

2021

Theory of Computation

Kun-Ta Chuang

**Department of Computer Science and Information Engineering
National Cheng Kung University**



Outline



Context-Free Grammars



Parsing and Ambiguity

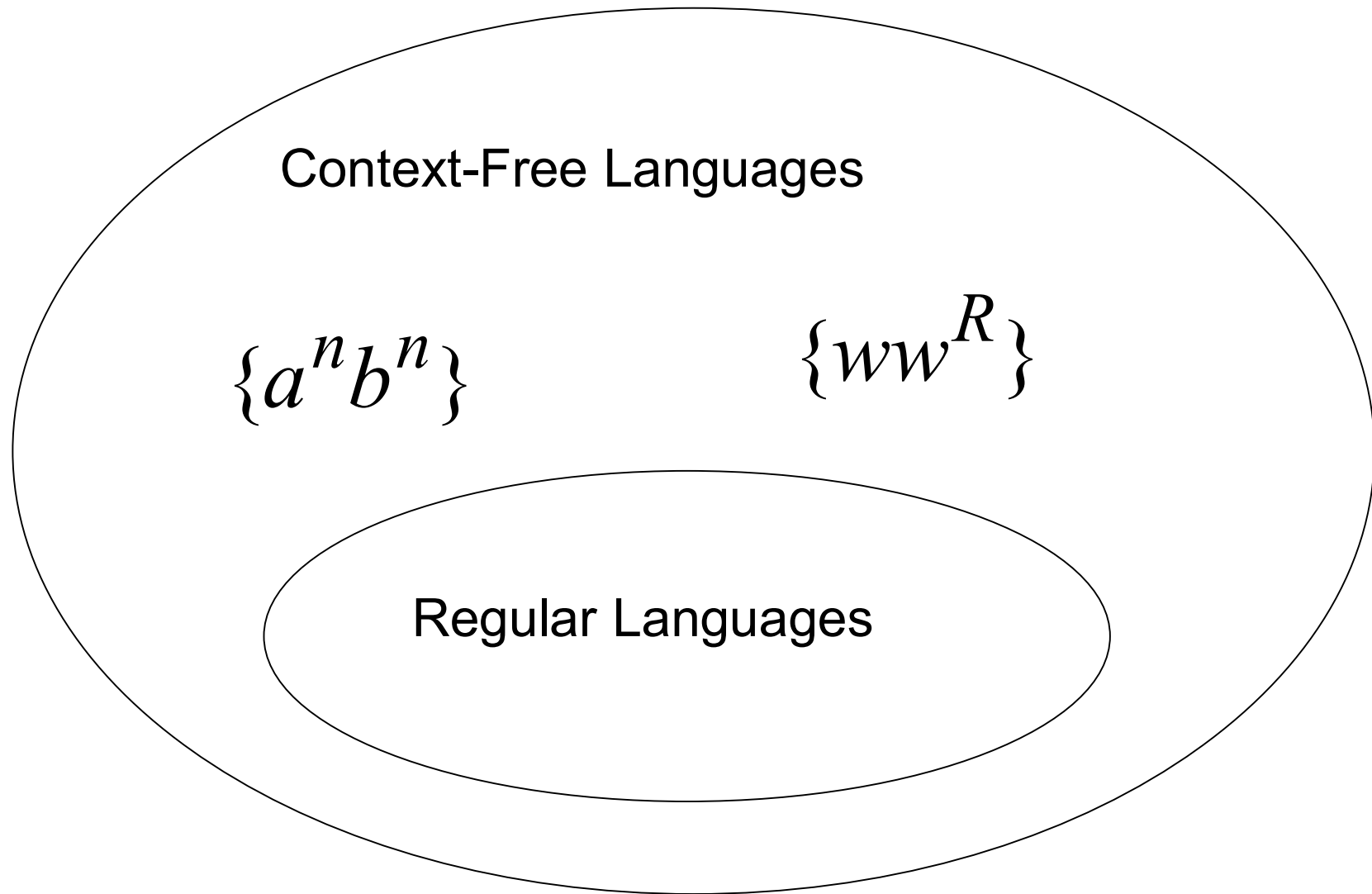


