

Context-Free Grammars

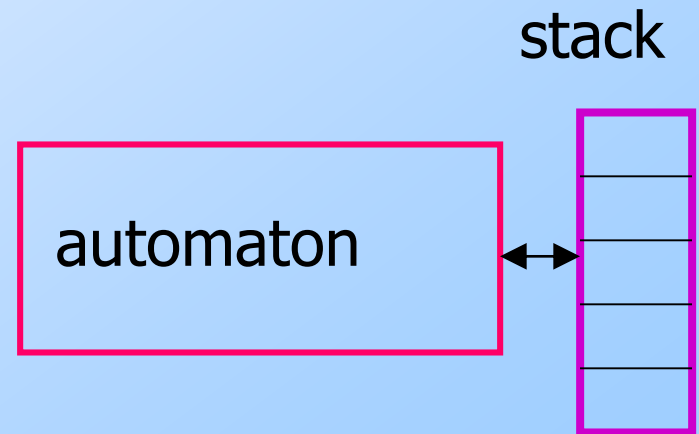
Md. Khorshed Alam
CSE, NDUB

"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 definition (DTD), XML (extensible markup language)

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 $\{0,1\}$.

◆ **palindrome** : 0110,11011

◆ **Not palindrome**: 011, 0101

◆ **Properties**: it must begin & end with the same symbol. When the 1st & last symbols are removed, the resulting string must also be a palindrome

◆ **Basis**: ε , 0 and 1 are palindromes

Induction: if w is a palindromes, so are **$0w0$** & **$1w1$** . 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. $P \rightarrow 1P1$
- Basis**
- Induction**

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 $\{ 0^n 1^n \mid n \geq 1 \}$

- ◆ Productions:

$S \rightarrow 01$

$S \rightarrow 0S1$

- ◆ Basis: 01 is in the language.

- ◆ Induction: if w is in the language, then so is $0w1$.

Definition of CFG

- 1. Terminals:** There is a finite set of symbols that form the strings of the language being defined.
- 2. Non-terminals:** There is a finite set of variables. Each variable represents a language (a set of strings).
- 3. Start symbol:** One of the variables represents the language being defined.

4. **Production:** There is a finite set of productions/rules that represent the recursive definition of a language. Each production consists of:
- (a) A variable that is being defined by the production. This variable is often called the **head of the production**.
 - (b) The production symbol \rightarrow
 - (c) A string of zero or more terminals & variables. This string called **body of the production**, represents one way to form strings in the language of the variable of the head.

CFG Formalism

- ◆ *Terminals* = symbols of the alphabet of the language being defined.
- ◆ *Variables* = *non-terminals* = 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** \rightarrow **string of variables & terminals**.
- ◆ **Convention:**
 - ◆ A, B, C, \dots are variables.
 - ◆ a, b, c, \dots are terminals.
 - ◆ \dots, X, Y, Z are either terminals or variables.
 - ◆ \dots, w, x, y, z are strings of terminals only.
 - ◆ $\alpha, \beta, \gamma, \dots$ 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 & 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 =
 - $S \rightarrow 01$
 - $S \rightarrow 0S1$

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

All are sets

Example of Context-Free Grammar

$$S \rightarrow aSb \mid \varepsilon$$

02 Productions

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

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

$$V = \{S\}$$

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}*.

◆ $G = (\{E, I\}, T, P, E)$

◆ $T = \{0, 1, a, b, +, *, (,)\}$

Productions:

- | | |
|--------------------------|------------------------|
| 1. $E \rightarrow I$ | 6. $I \rightarrow b$ |
| 2. $E \rightarrow E + E$ | 7. $I \rightarrow Ia$ |
| 3. $E \rightarrow E * E$ | 8. $I \rightarrow Ib$ |
| 4. $E \rightarrow (E)$ | 9. $I \rightarrow I0$ |
| 5. $I \rightarrow a$ | 10. $I \rightarrow I1$ |

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

Derivation using grammar

◆ $a*(a+b00)$

1. $E \Rightarrow E * E$ (3)
 $\Rightarrow I * E$ (1)
 $\Rightarrow a * E$ (5)
 $\Rightarrow a * (E)$ (4)
 $\Rightarrow a * (E + E)$ (2)
 $\Rightarrow a * (I + E)$ (1)
 $\Rightarrow a * (a + E)$ (5)
 $\Rightarrow a * (a + I)$ (1)
 $\Rightarrow a * (a + I0)$ (9)
 $\Rightarrow a * (a + I00)$ (9)
 $\Rightarrow a * (a + b00)$ (6)

Productions:

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

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

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

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

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

$S = S$

$R = \{$

$S \rightarrow NP VP$

$S \rightarrow Aux NP VP$

$S \rightarrow VP$

$NP \rightarrow Det NOM$

$NOM \rightarrow Noun$

$NOM \rightarrow Noun NOM$

$VP \rightarrow Verb$

$VP \rightarrow Verb NP$

$Det \rightarrow that \mid this \mid a \mid the$

$Noun \rightarrow book \mid flight \mid meal \mid man$

$Verb \rightarrow book \mid include \mid read$

$Aux \rightarrow does$

$\}$

An example of CFG

S → NP VP	Det → <i>that</i> <i>this</i> <i>a</i> <i>the</i>
S → Aux NP VP	Noun → <i>book</i> <i>flight</i> <i>meal</i> <i>man</i>
S → VP	Verb → <i>book</i> <i>include</i> <i>read</i>
NP → Det NOM	Aux → <i>does</i>
NOM → Noun	
NOM → Noun NOM	
VP → Verb	
VP → Verb NP	

S → NP VP
→ Det NOM VP
→ *The* NOM VP
→ *The* Noun VP
→ *The man* VP
→ *The man* Verb NP
→ *The man read* NP
→ *The man read* Det NOM
→ *The man read this* NOM
→ *The man read this* Noun
→ *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. $S \rightarrow AB$ | 2. $A \rightarrow aaA$ | 4. $B \rightarrow Bb$ |
| | 3. $A \rightarrow \lambda$ | 5. $B \rightarrow \lambda$ |

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

Leftmost derivation order of string : aab

$$\begin{array}{ccccccccc}
 & 1 & & 2 & & 3 & & 4 & & 5 \\
 S & \Rightarrow & AB & \Rightarrow & aaAB & \Rightarrow & aaB & \Rightarrow & aaBb & \Rightarrow & aab
 \end{array}$$

At each step, we substitute the leftmost variable

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

Rightmost derivation order of string : aab

$$\begin{array}{cccccc}
 1 & & 4 & & 5 & & 2 & & 3 \\
 S \Rightarrow AB \Rightarrow ABb \Rightarrow Ab \Rightarrow aaAb \Rightarrow aab
 \end{array}$$

At each step, we substitute the rightmost variable

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

Leftmost derivation of aab :

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

Rightmost derivation of aab :

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

Another Example: LMD

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

◆ $a^*(a+b00)$

◆ $E \Rightarrow E^*E$

$lm \Rightarrow I^*E$

$lm \Rightarrow a^*E$

$lm \Rightarrow a^*(E)$

$lm \Rightarrow a^*(E+E)$

$lm \Rightarrow a^*(I+E)$

$lm \Rightarrow a^*(a+E)$

$lm \Rightarrow a^*(a+I)$

$lm \Rightarrow a^*(a+I0)$

$lm \Rightarrow a^*(a+I00)$

$lm \Rightarrow a^*(a+b00)$

Another Example: LMD

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

◆ $a^*(a+b00)$

◆ $E \Rightarrow E^*E$

$rm \Rightarrow E^*(E)$

$rm \Rightarrow E^*(E+E)$

$rm \Rightarrow E^*(E+I)$

$rm \Rightarrow E^*(E+I0)$

$rm \Rightarrow E^*(E+I00)$

$rm \Rightarrow E^*(E+b00)$

$rm \Rightarrow E^*(I+b00)$

$rm \Rightarrow E^*(a+b00)$

$rm \Rightarrow I^*(a+b00)$

$rm \Rightarrow a^*(a+b00)$

Derivation/Parse Tree

- ◆ Representation for derivations which shows clearly how 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:

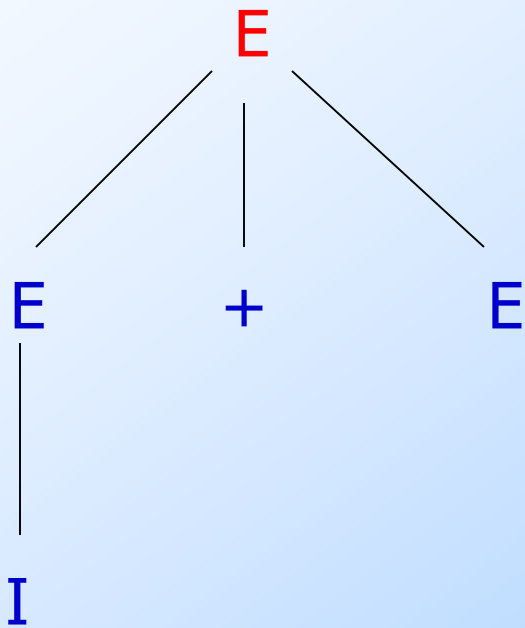
1. Each interior node is labeled by a variable V
2. Each leaf is labeled by either variable, a terminal or ϵ
3. If an interior node is labeled A , & its children are labeled

X_1, X_2, \dots, X_k

respectively, from the left, then $A \rightarrow X_1 X_2 \dots X_k$ is a production.

