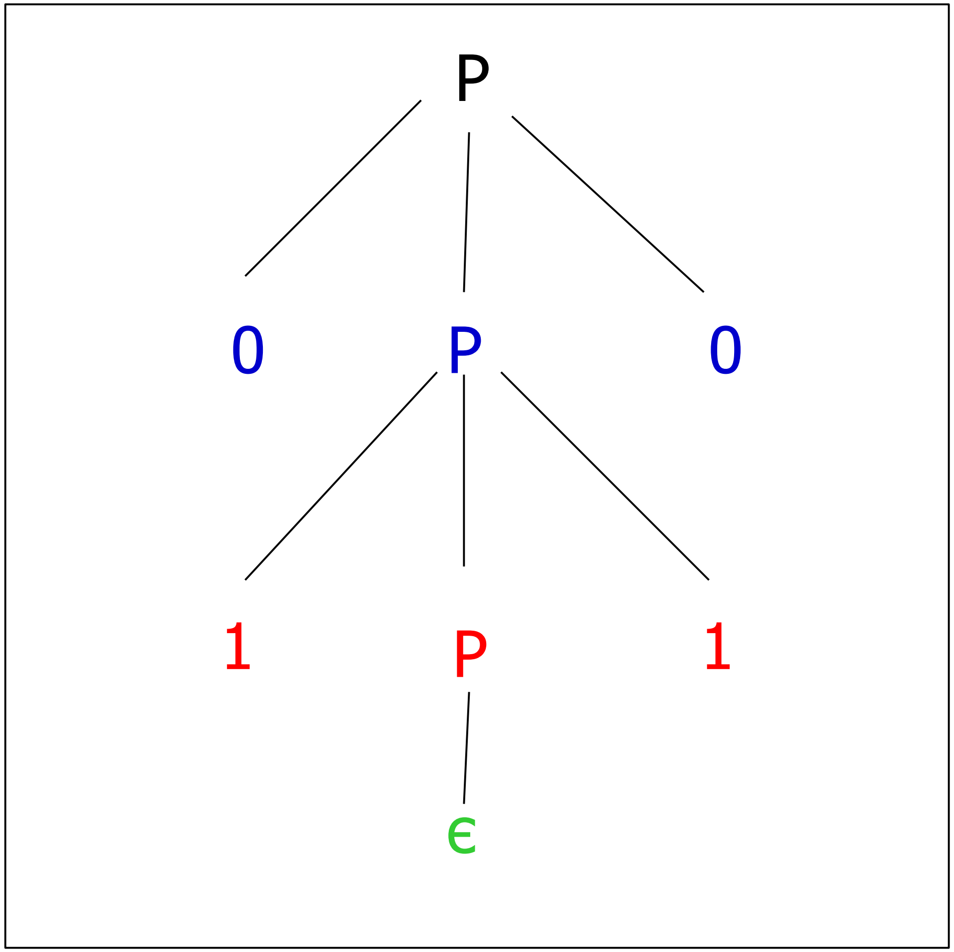
Give same derivation tree



Example

- A parse tree showing the derivation $P \overset{*}{\Rightarrow} 0110$.

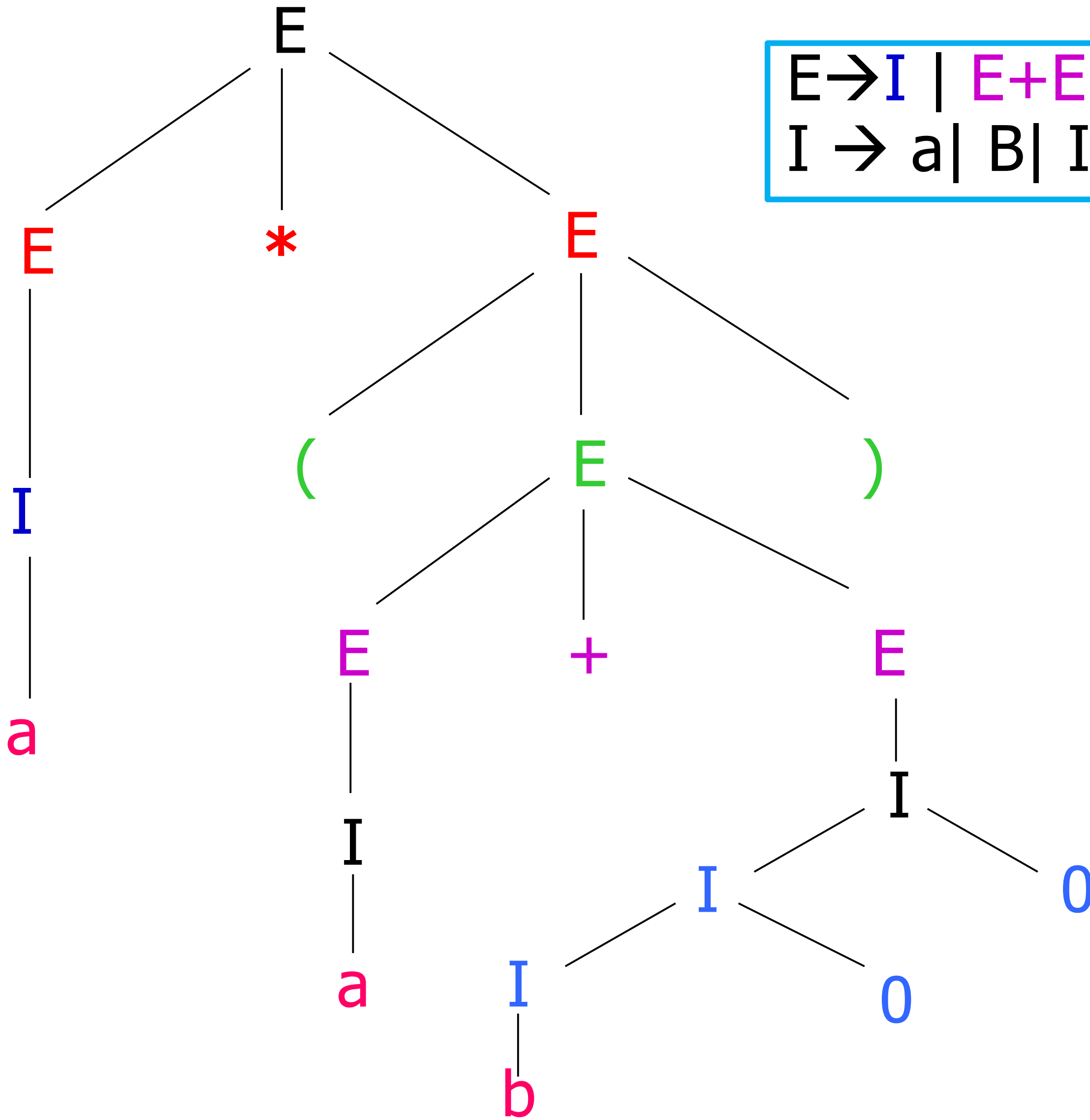
1. $P \to \epsilon$
2. $P \to 0$
3. $P \to 1$
4. $P \to 0P0$
5. $P \to 1P1$