Context-Free Grammars and Programming Languages

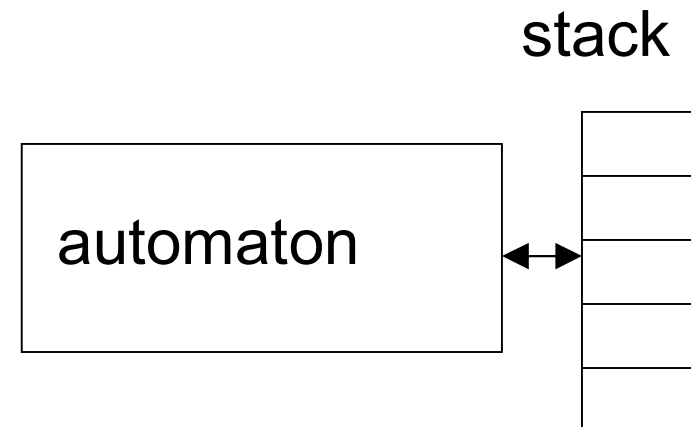
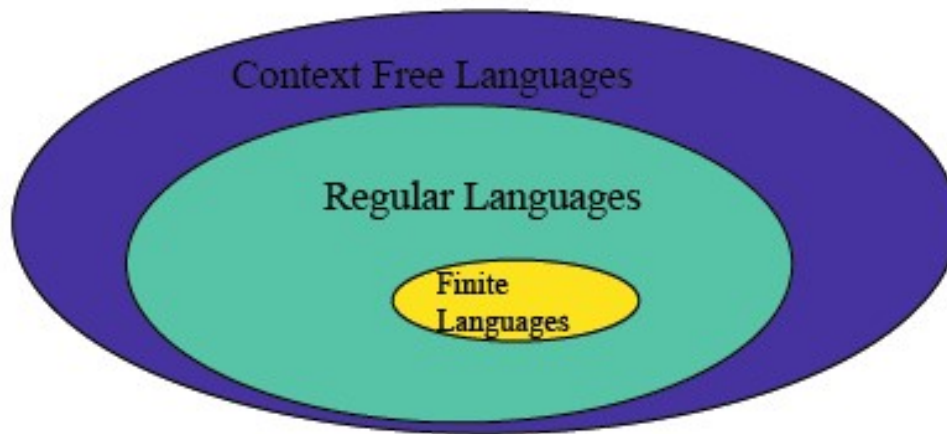
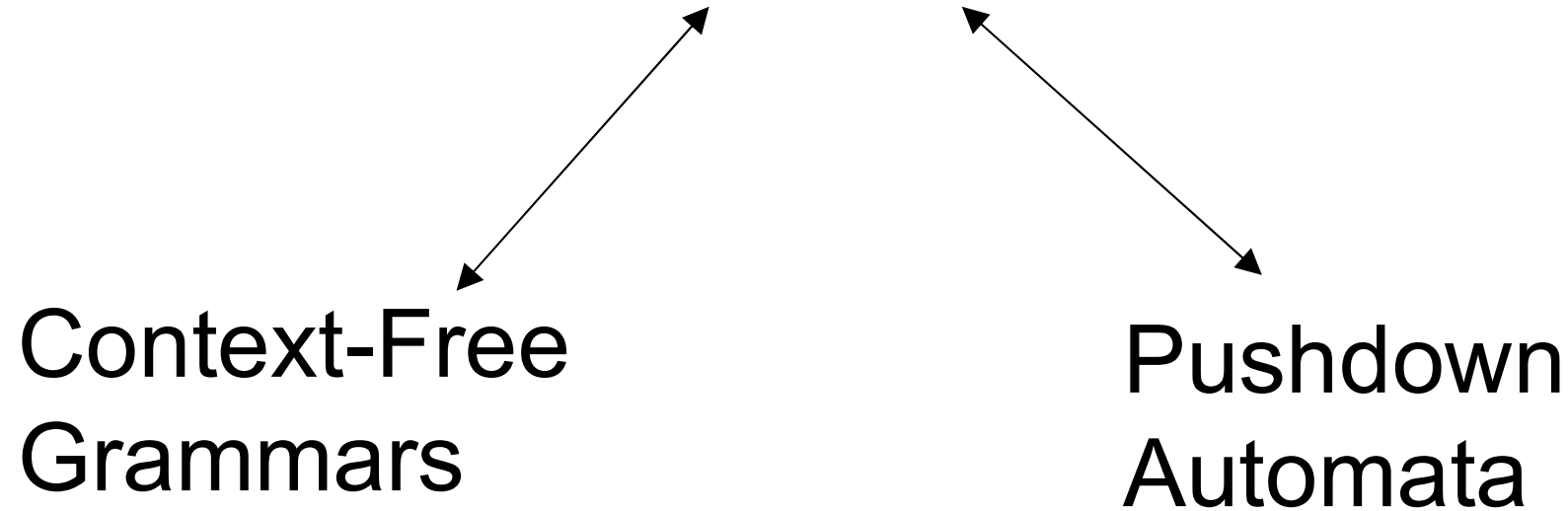
$$\{a^n b^n : n \geq 0\} \qquad \{ww^R\}$$