Example

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



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

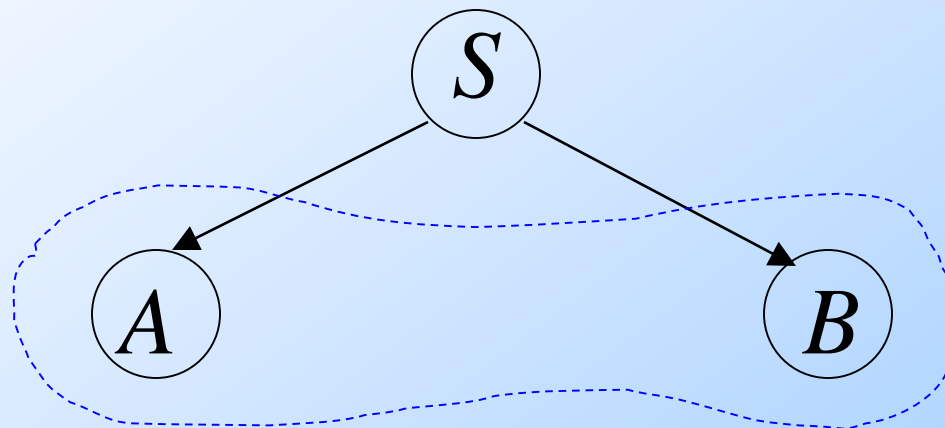
Example

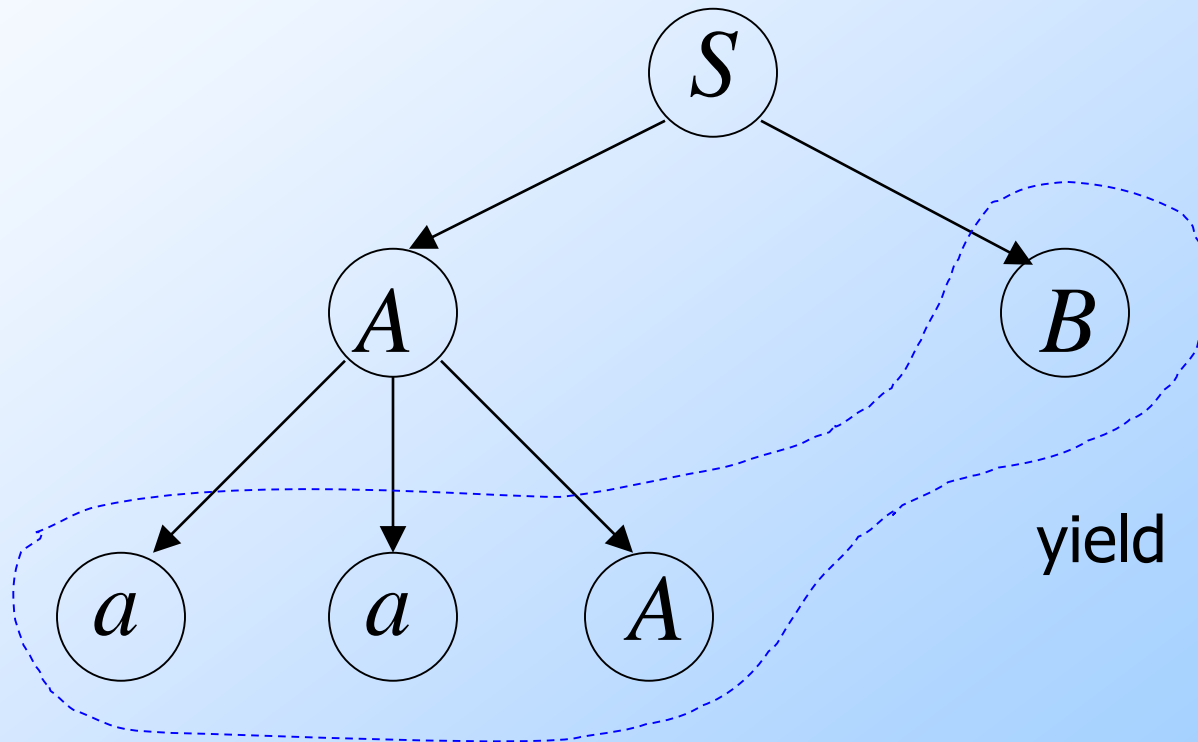
Consider the same example grammar:

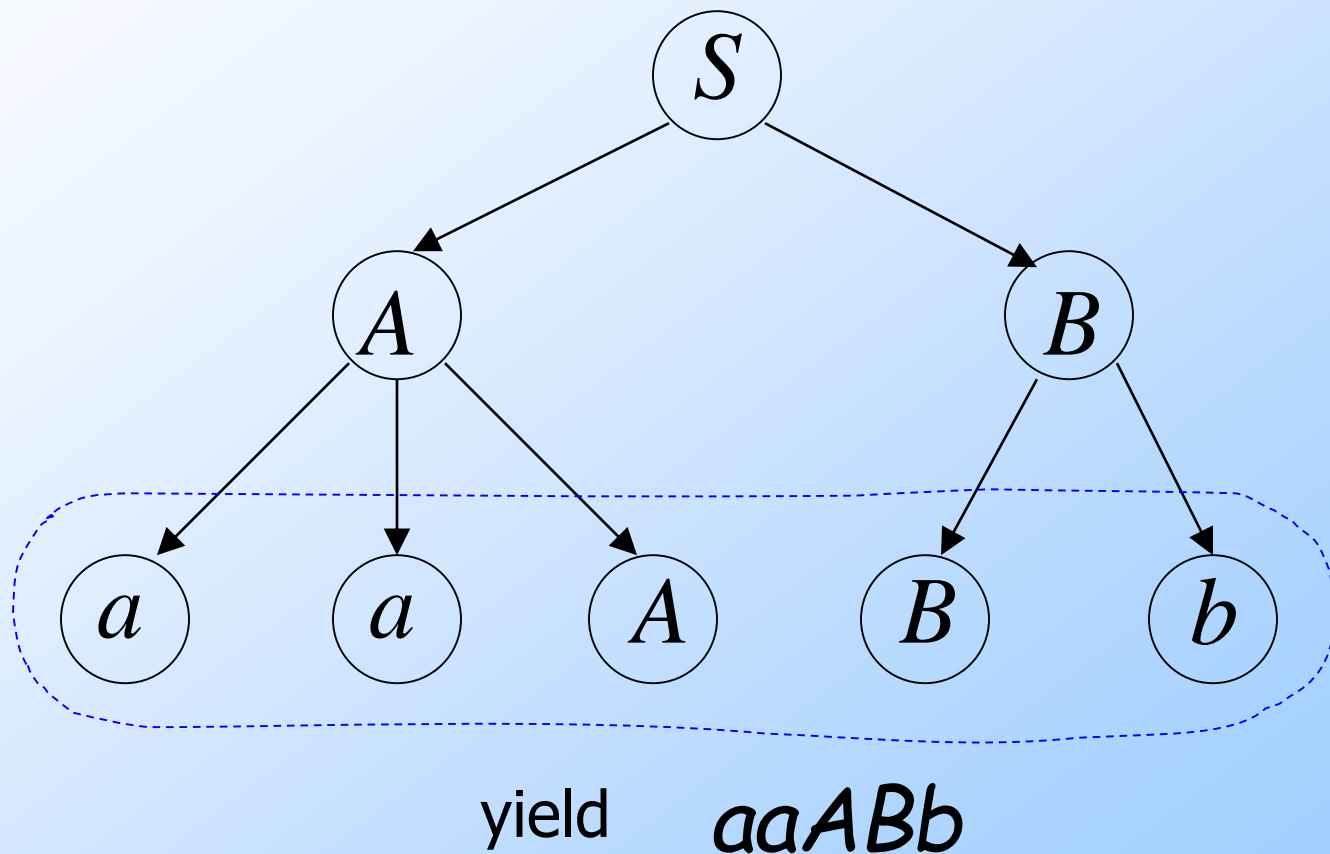
$$S \rightarrow AB \quad A \rightarrow aaA \mid \lambda \quad B \rightarrow Bb \mid \lambda$$