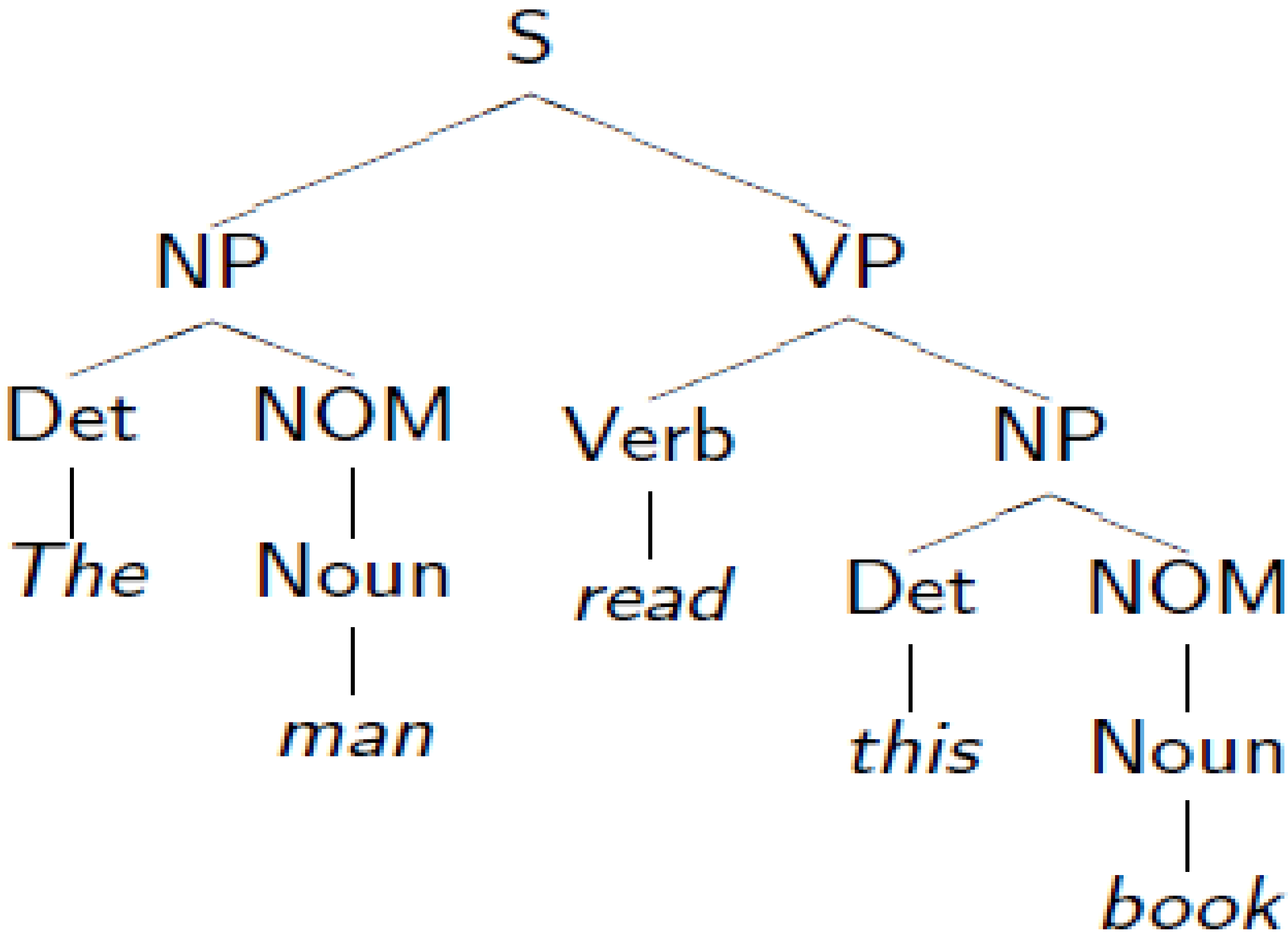
Parse tree showing a*(a+b00)

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

$I \rightarrow a \mid B \mid Ia \mid Ib \mid I0 \mid I1$



The man read this book



Sentential Forms

- Derivations from the start symbol produce strings that have a special role. We call these "**sentential forms**."

- That is, if G = (V,T,P,S) be a CFG, then any string α in (V U T)*

such that $S \Rightarrow^{*} \alpha$ is a sentential form.

- If $S \Rightarrow^{*}_{lm} \alpha$ then α is a left-sentential form,

- and if $S \Rightarrow^{*}_{rm} \alpha$ then α is a right-sentential form

Sentential Forms: Example

$E \rightarrow I \mid E+E \mid E*E \mid (E)$
$I \rightarrow a \mid B \mid Ia \mid Ib \mid I0 \mid I1$

- $E*(I+E)$ is a sentential form, since there is a derivation
- $E \Rightarrow E*E \Rightarrow E*(E) \Rightarrow E*(E+E) \Rightarrow E*(I+E)$
- This derivation is neither leftmost nor rightmost, since at the last step, the middle E is replaced.