Regular Languages

$$a^* b^* \qquad (a + b)^*$$



Context-Free Languages



Context-Free Grammars

Regular Grammar

$$S \rightarrow abS$$

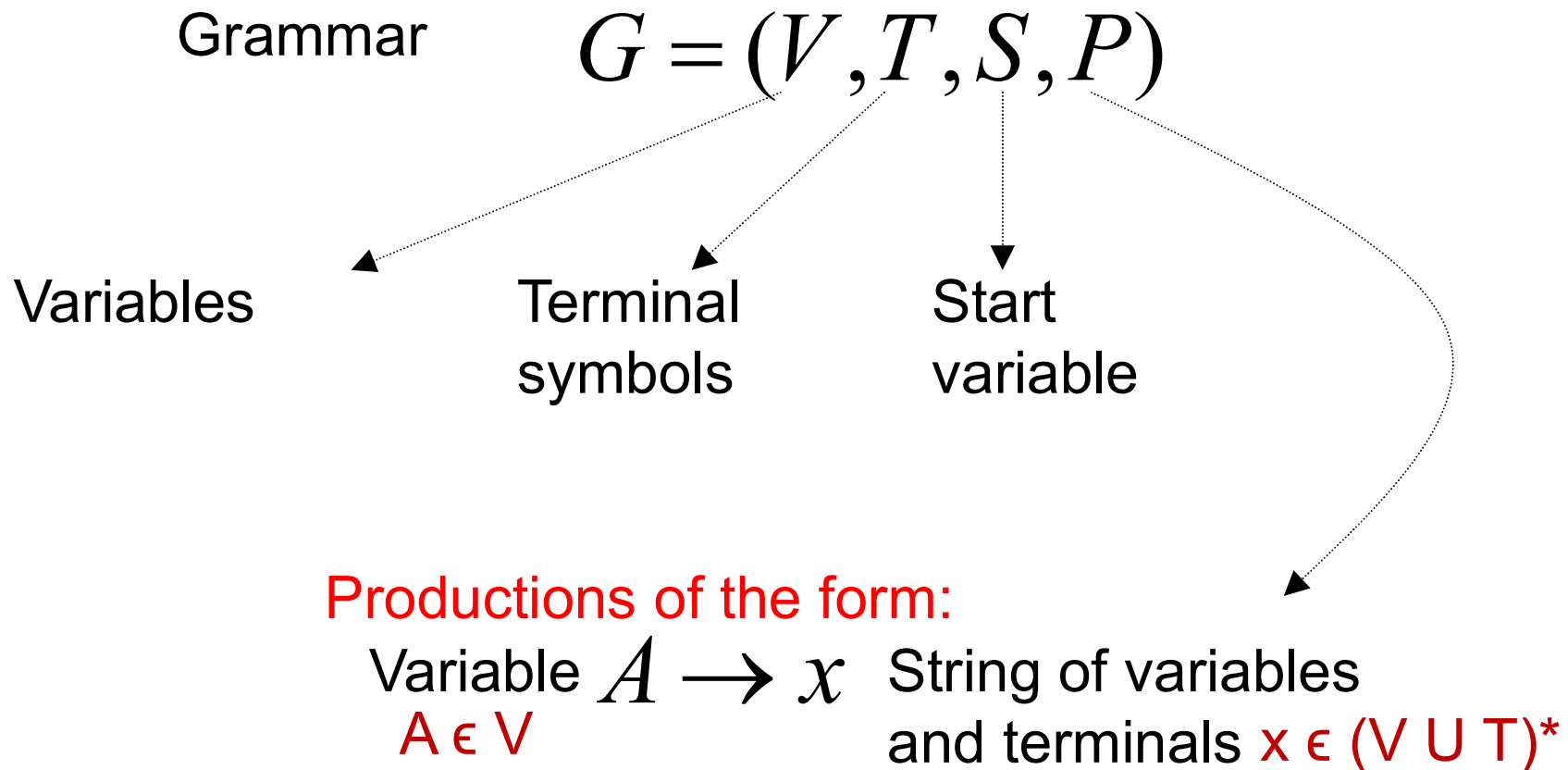
$$S \rightarrow a$$

$$S \rightarrow Aab$$

$$A \rightarrow Aab \mid B$$

$$B \rightarrow a$$

Definition 5.1: Context-Free Grammars



We say that the grammar is **context-free** since this substitution can take place regardless of where A is.