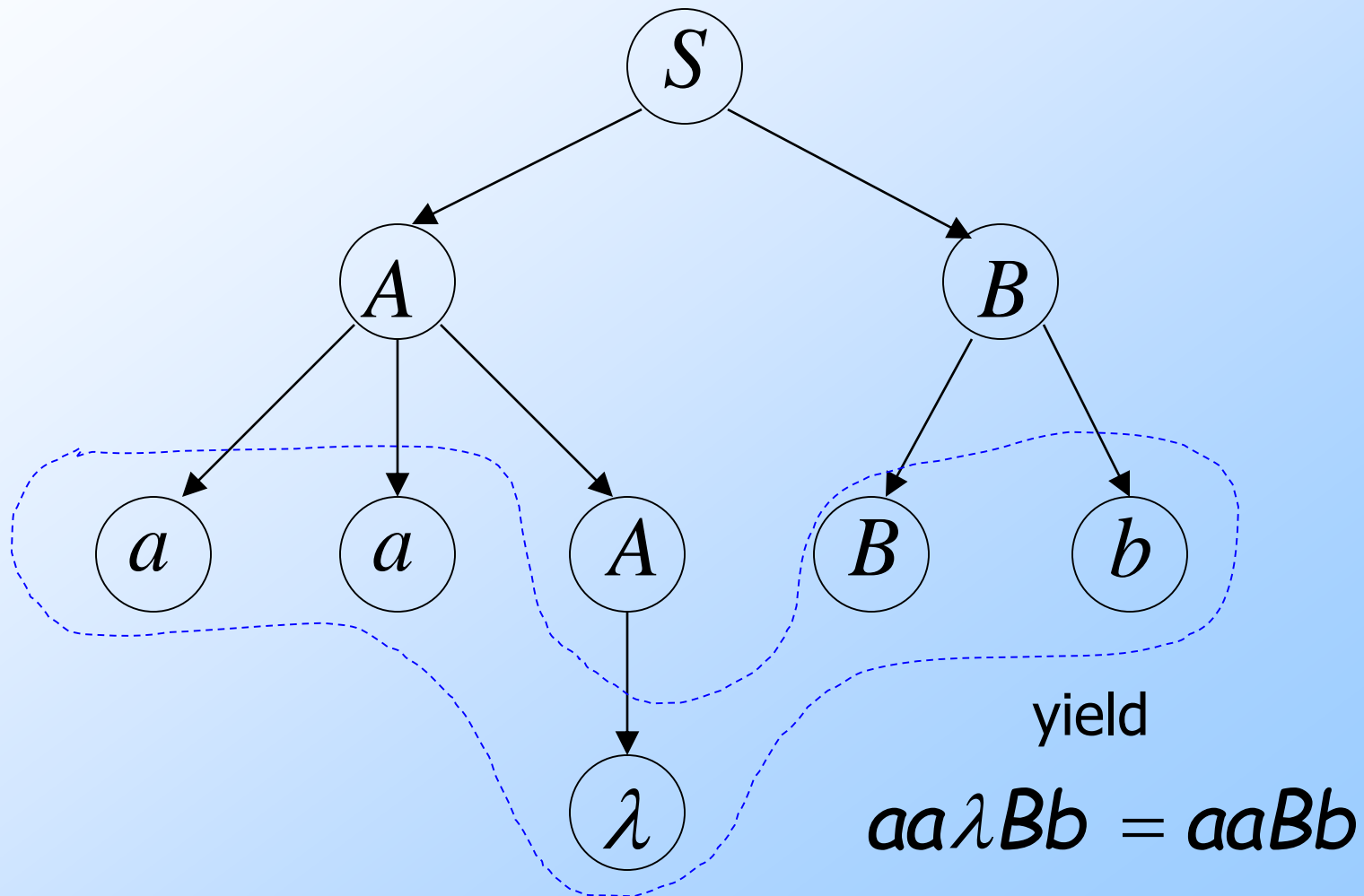
And a derivation of aab :

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

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

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

$S \rightarrow AB$ $A \rightarrow aaA \mid \lambda$ $B \rightarrow Bb \mid \lambda$ $S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb$ 

$S \rightarrow AB$ $A \rightarrow aaA \mid \lambda$ $B \rightarrow Bb \mid \lambda$ $S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb \Rightarrow aaBb$ 

$$S \rightarrow AB$$

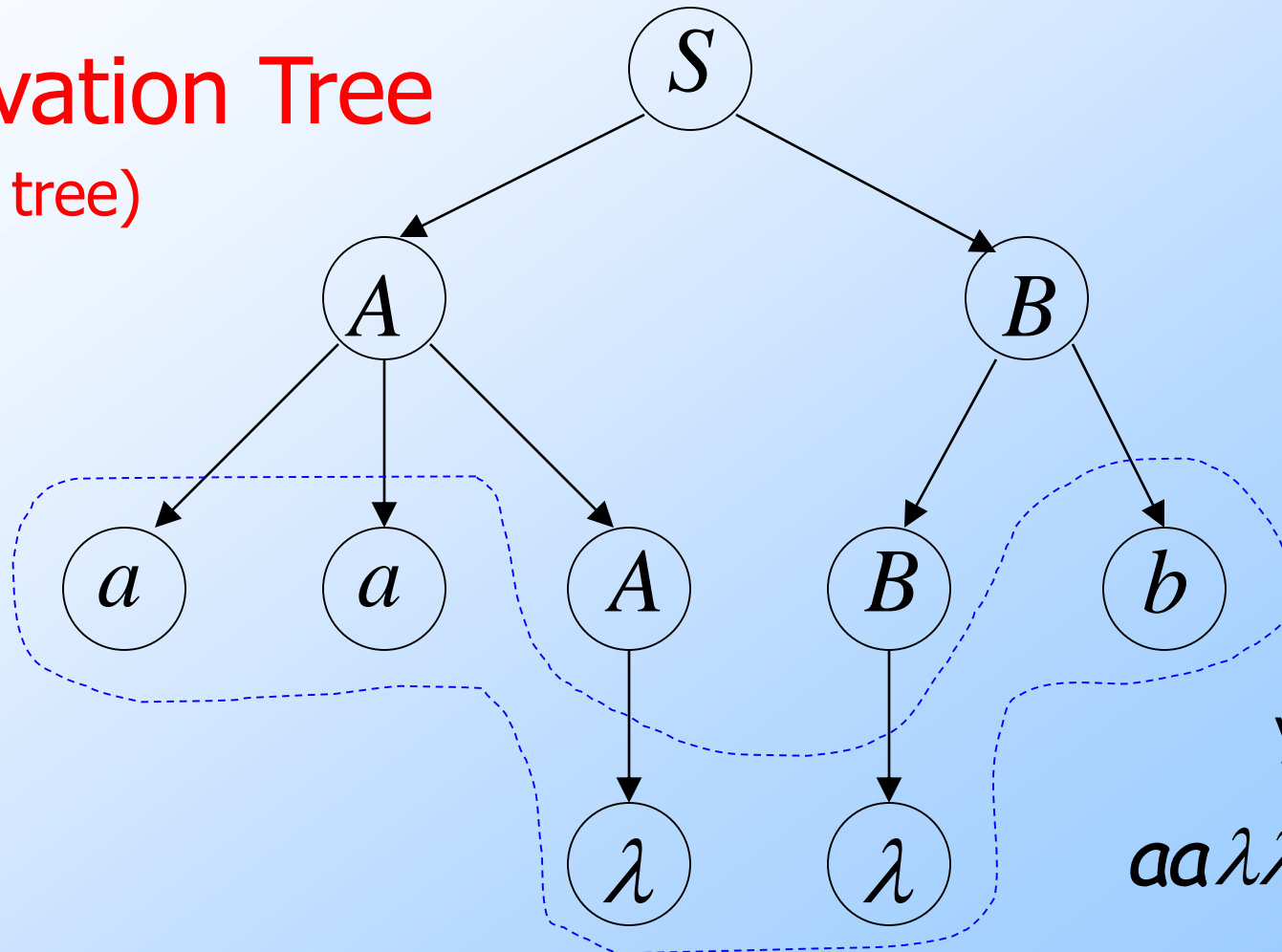
$$A \rightarrow aaA \mid \lambda$$

$$B \rightarrow Bb \mid \lambda$$

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

Derivation Tree

(parse tree)



Sometimes, derivation order doesn't matter

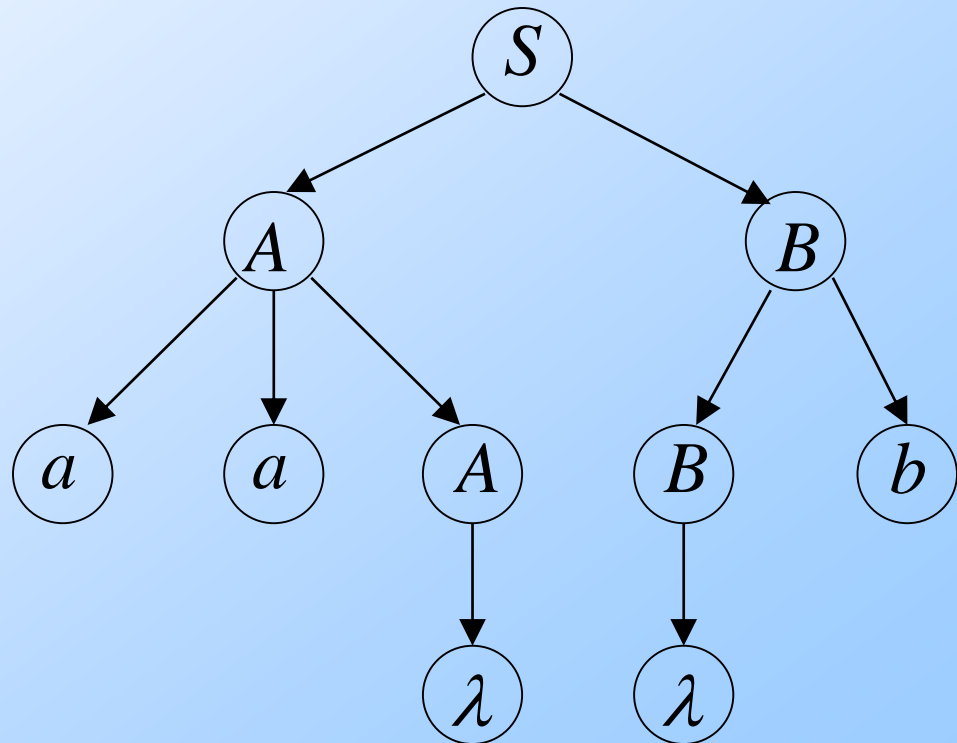
Leftmost derivation:

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

Rightmost derivation:

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

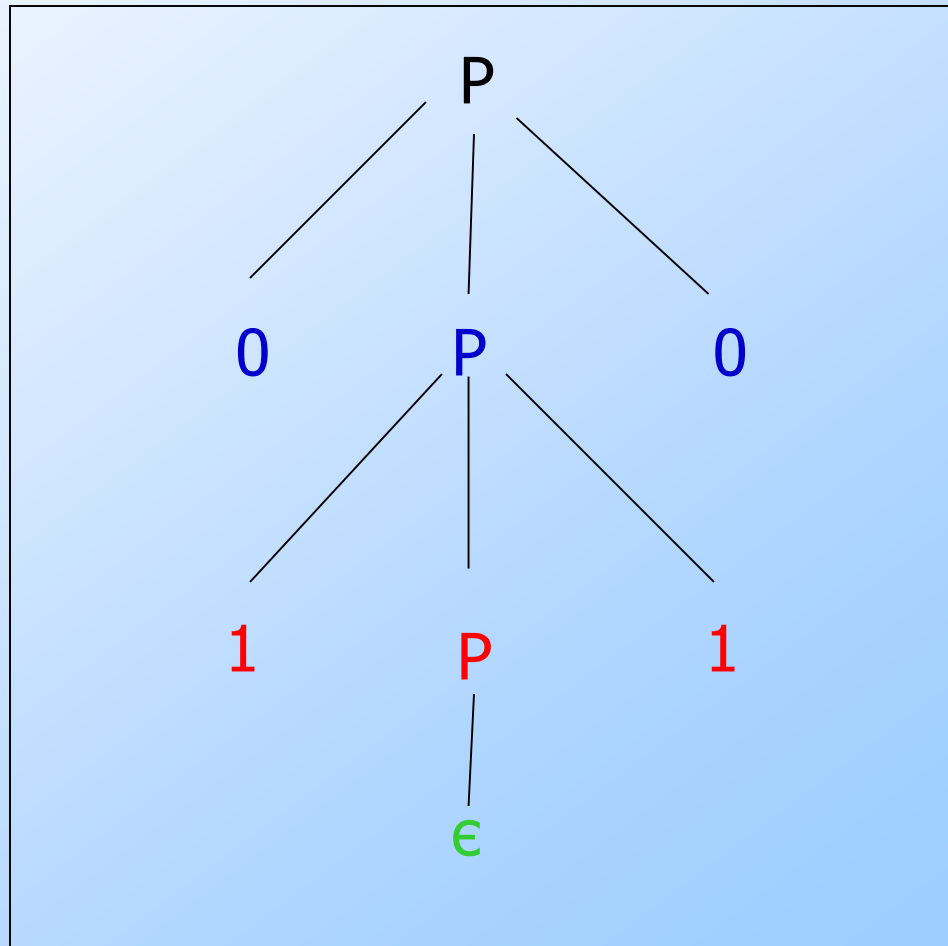
Give same
derivation tree



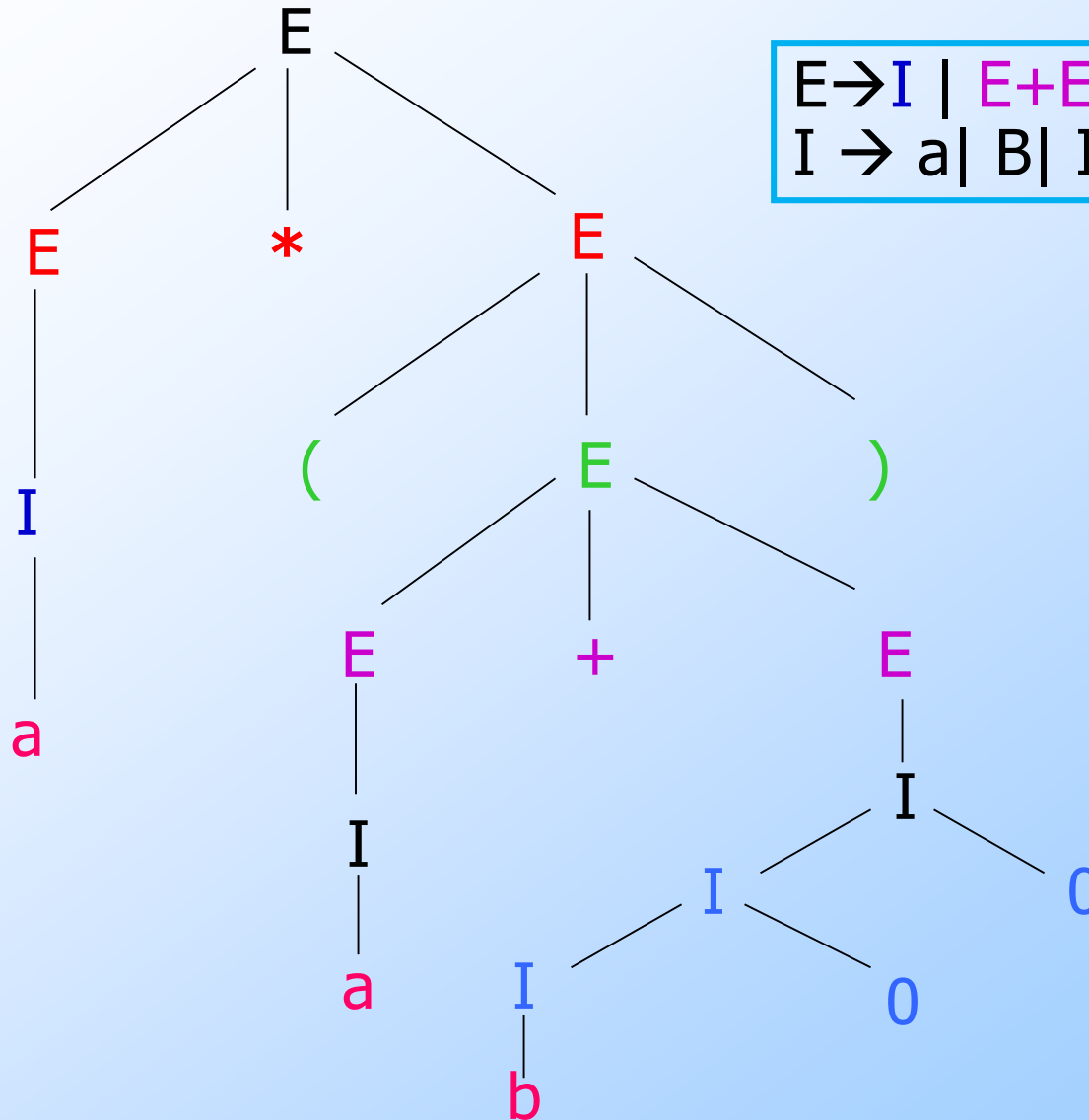
Example

◆ A parse tree showing the derivation $P \xrightarrow{*} 0110$.

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

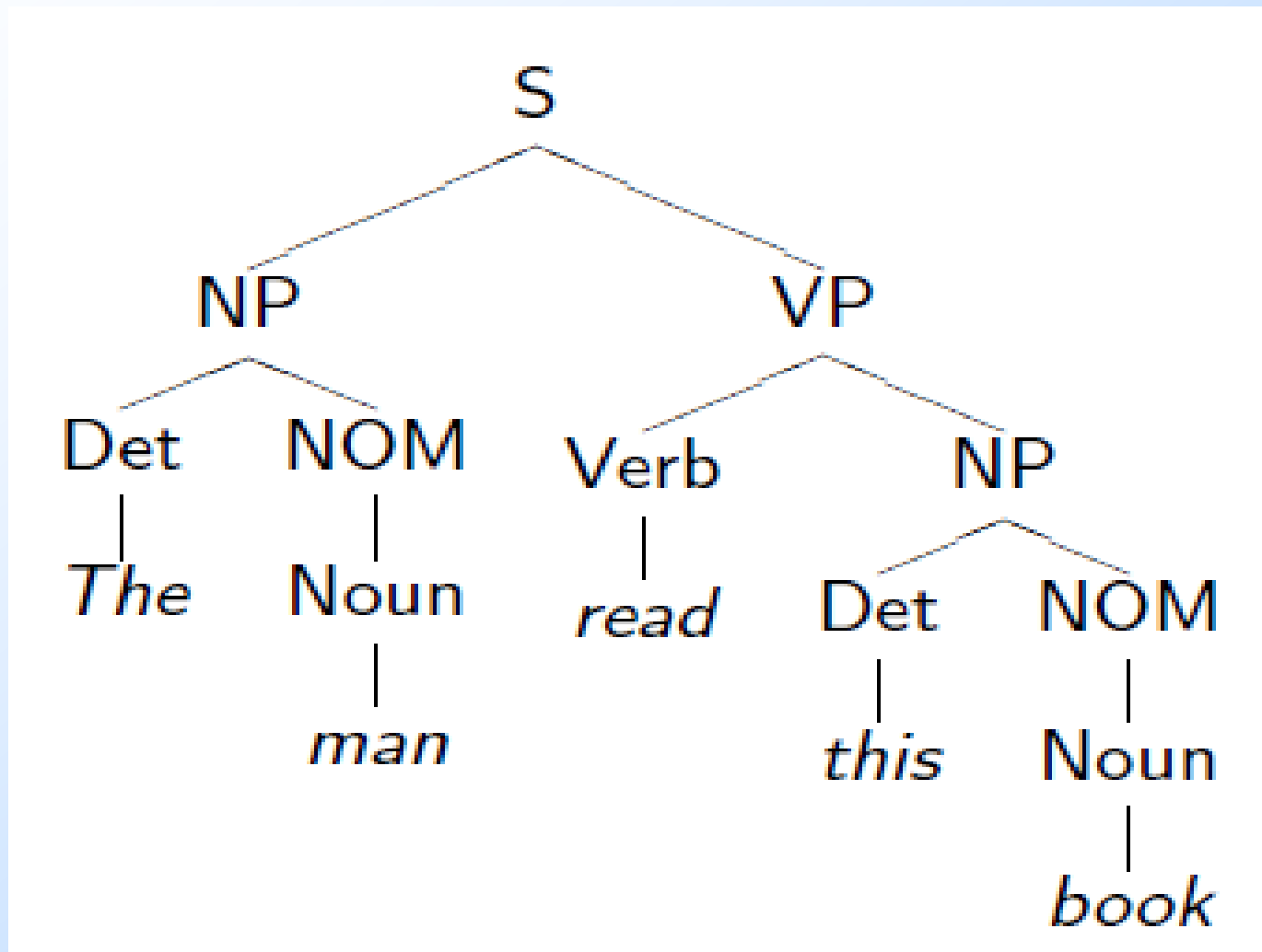


Parse tree showing $a^*(a+b00)$



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

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 $(V \cup T)^*$ such that $S \xRightarrow{*} \alpha$ is a **sentential form.**
- ◆ If $S \xRightarrow[im]{*} \alpha$, then α is a left-sentential form,
- ◆ and if $S \xRightarrow[rm]{*} \alpha$ then α is a right-sentential form