Sentential Forms: Example

- $E \underset{lm}{\Rightarrow} E*E \underset{lm}{\Rightarrow} I*E \underset{lm}{\Rightarrow} a*E$ Left sentential form

- $E \underset{rm}{\Rightarrow} E*E \underset{rm}{\Rightarrow} E*(E) \underset{rm}{\Rightarrow} E*(E+E)$ Right Sentential Form

Ambiguity

- A grammar uniquely determines a structure for each string in its language. Not every grammar does provide unique structures.

- When a grammar fails to provide unique structure, it is known as ambiguous grammar.

- More than one derivation/parse tree.

Grammar for mathematical expressions

$$\boxed{E \rightarrow E+E \mid E*E \mid (E) \mid a}$$

Example strings:

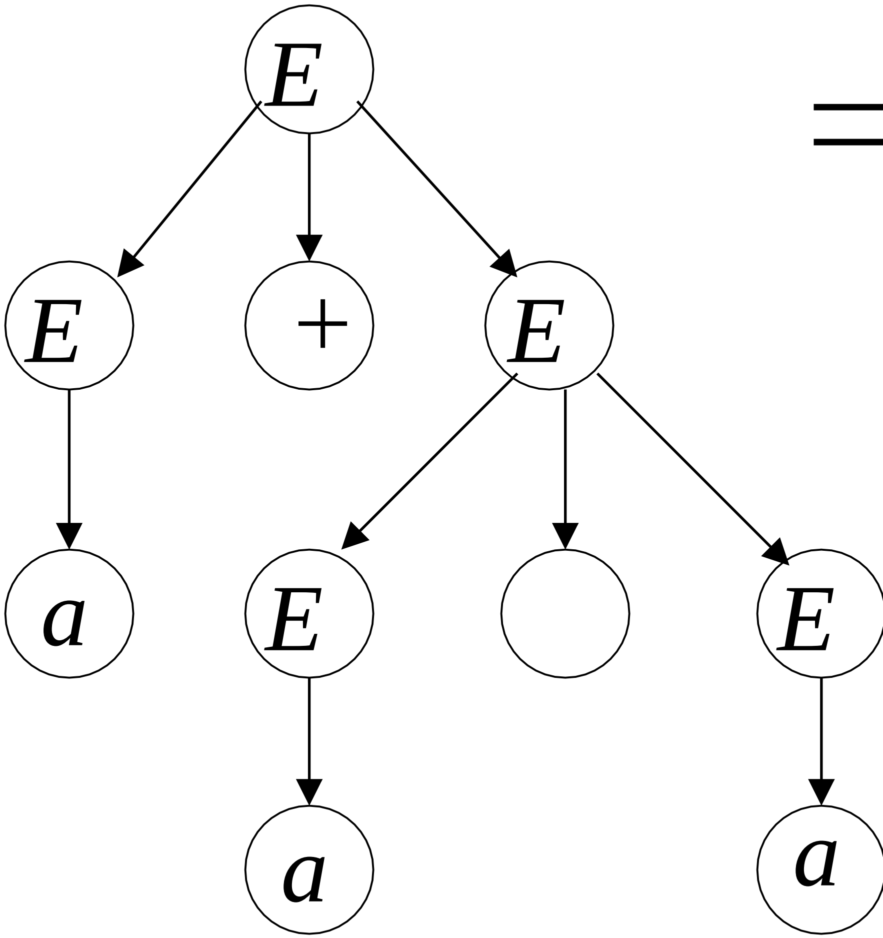
$$(a+a)*a+(a+a*(a+a))$$

Denotes any number

$$\boxed{E \to E+E \mid E*E \mid (E) \mid a}$$

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