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

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

Regular and linear grammars are clearly context-free
But a context-free grammar is not necessarily linear

Definition: Context-Free Languages

A language L is context-free

if and only if

there is a context-free grammar G

with $L = L(G)$

Example

A context-free grammar : G

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

A derivation:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

Example

A context-free grammar : G

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

Another derivation:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbbb$$

$$S \rightarrow aSb$$

$$S \rightarrow \lambda$$

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

Describes parentheses: (((()))

Example 5.1

A context-free grammar: G

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

A derivation:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abba$$

A context-free grammar: G

$$\begin{aligned} S &\rightarrow aSa \\ S &\rightarrow bSb \\ S &\rightarrow \lambda \end{aligned}$$

Another derivation:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abaSaba \Rightarrow abaaba$$

$$S \rightarrow aSa$$

$$S \rightarrow bSb$$

$$S \rightarrow \lambda$$

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

This language is context-free, but it is not regular

Example 5.2

A context-free grammar: G

$$\begin{aligned} S &\rightarrow abB \\ A &\rightarrow aaBb \\ B &\rightarrow bbAa \\ A &\rightarrow \lambda \end{aligned}$$

A derivation:

$$\begin{aligned} S &\Rightarrow abB \Rightarrow abbbAa \Rightarrow abbbaaBba \\ &\Rightarrow abbbbaabbAaba \Rightarrow abbbbaabbaba \end{aligned}$$

$$S \rightarrow abB$$

$$A \rightarrow aaBb$$

$$B \rightarrow bbAa$$

$$A \rightarrow \lambda$$

$$L(G) = \{ab(bbaa)^n bba(ba)^n : n \geq 0\}$$

The above grammars are not only context-free, but linear.

Regular and linear grammars are context-free,

But a context-free grammar is not necessarily linear.

Example 5.3

The language $L = \{a^n b^m : n \neq m\}$ is context-free

$$\begin{array}{l} S \rightarrow aSb \\ S \rightarrow \lambda \end{array} \quad \Rightarrow \quad \begin{array}{l} S \rightarrow AS_1 \\ S_1 \rightarrow aS_1b \mid \lambda \\ A \rightarrow aA \mid a \end{array} \quad \Rightarrow \quad \begin{array}{l} S \rightarrow AS_1 \mid S_1B \\ S_1 \rightarrow aS_1b \mid \lambda \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid b \end{array}$$

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

$$n > m$$

$$n < m$$

The resulting grammar is context-free but not **linear**

Example 5.4

A context-free grammar: G $S \rightarrow aSb$

$$S \rightarrow SS$$

$$S \rightarrow \lambda$$

Derivations:

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

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

A context-free grammar: $G \quad S \rightarrow aSb$

$$S \rightarrow SS$$

$$S \rightarrow \lambda$$

More derivations:

$$\begin{aligned} S &\Rightarrow SS \Rightarrow aSbS \Rightarrow abS \Rightarrow abaSb \\ &\Rightarrow abaaSbb \Rightarrow abaabb \end{aligned}$$

$$\begin{aligned} S &\Rightarrow aSb \Rightarrow aSSb \Rightarrow aaSbSb \\ &\Rightarrow aabSb \Rightarrow aabaSbb \Rightarrow aababb \end{aligned}$$

$$S \rightarrow aSb$$

$$S \rightarrow SS$$

$$S \rightarrow \lambda$$

$$L(G) = \{w : n_a(w) = n_b(w),$$

and $n_a(v) \geq n_b(v)$

in any prefix $v\}$

Describes
matched
parentheses:

$()$ $()()$ $()(())$ $((()))$

Derivation Order

In CFGs that are not linear, a derivation may involve sentential forms with more than one variables.

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

$$L(G) = \{a^{2n}b^m: n \geq 0, m \geq 0\}$$

Derivation Order

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

Leftmost derivation:

$$S \xRightarrow{1} \boxed{A}B \xRightarrow{2} aa\boxed{A}B \xRightarrow{3} aa\boxed{B} \xRightarrow{4} aa\boxed{B}b \xRightarrow{5} aab$$

Rightmost derivation:

$$S \xRightarrow{1} A\boxed{B} \xRightarrow{4} A\boxed{B}b \xRightarrow{5} \boxed{A}b \xRightarrow{2} aa\boxed{A}b \xRightarrow{3} aab$$

Example 5.5

$$1. S \rightarrow aAB \quad 2. A \rightarrow bBb$$

$$3. B \rightarrow A \quad 4. B \rightarrow \lambda$$

Leftmost derivation:

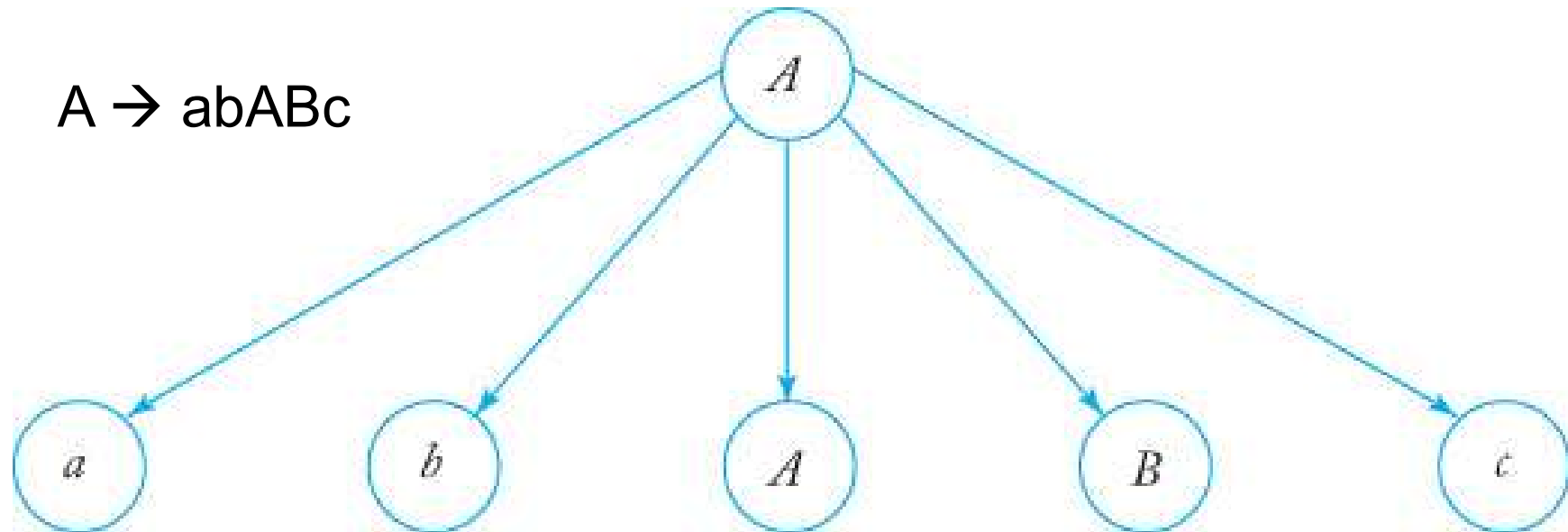
$$\begin{aligned} S &\xRightarrow{1} aAB \xRightarrow{2} abBbB \xRightarrow{3} abAbB \xRightarrow{2} abbBbbB \\ &\xRightarrow{4} abbbbB \xRightarrow{4} abbbb \end{aligned}$$

Rightmost derivation:

$$\begin{aligned} S &\xRightarrow{1} aAB \xRightarrow{4} aA \xRightarrow{2} abBb \xRightarrow{3} abAb \\ &\xRightarrow{2} abbBbb \xRightarrow{4} abbbb \end{aligned}$$

Derivation (Parse) Trees

An ordered tree in which **nodes** are labeled with the **left sides of productions** and in which the **children** of a node represent its corresponding **right sides**.



Definition 5.3

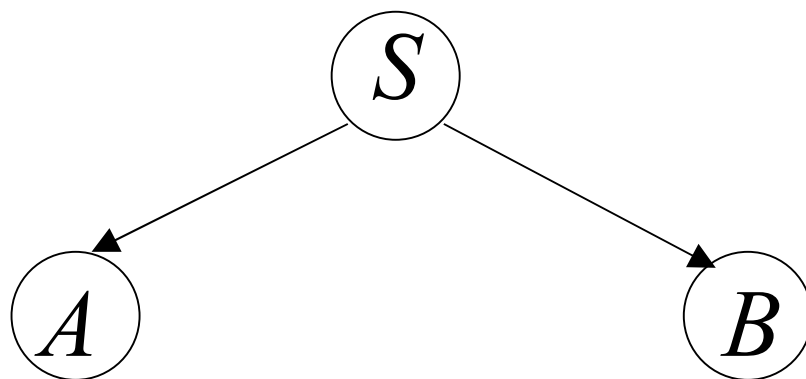
- Let $G = (V, T, S, P)$ be a CFG. An ordered tree is a **derivation tree** for G **iff** it has the following properties.
 1. The **root** is labeled **S**
 2. Every **leaf** has a label from **$T \cup \{\lambda\}$**
 3. Every **internal vertex** has a label from **V**
 4. If a vertex has label $A \in V$, and its children are labeled a_1, a_2, \dots, a_n , then P must contain a production
$$A \rightarrow a_1 a_2 \dots a_n$$
 5. A leaf labeled λ has no sibling