Sentential Forms: Example

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

- ◆ $E^*(I+E)$ 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

◆ $E \xRightarrow{\text{lm}} E^*E \xRightarrow{\text{lm}} I^*E \xRightarrow{\text{lm}} a^*E$ Left sentential form

◆ $E \xRightarrow{\text{rm}} E^*E \xRightarrow{\text{rm}} E^*(E) \xRightarrow{\text{rm}} E^*(E+E)$ 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

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

Example strings:

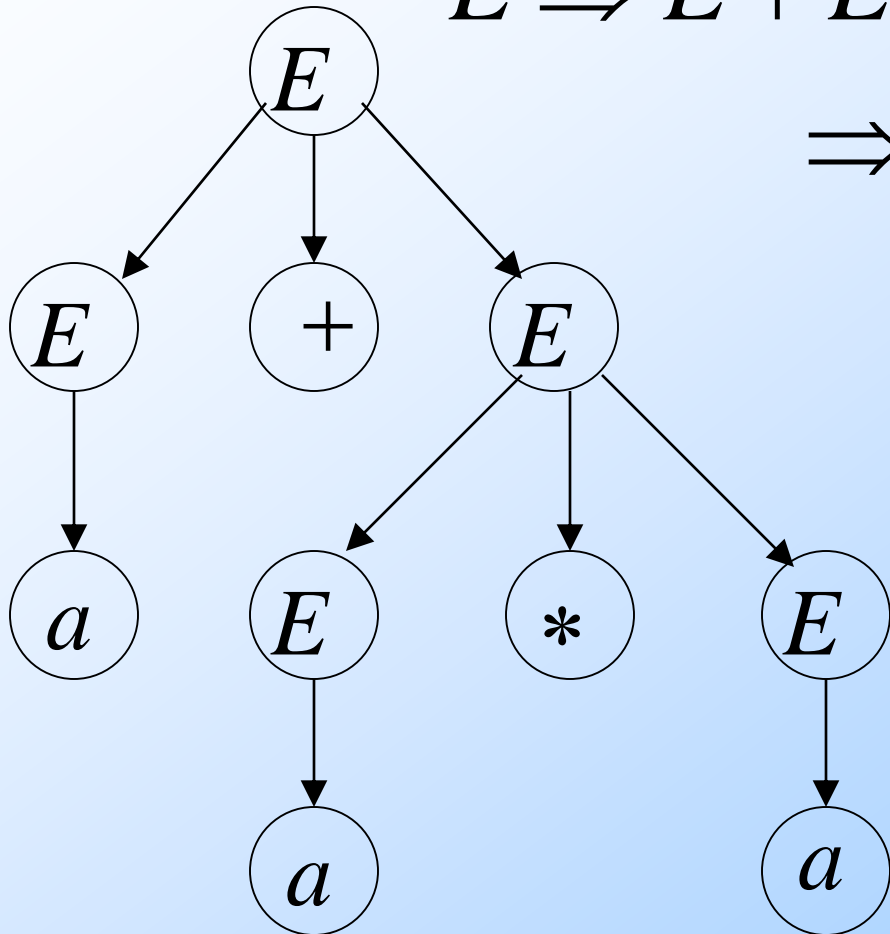
$$(a + a) * a + (a + a * (a + a))$$



Denotes any number

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

$$\begin{aligned} E &\Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E \\ &\Rightarrow a + a * E \Rightarrow a + a * a \end{aligned}$$