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



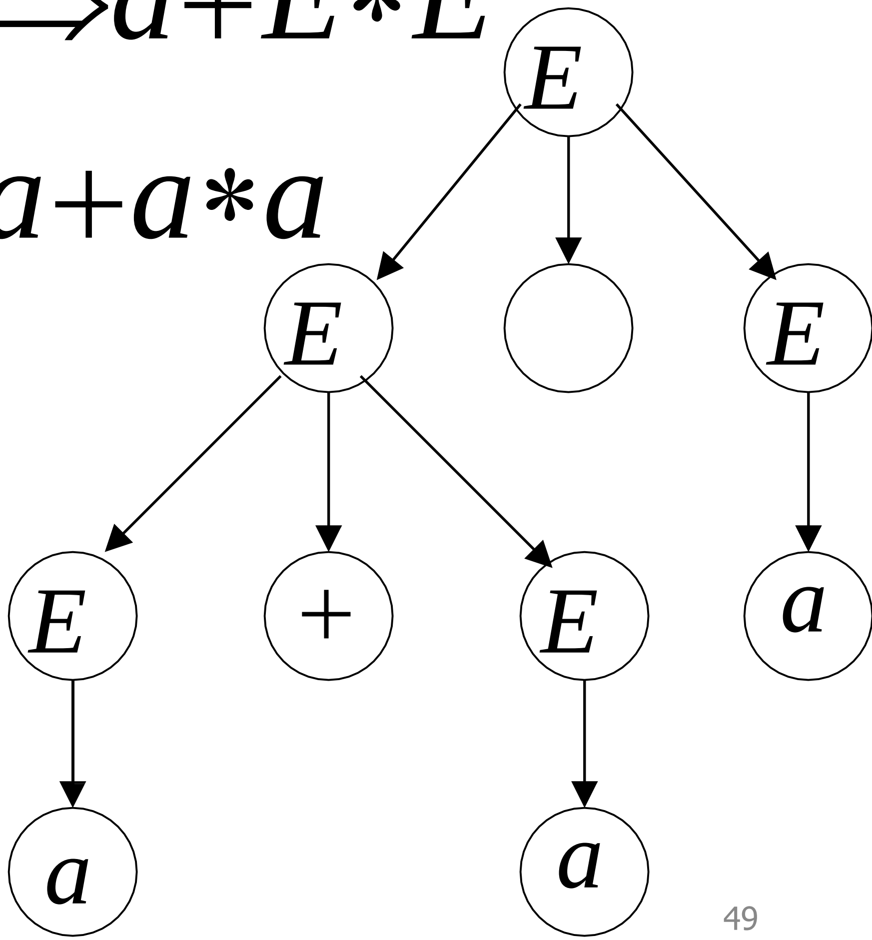
A leftmost derivation for

$$a + a * a$$

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

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

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



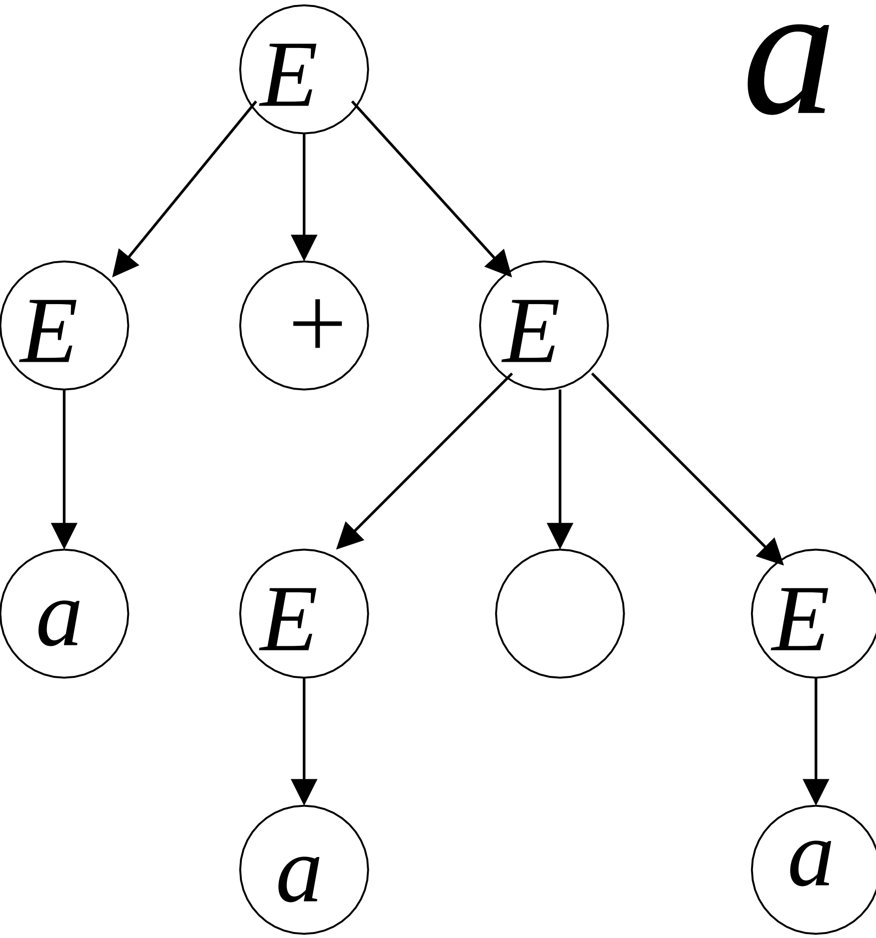
Another leftmost derivation for

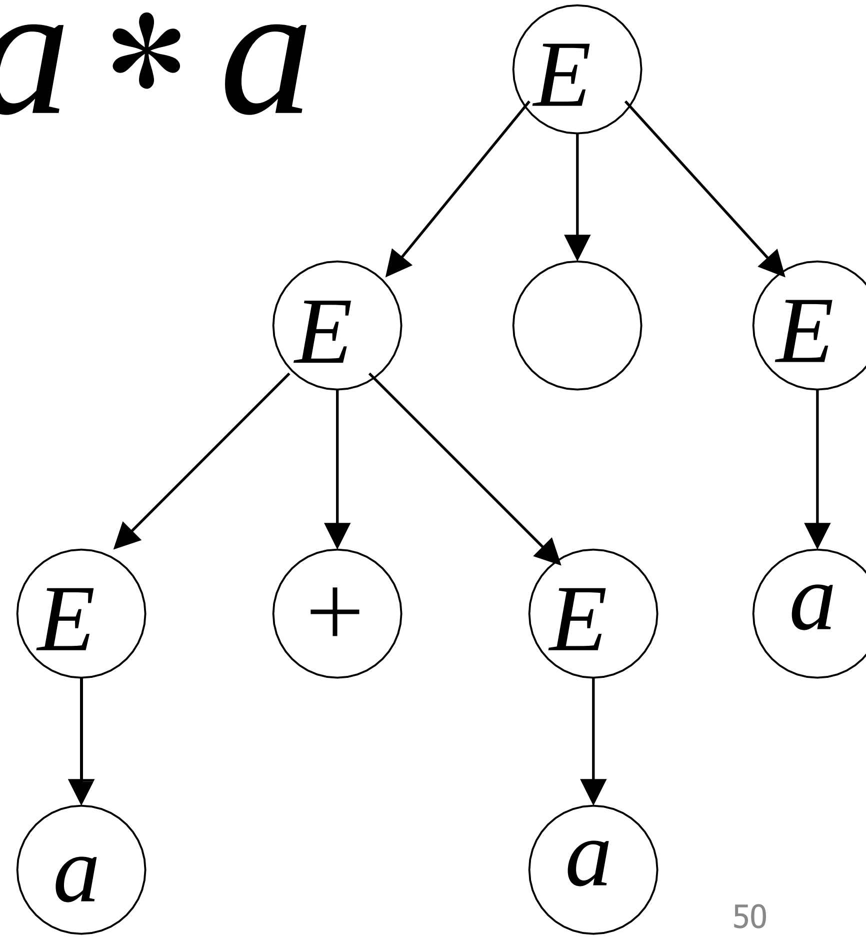
$$a+a*a$$