$$S \rightarrow AB$$

$$A \rightarrow aaaA \mid \lambda$$

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

$$S \Rightarrow AB$$

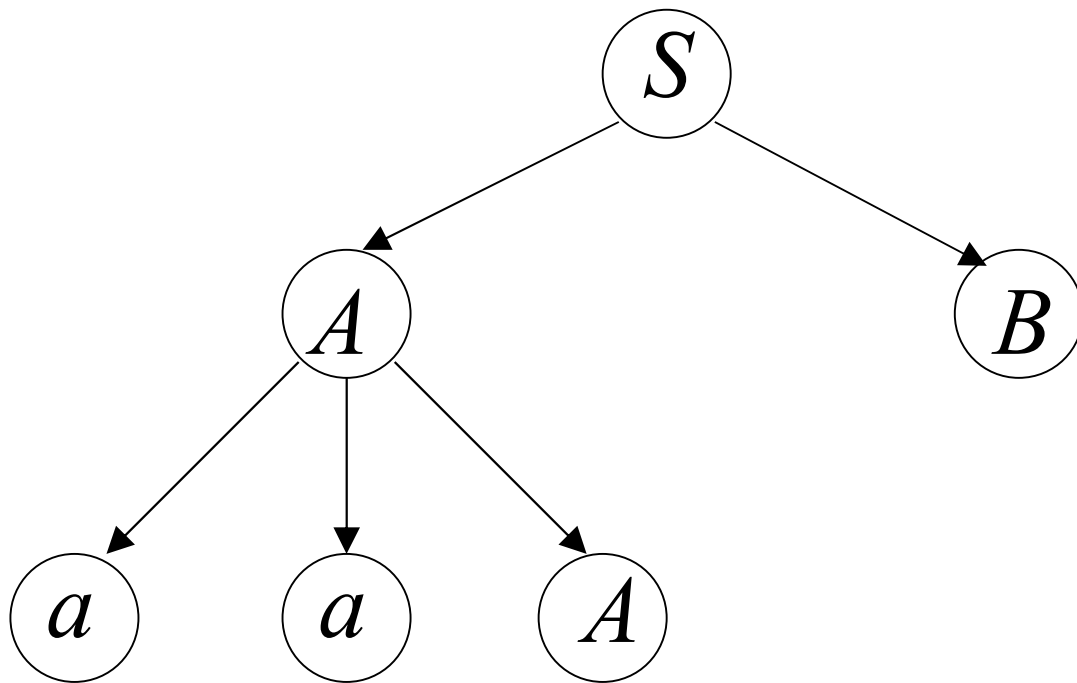


$$S \rightarrow AB$$

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