A leftmost derivation
for

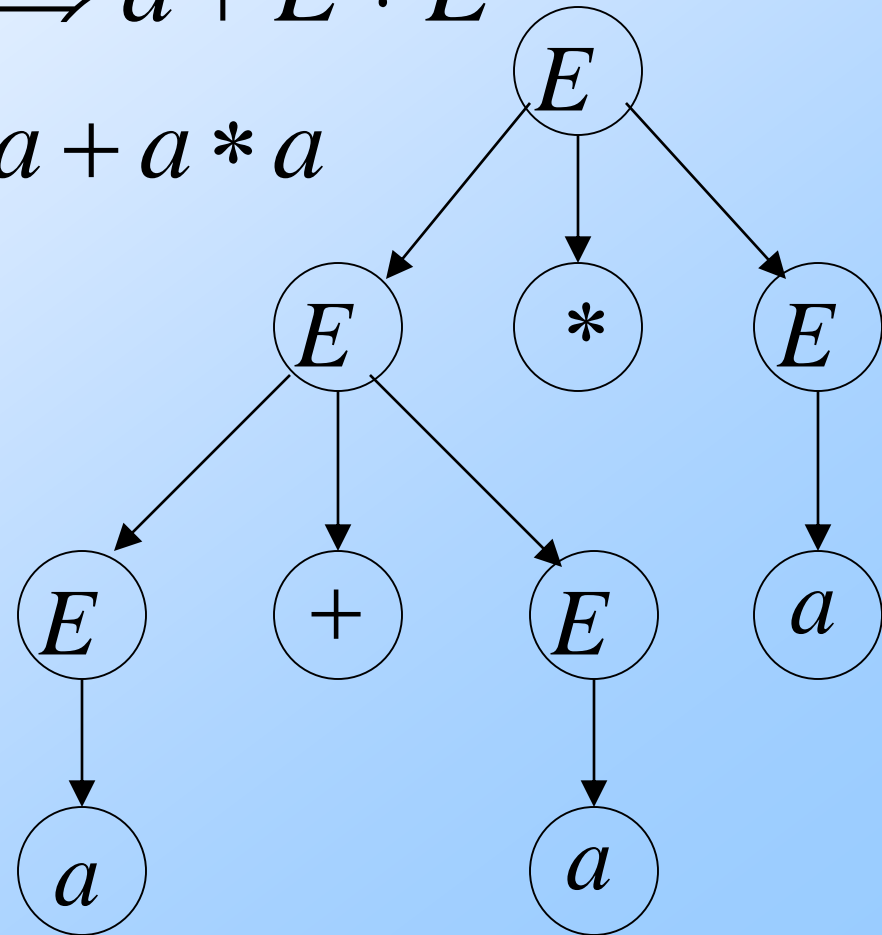
$$a + a * a$$

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

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

Another
leftmost derivation
for

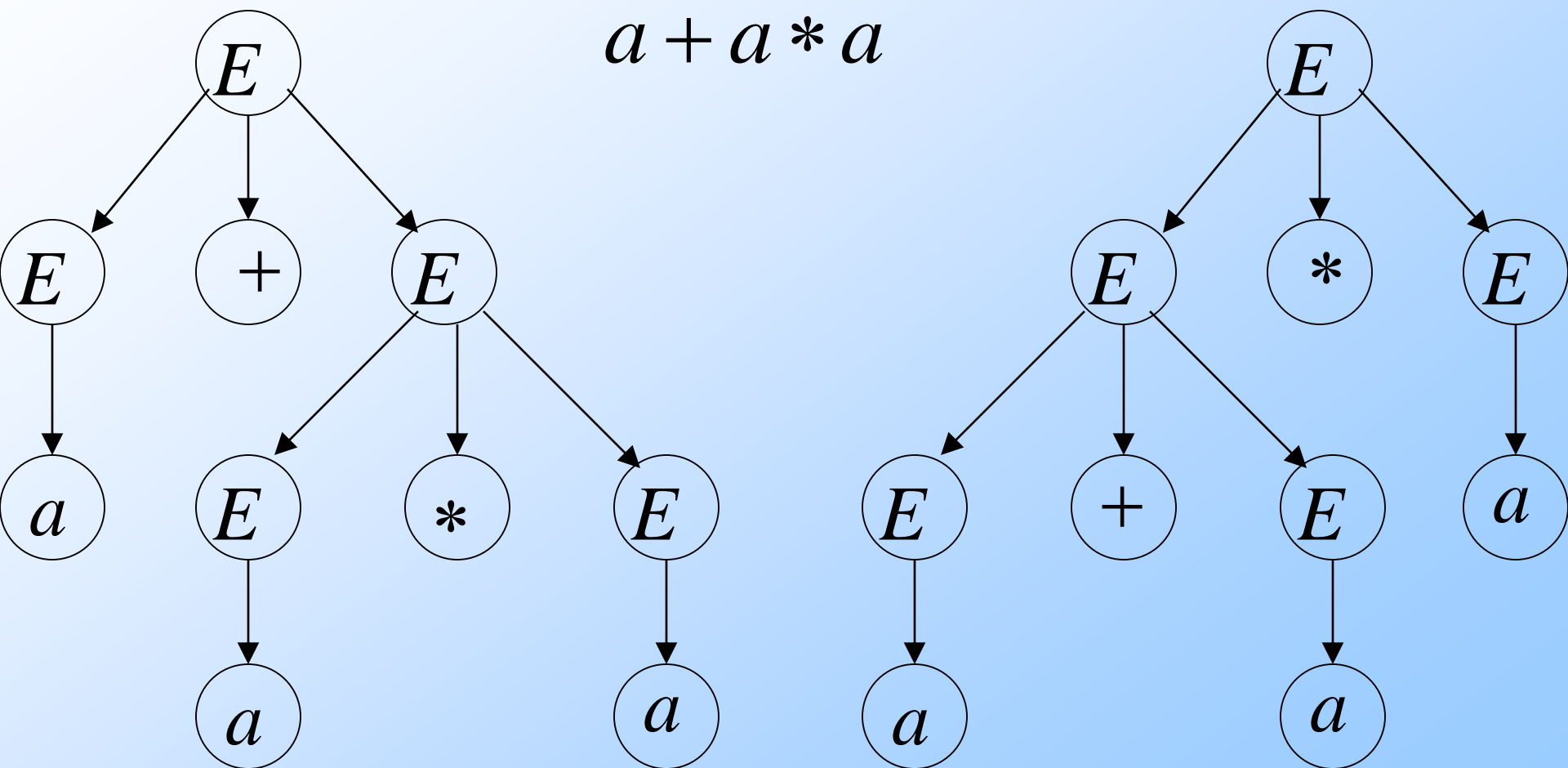
$$a + a * a$$



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

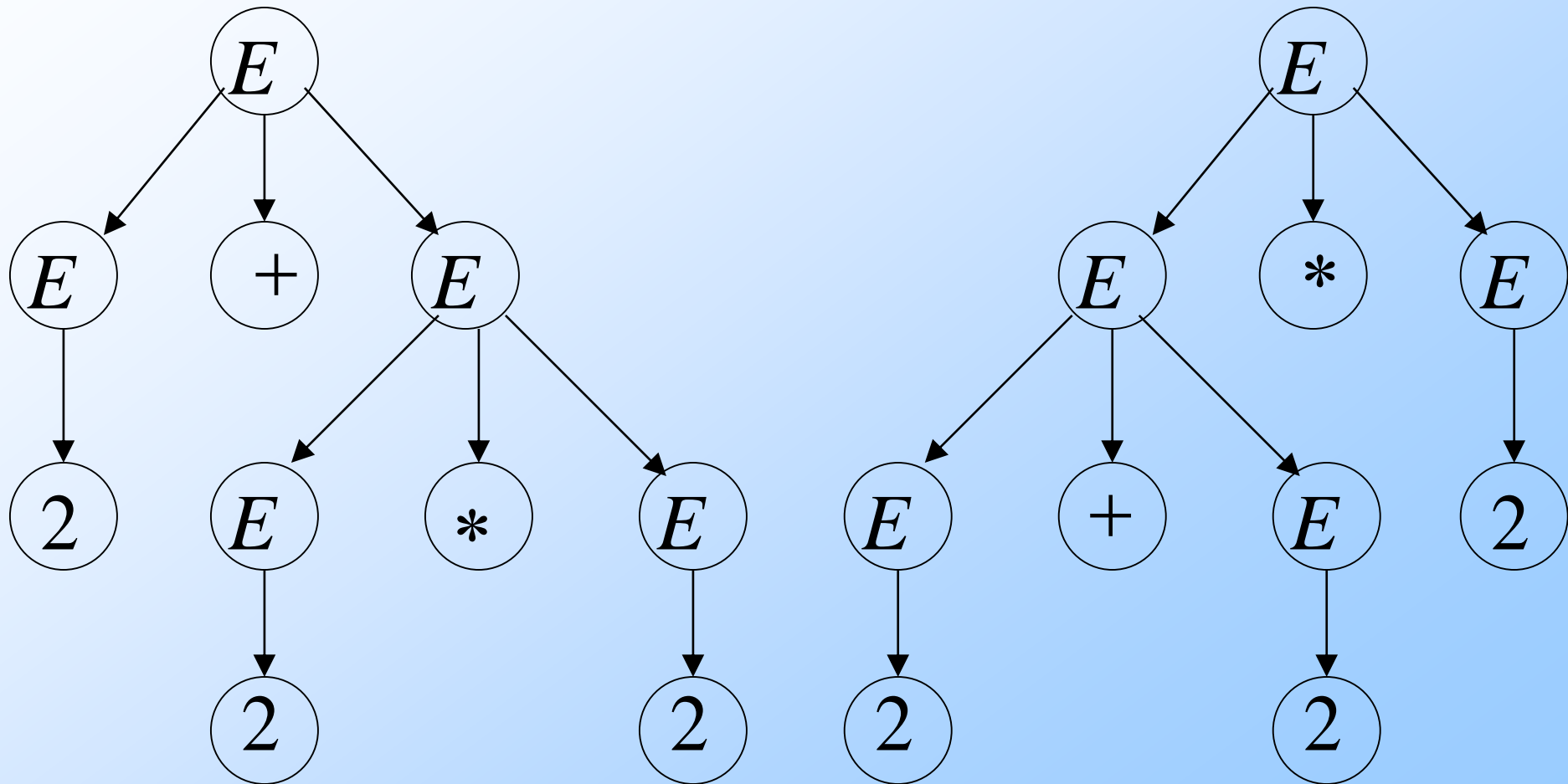
Two derivation trees for

$$a + a * a$$



take $a = 2$

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



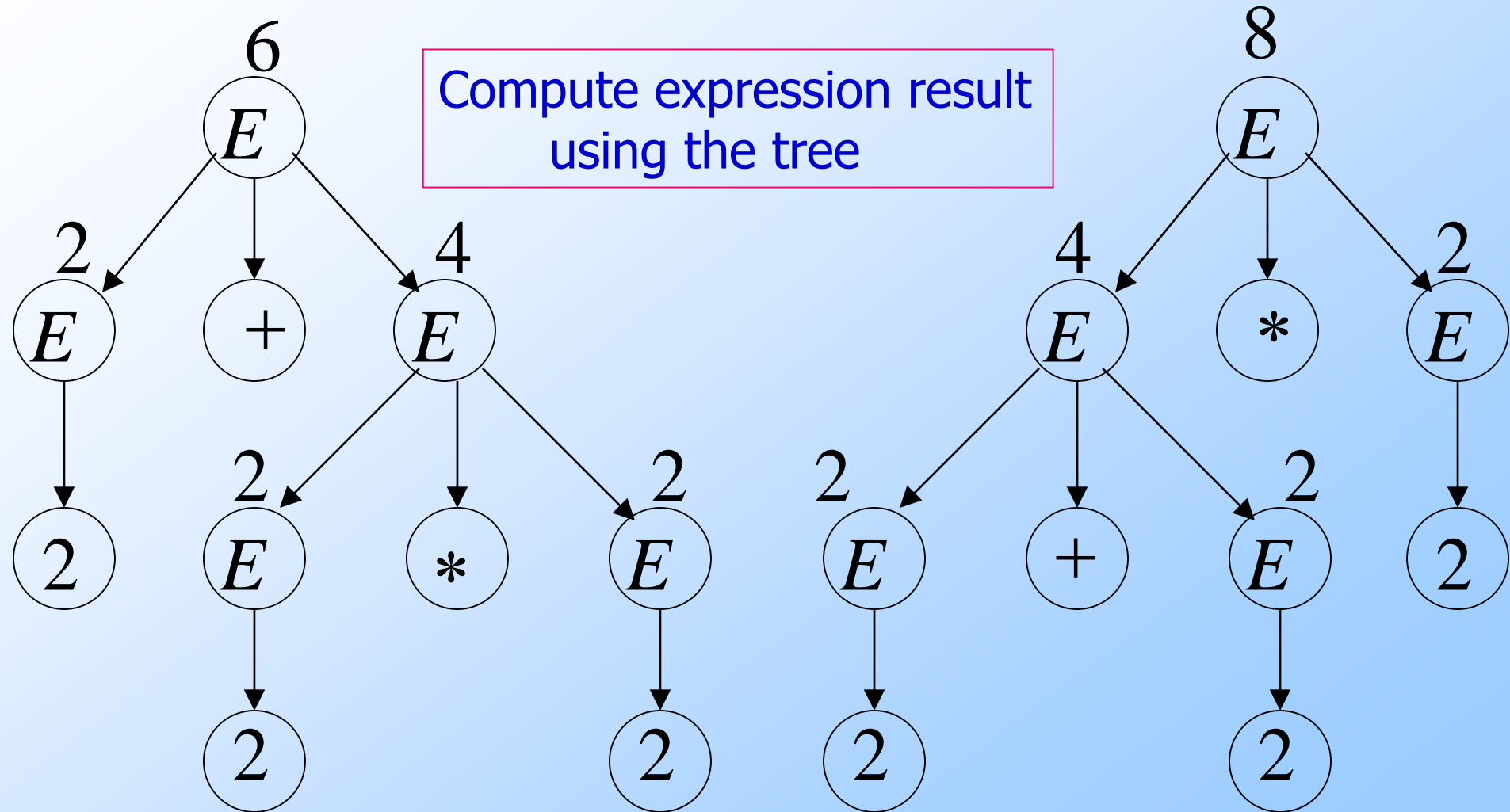
Good Tree

$$2 + 2 * 2 = 6$$

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

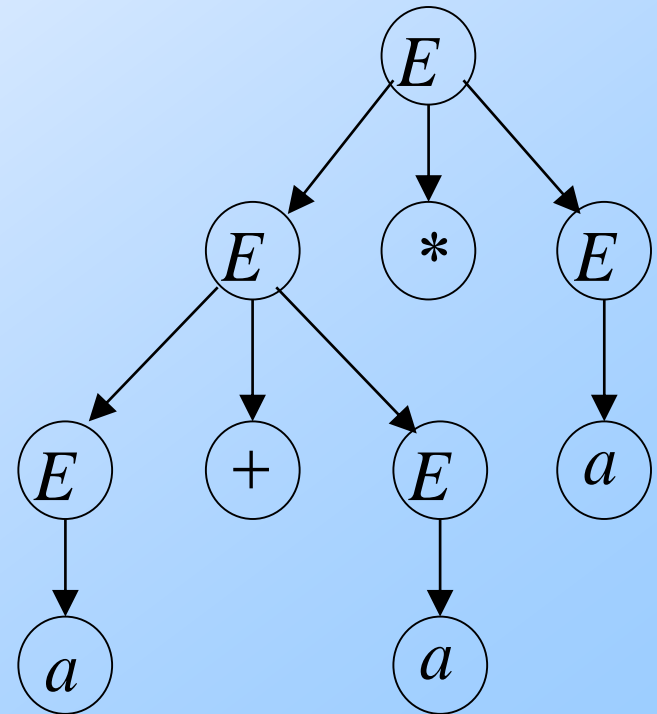
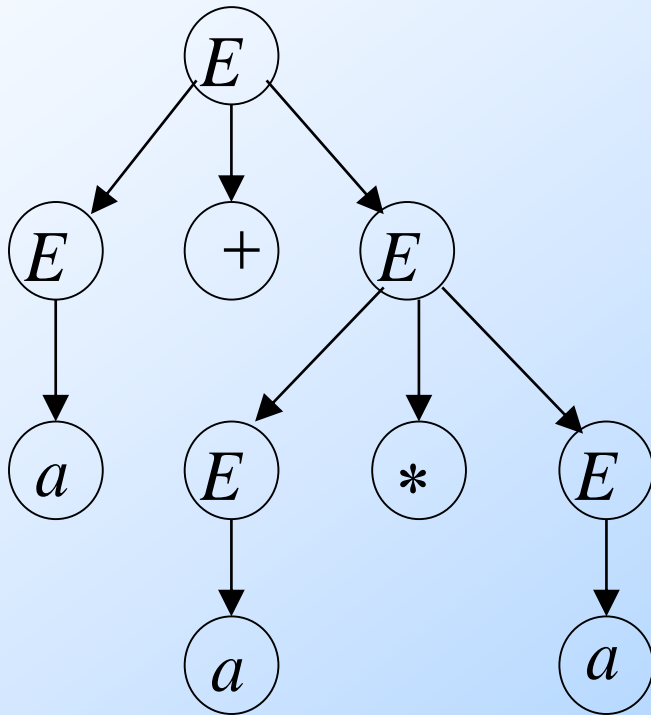
(Two different derivation trees give two different leftmost derivations and vice-versa)