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

Two derivation trees for

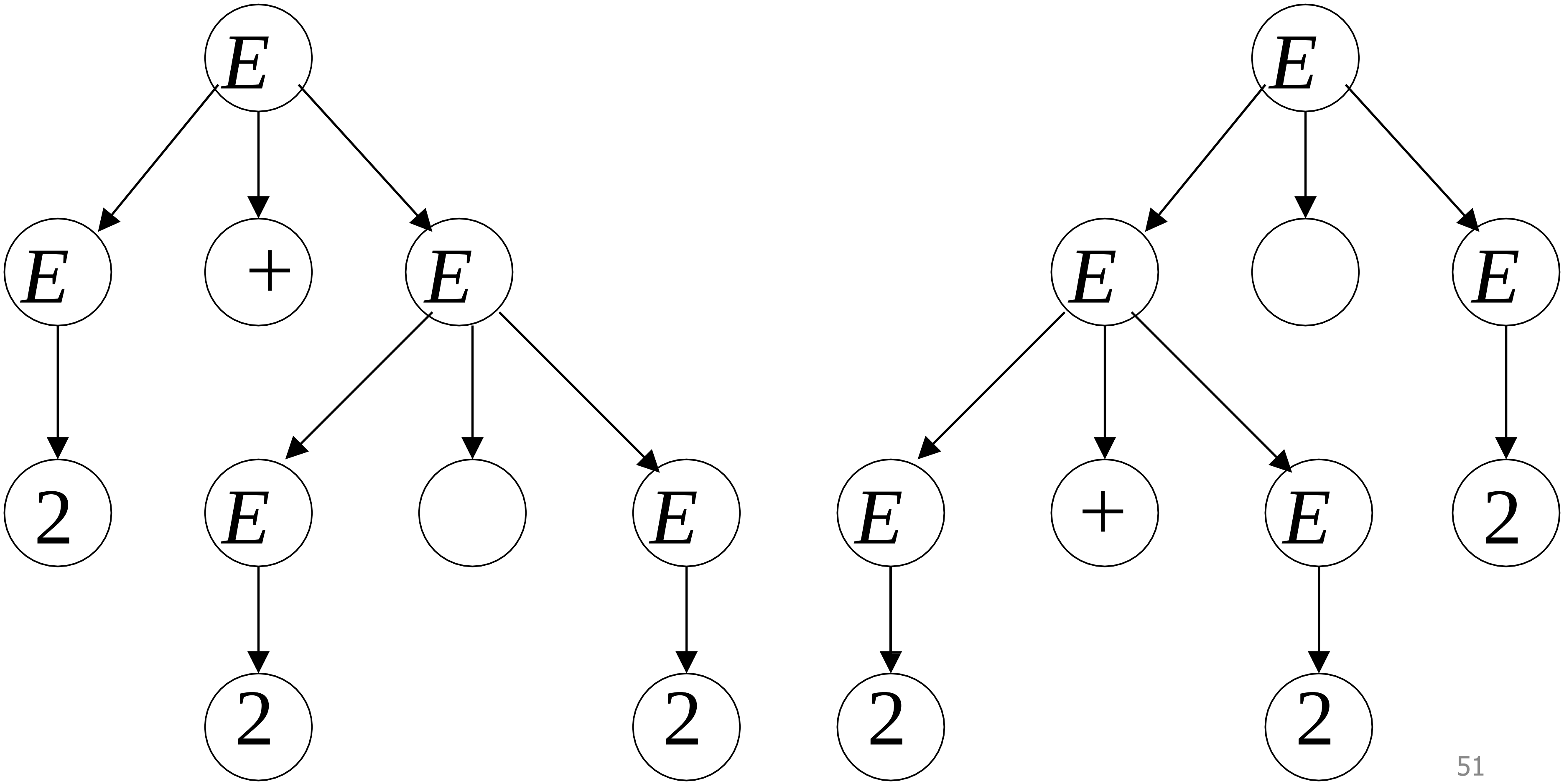
$$a + a * a$$





take $a = 2$

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



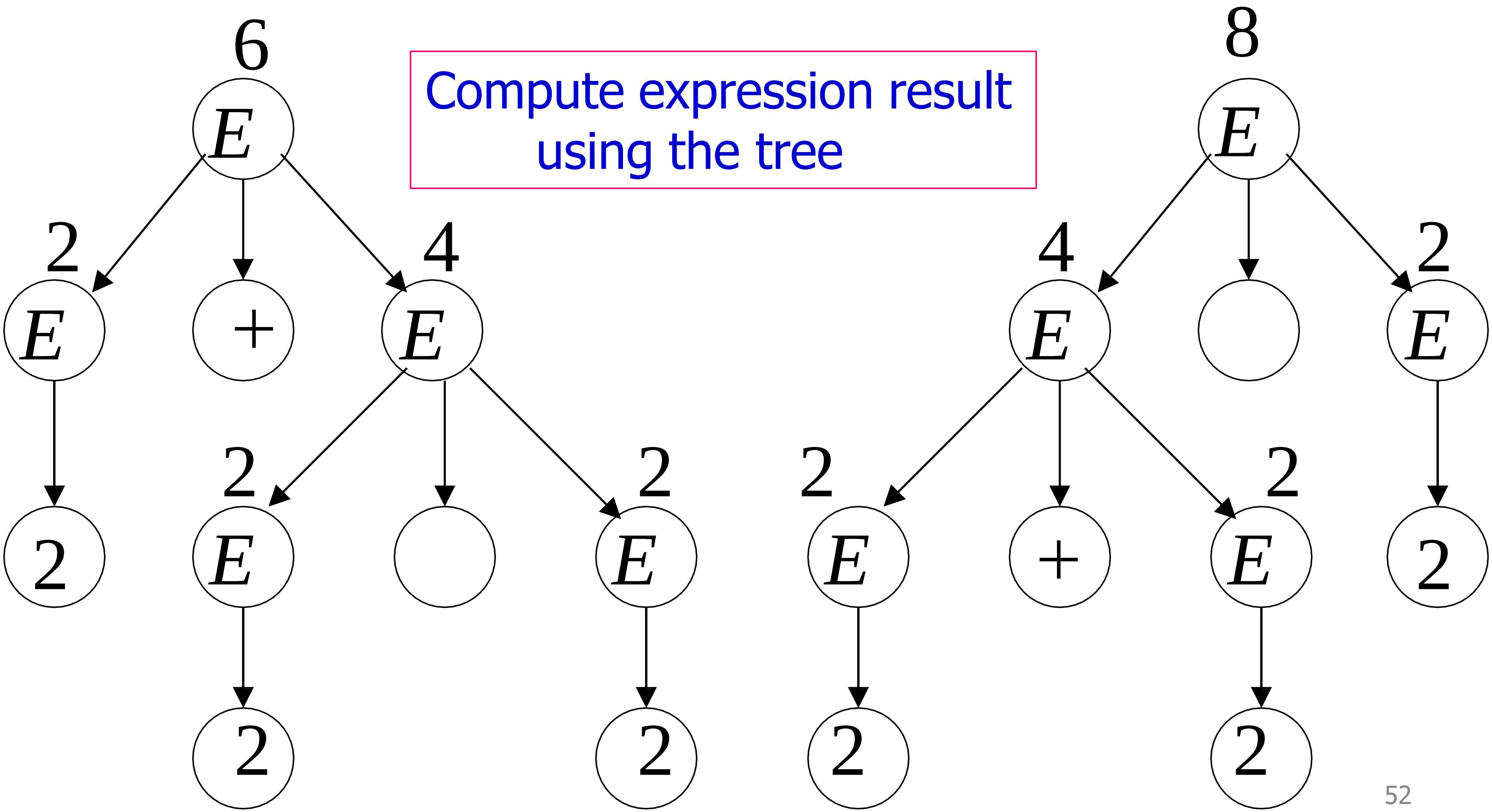
Good Tree

$$2+2*2=6$$

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

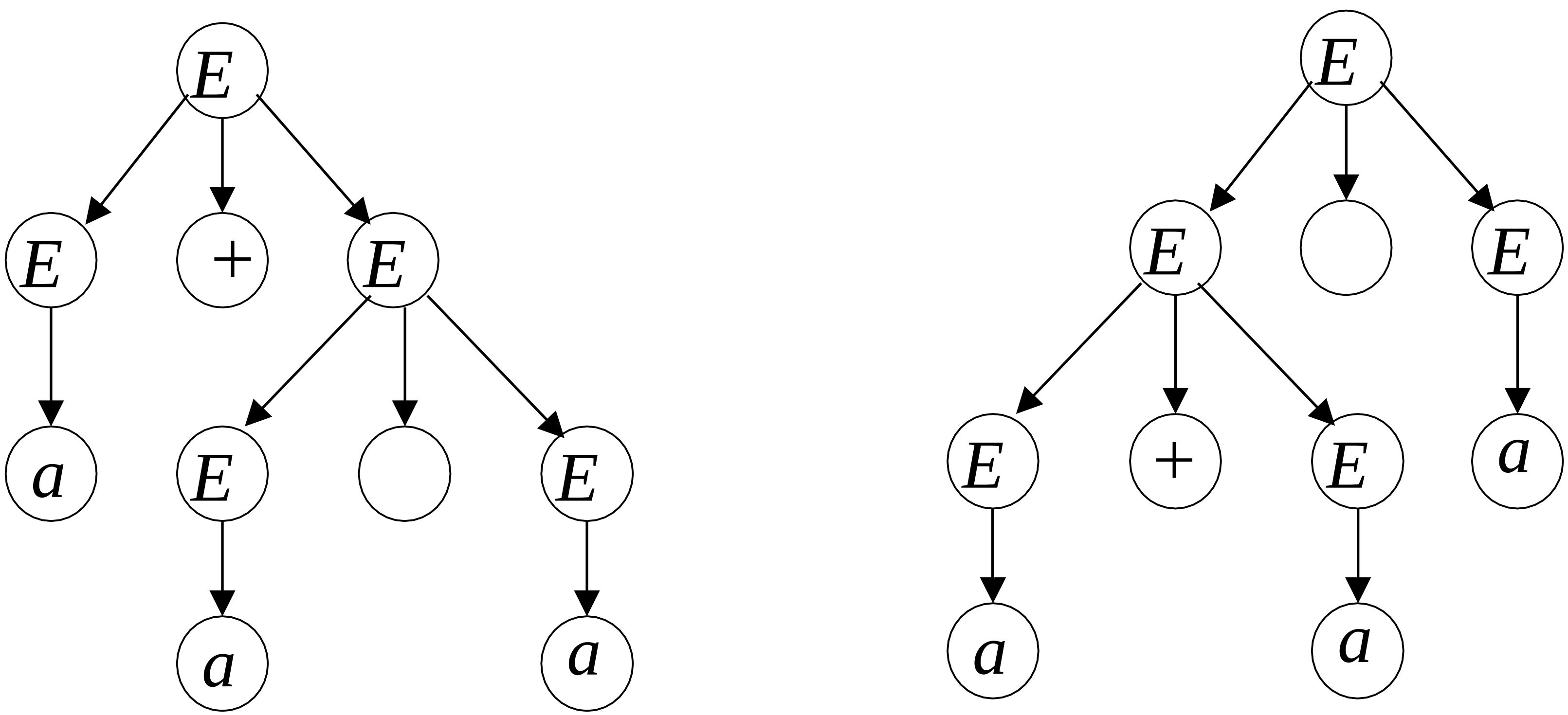
(Two different derivation trees give two different leftmost derivations and vice-versa)