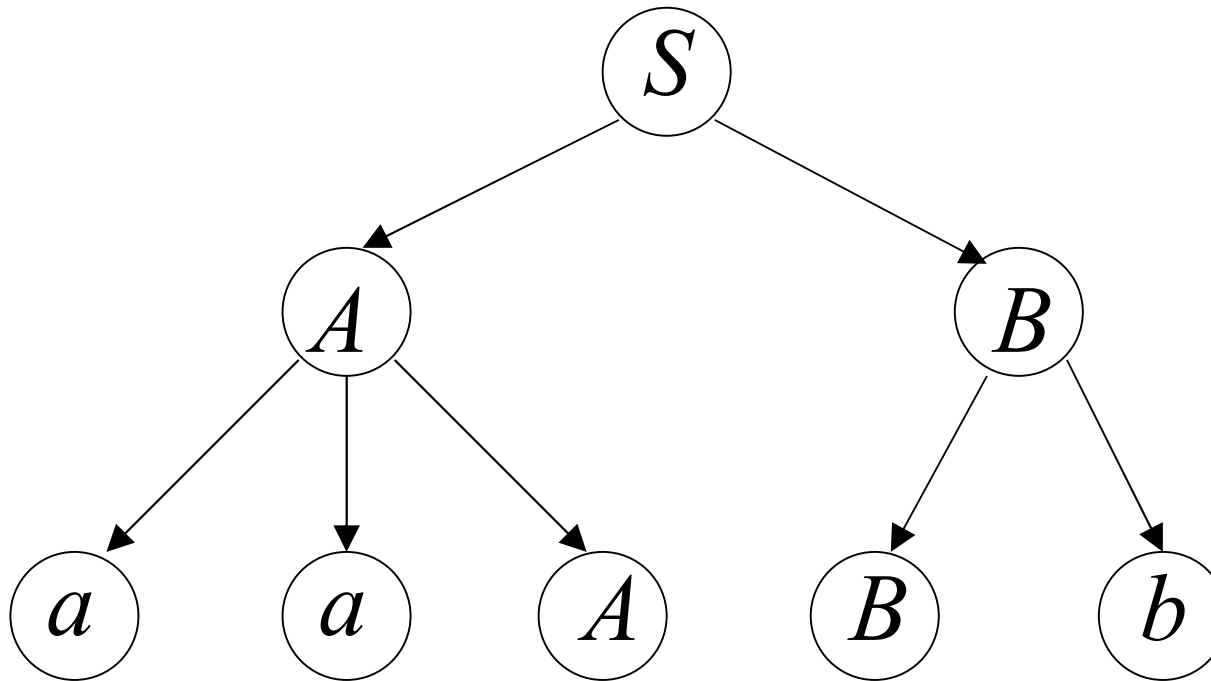
$$B \rightarrow Bb \mid \lambda$$

$$S \Rightarrow AB \Rightarrow aaAB$$



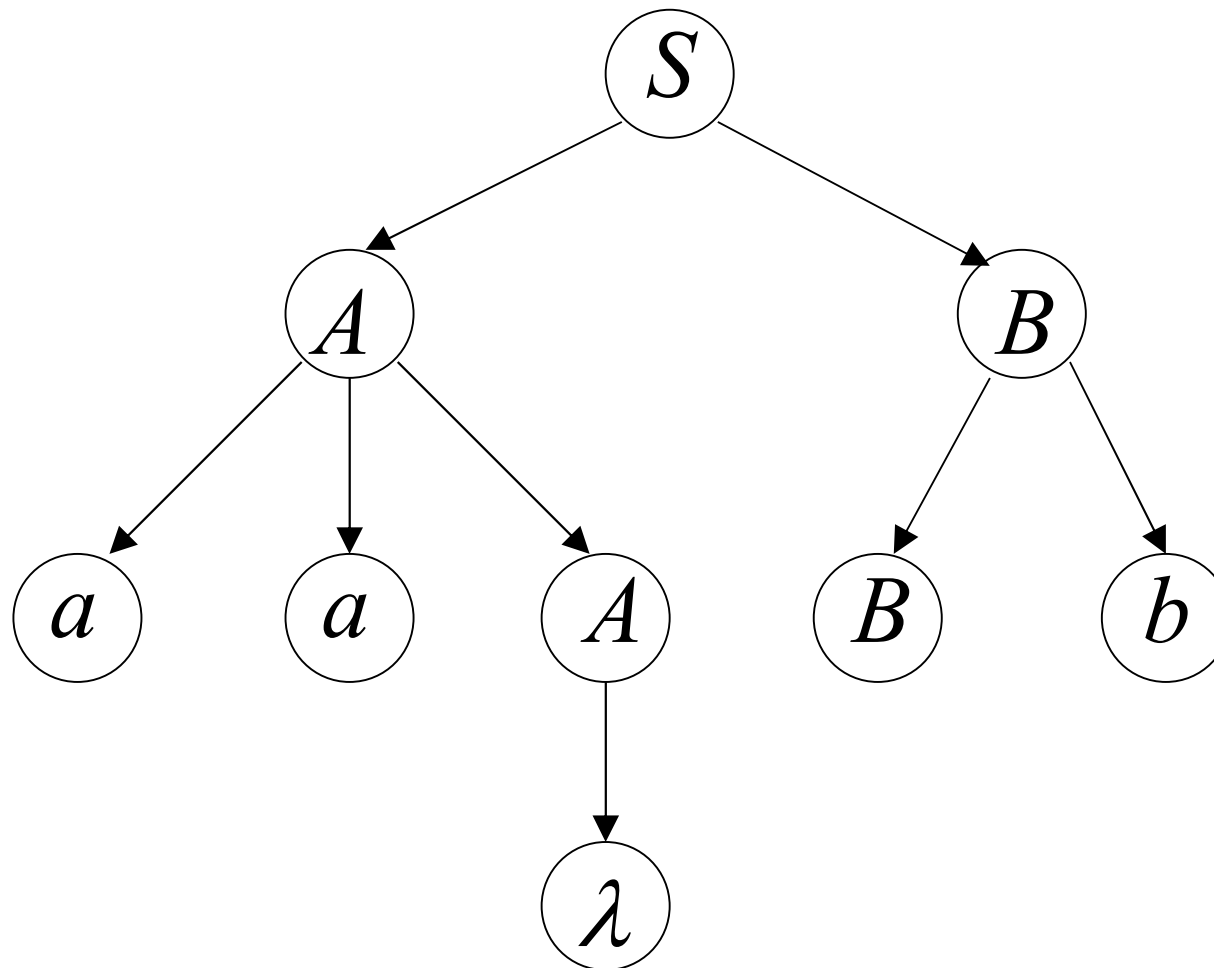
$$S \rightarrow AB \qquad A \rightarrow aaA \mid \lambda \qquad B \rightarrow Bb \mid \lambda$$