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



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

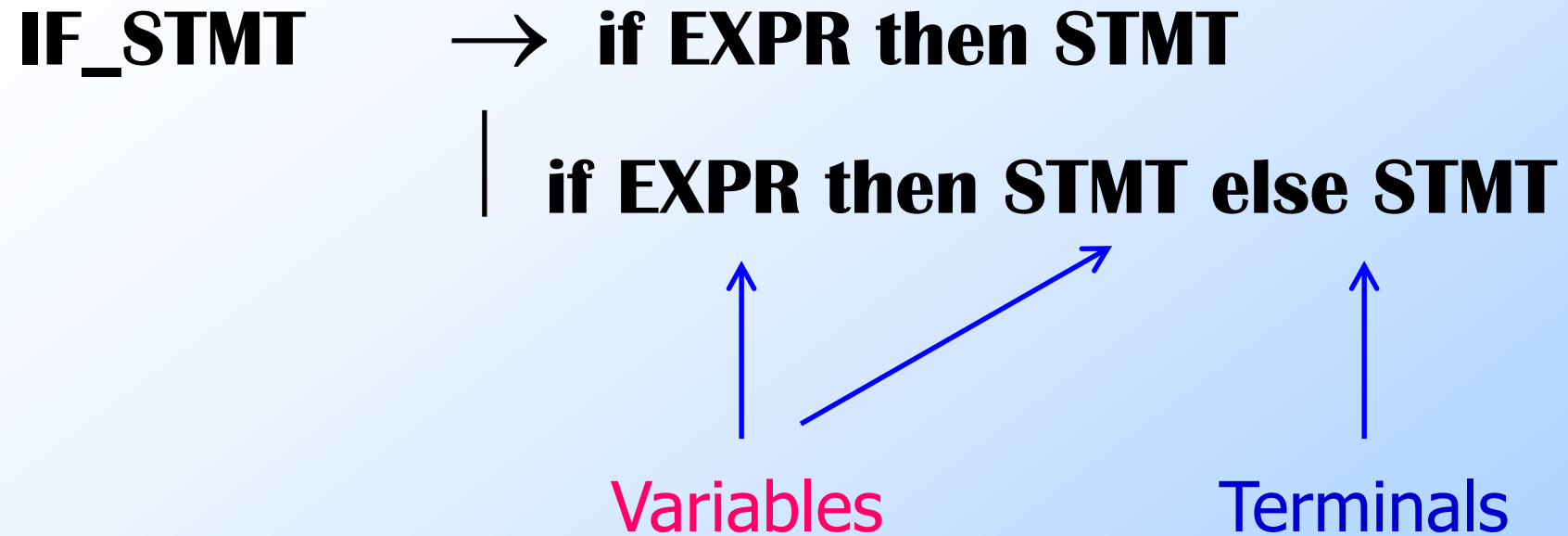
this grammar is ambiguous also because

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

$$\begin{aligned} E &\Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E \\ &\Rightarrow a + a * E \Rightarrow a + a * a \end{aligned}$$

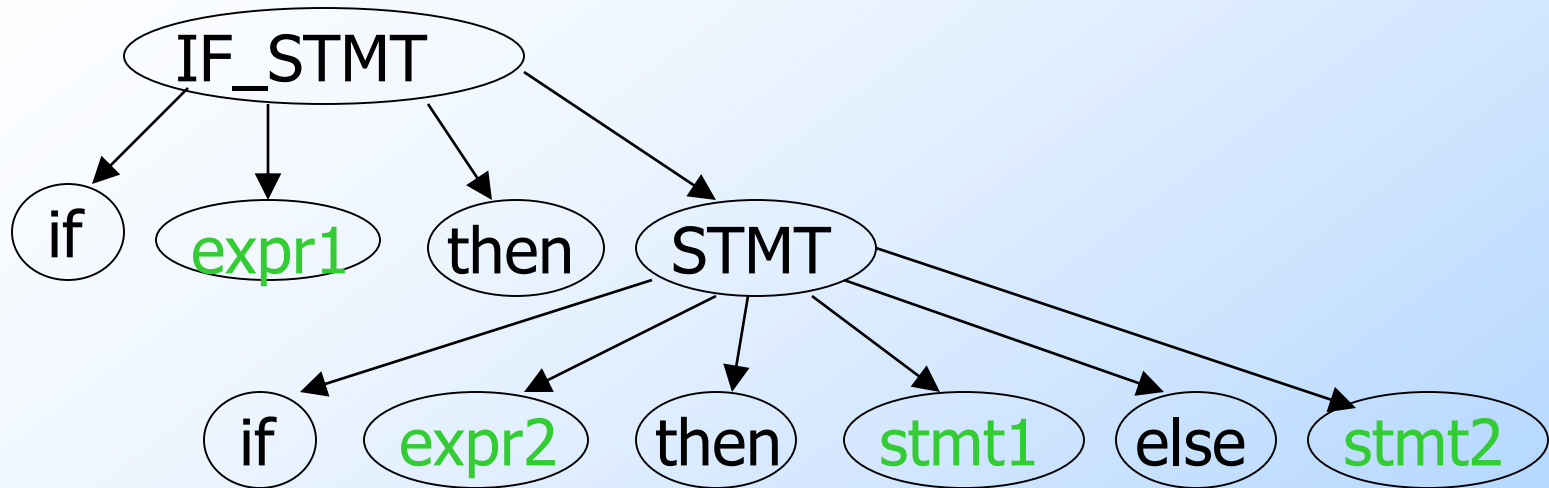
$$\begin{aligned} E &\Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E \\ &\Rightarrow a + a * E \Rightarrow a + a * a \end{aligned}$$

Another ambiguous grammar:

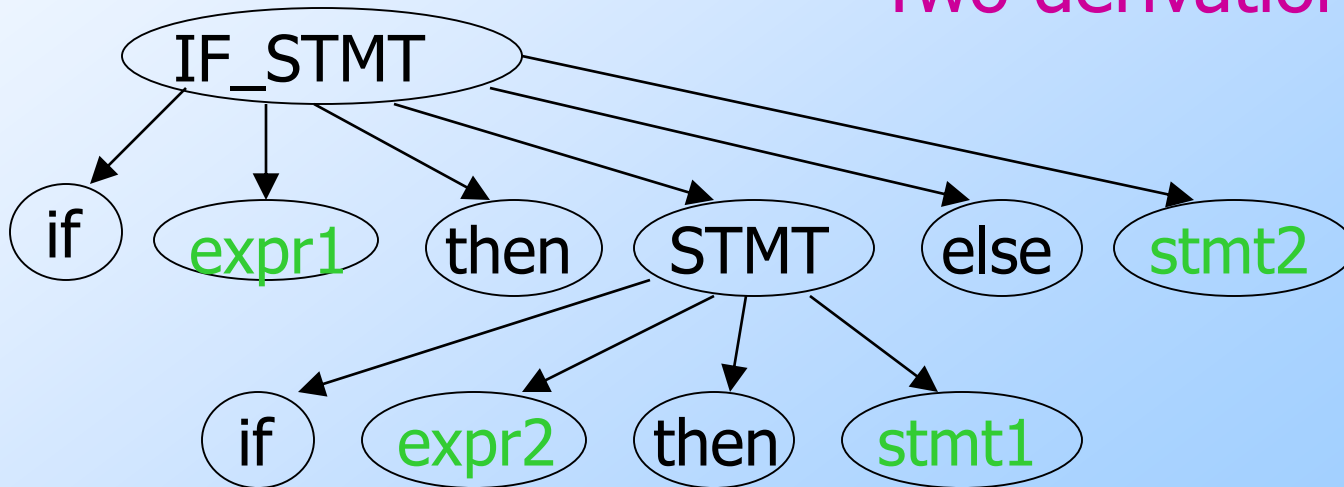


Very common piece of grammar
in programming languages

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 it is difficult to achieve this