Example:

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

this grammar is ambiguous since

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



$$\boxed{E \to E + E \ \mid E * E \mid (E) \mid a}$$

this grammar is ambiguous also because

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

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

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

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 it is difficult to achieve this

A successful example:

Ambiguous Grammar

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

Equivalent

Non-Ambiguous Grammar

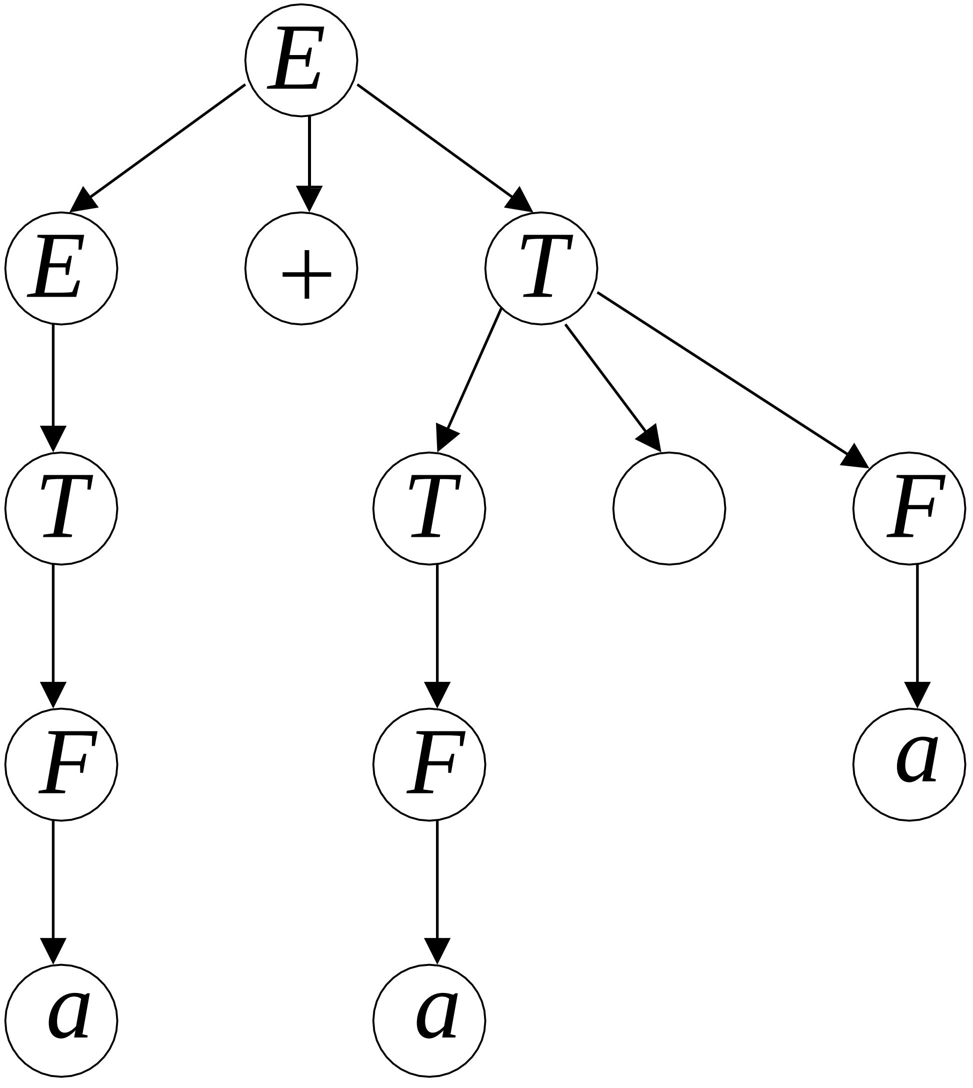
$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (E) \mid a$$

generates the same language

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

$$\Rightarrow a+F*F \Rightarrow a+a*F \Rightarrow a+a*a$$

$$\boxed{\begin{array}{l} E \rightarrow E+T \mid T \\ T \rightarrow T*F \mid F \\ F \rightarrow (E) \mid a \end{array}}$$



Unique derivation tree for $a+a*a$

Ambiguous Grammar example

- Let us consider a CFG:
- CFG: $E \rightarrow I \mid E+E \mid E*E \mid (E)$

  $I \rightarrow a \mid B \mid Ia \mid Ib \mid I0 \mid I1$

Expression: $a + a*a$

LMD: $E \underset{lm}{\Rightarrow} E+E \underset{lm}{\Rightarrow} I+E \underset{lm}{\Rightarrow} a+E \underset{lm}{\Rightarrow} a+E*E \underset{lm}{\Rightarrow} a+I*E \underset{lm}{\Rightarrow} a+a*E \underset{lm}{\Rightarrow} a+a*I \underset{lm}{\Rightarrow} a+a*a$

RMD: $E \underset{rm}{\Rightarrow} E*E \underset{rm}{\Rightarrow} E*I \underset{rm}{\Rightarrow} E*a \underset{rm}{\Rightarrow} E+E*a \underset{rm}{\Rightarrow} E+I*a \underset{rm}{\Rightarrow} E+a*a \underset{rm}{\Rightarrow} I+a*a \underset{rm}{\Rightarrow} a+a*a$

LMD

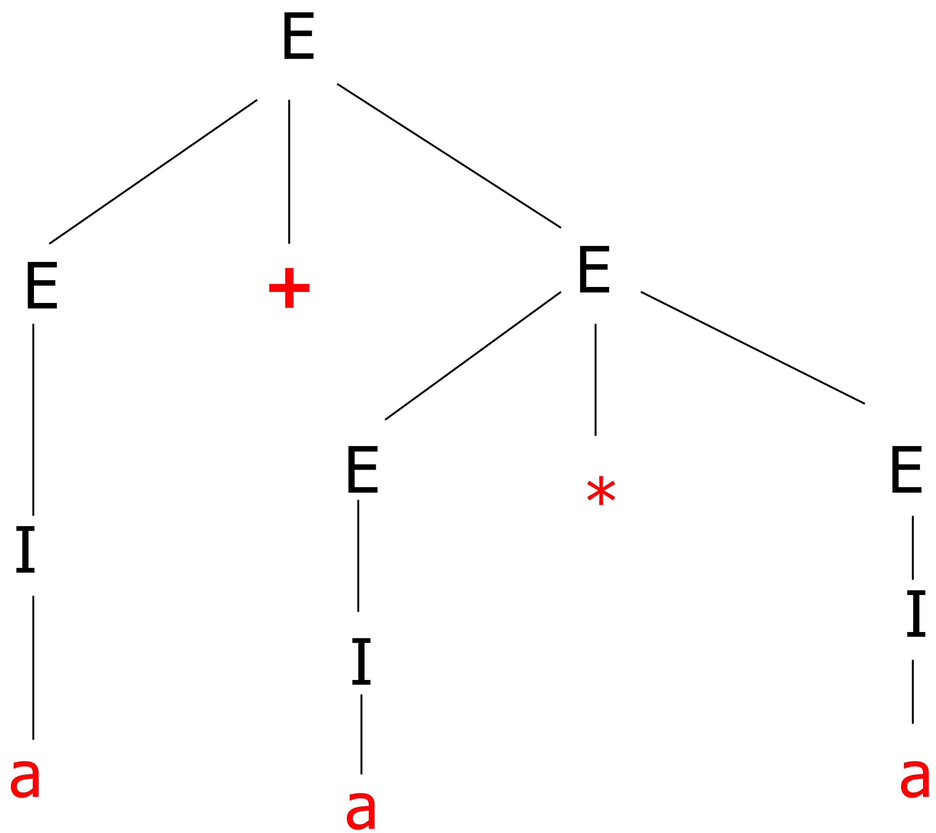


Fig: Trees yield a+a*a

RMD

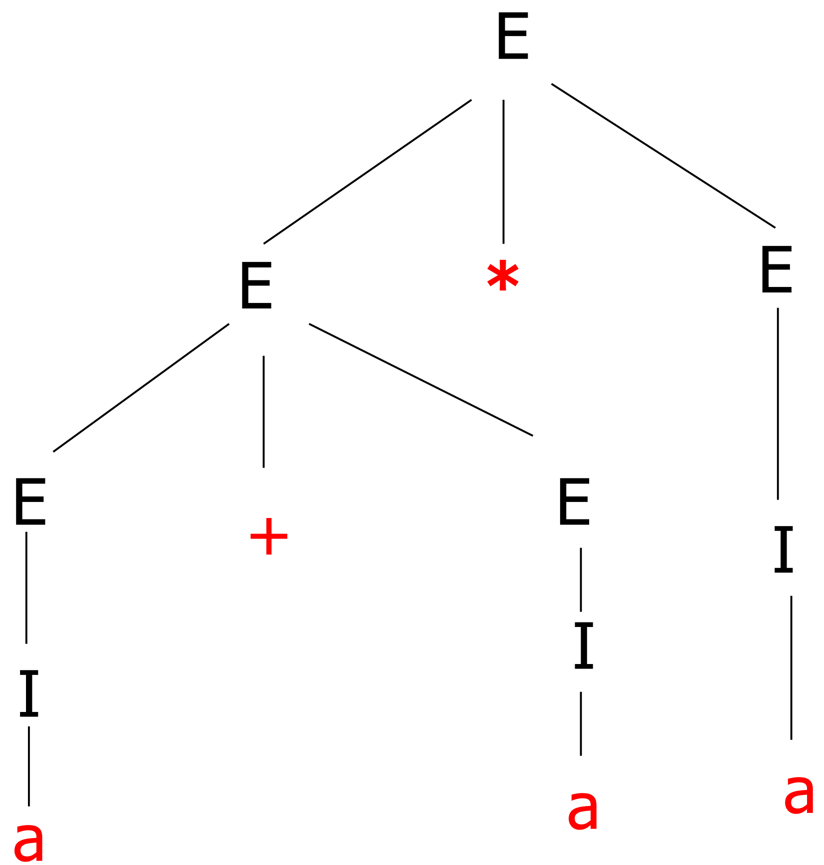


Fig: Trees yield a+a*a

Two causes of ambiguity in the grammar :

1. The precedence of operator is not respected.

2. A sequence of identical operators can group either from the left or from the right.

Removing Ambiguity from Grammar

The solution of the problem of enforcing precedence is to introduce several different variables.

1. A *factor*- is an expression that cannot be broken apart by any adjacent operators. The only factors in our expression language are:

   i. Identifiers: It is not possible to separate the letters of identifier by attaching an operator.

   ii. Any parenthesized expression, no matter what appears inside the parenthesis.

2. A *term*- is an expression that cannot be broken by the '+' operator. Term is product of one or more *factor*s.

3. An *expression*-is a sum of one or more terms.

Removing Ambiguity from Grammar

- Let us consider a CFG:
- CFG: $E \rightarrow I \mid E+E \mid E*E \mid (E)$

  $I \rightarrow a \mid B \mid Ia \mid Ib \mid I0 \mid I1$

- An unambiguous expression grammar :

  $I \rightarrow a \mid B \mid Ia \mid Ib \mid I0 \mid I1$

  $F \rightarrow I \mid (E)$

  $T \rightarrow F \mid T*F$

  $E \rightarrow T \mid E+T$

Exercises

Solve the following exercises:

5.1.1, 5.1.2, 5.1.5, 5.2.1,5.4.1,5.4.2,5.4.3, 5.4.4, 5.4.5,5.4.6, 5.4.7