A successful example:

Ambiguous
Grammar

$$E \rightarrow E + E$$

$$E \rightarrow E * E$$

$$E \rightarrow (E)$$

$$E \rightarrow a$$

Equivalent

Non-Ambiguous
Grammar

$$E \rightarrow E + T \mid T$$

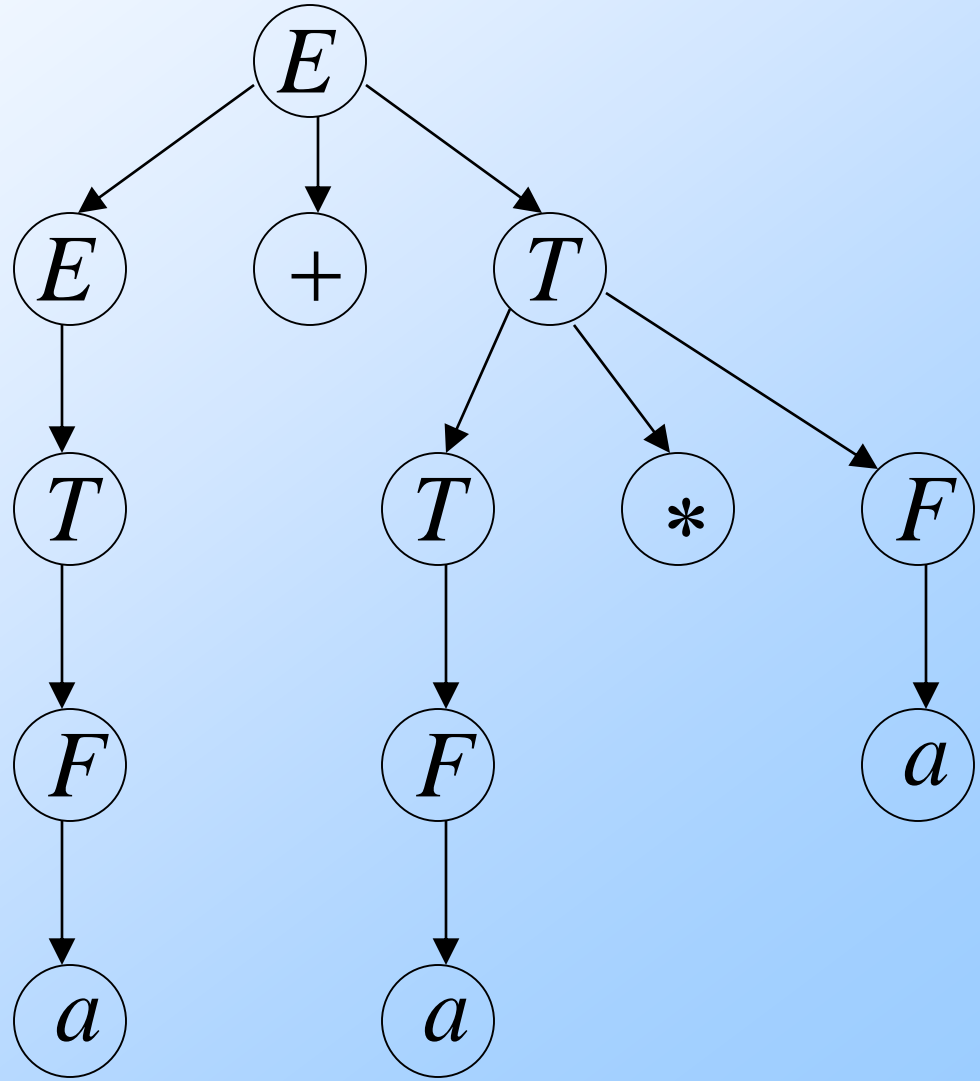
$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid a$$

generates the same
language

$$\begin{aligned}
 E &\Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow a + T * F \\
 &\Rightarrow a + F * F \Rightarrow a + a * F \Rightarrow a + a * a
 \end{aligned}$$

$$\begin{aligned}
 E &\rightarrow E + T \mid T \\
 T &\rightarrow T * F \mid F \\
 F &\rightarrow (E) \mid a
 \end{aligned}$$



Unique
 derivation tree for
 $a + a * a$

Ambiguous Grammar example

◆ Let us consider a CFG:

◆ CFG: $E \rightarrow I \mid E + E \mid E * E \mid (E)$
 $I \rightarrow a \mid B \mid Ia \mid Ib \mid IO \mid I1$

Expression: $a + a * a$

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

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

LMD

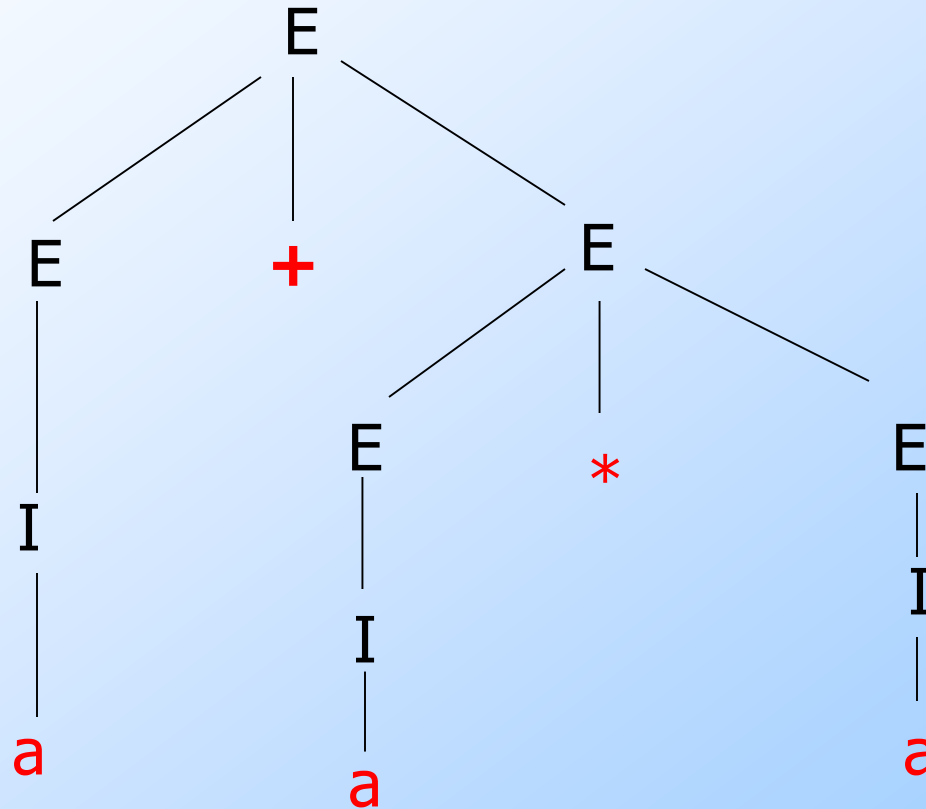


Fig: Trees yield $a+a*a$

RMD

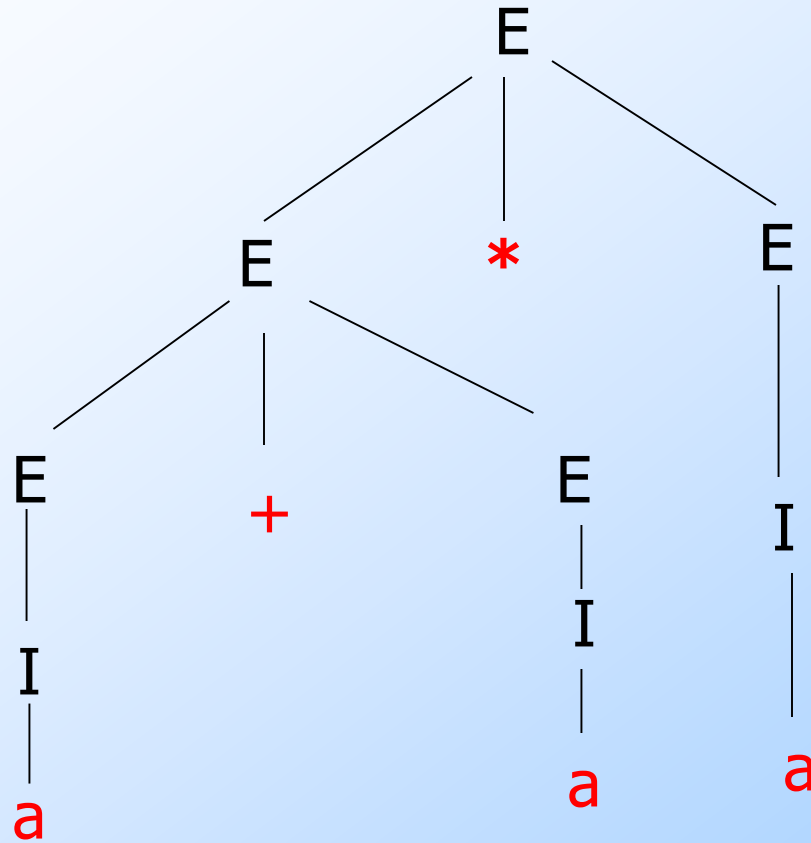


Fig: Trees yield $a+a*a$

Removing Ambiguity from Grammar

- ◆ 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 *factors*.
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 :

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

$F \rightarrow I \mid (E)$

$T \rightarrow F \mid T * F$

$E \rightarrow T \mid E + T$