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



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

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

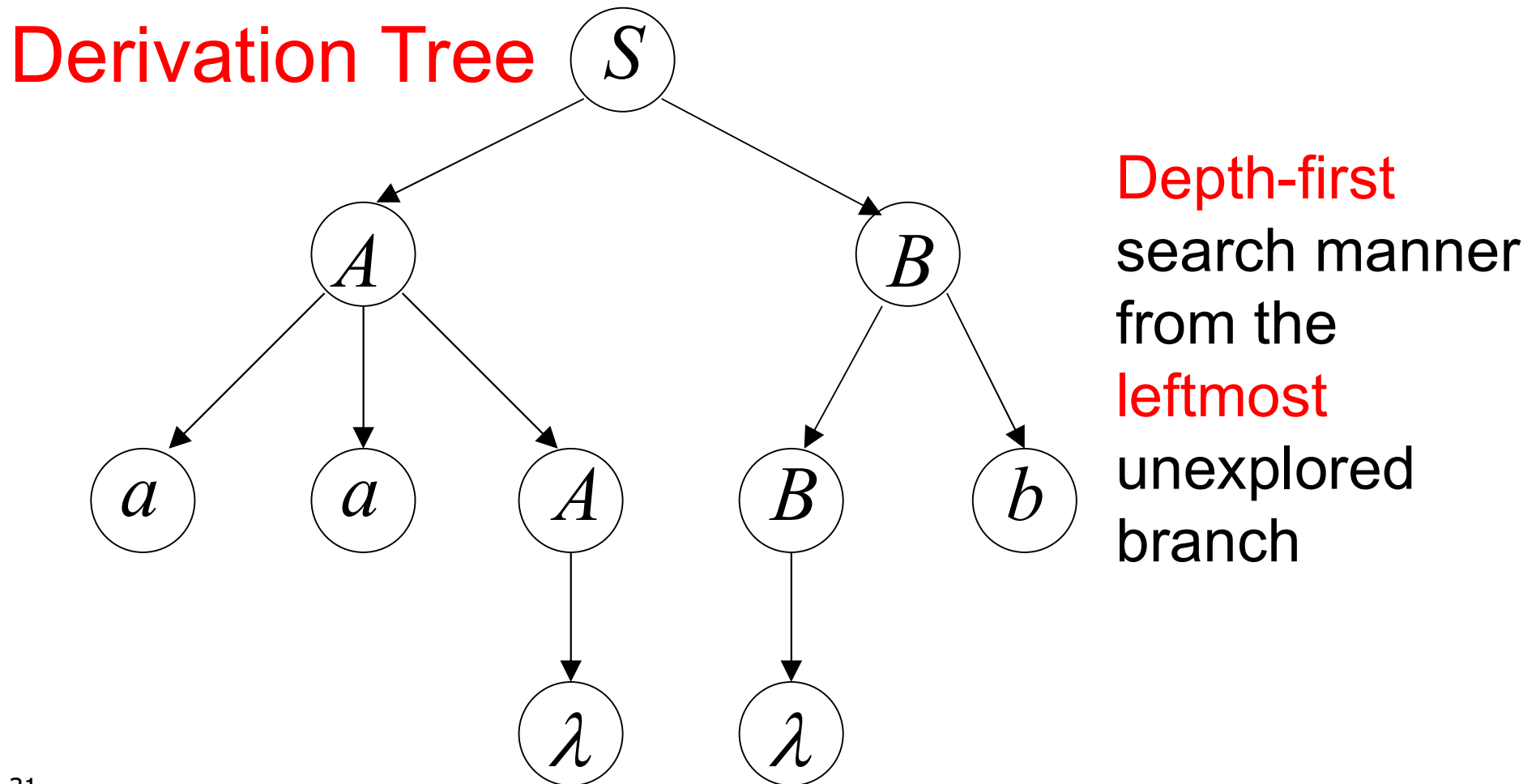


$$S \rightarrow AB$$

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

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

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



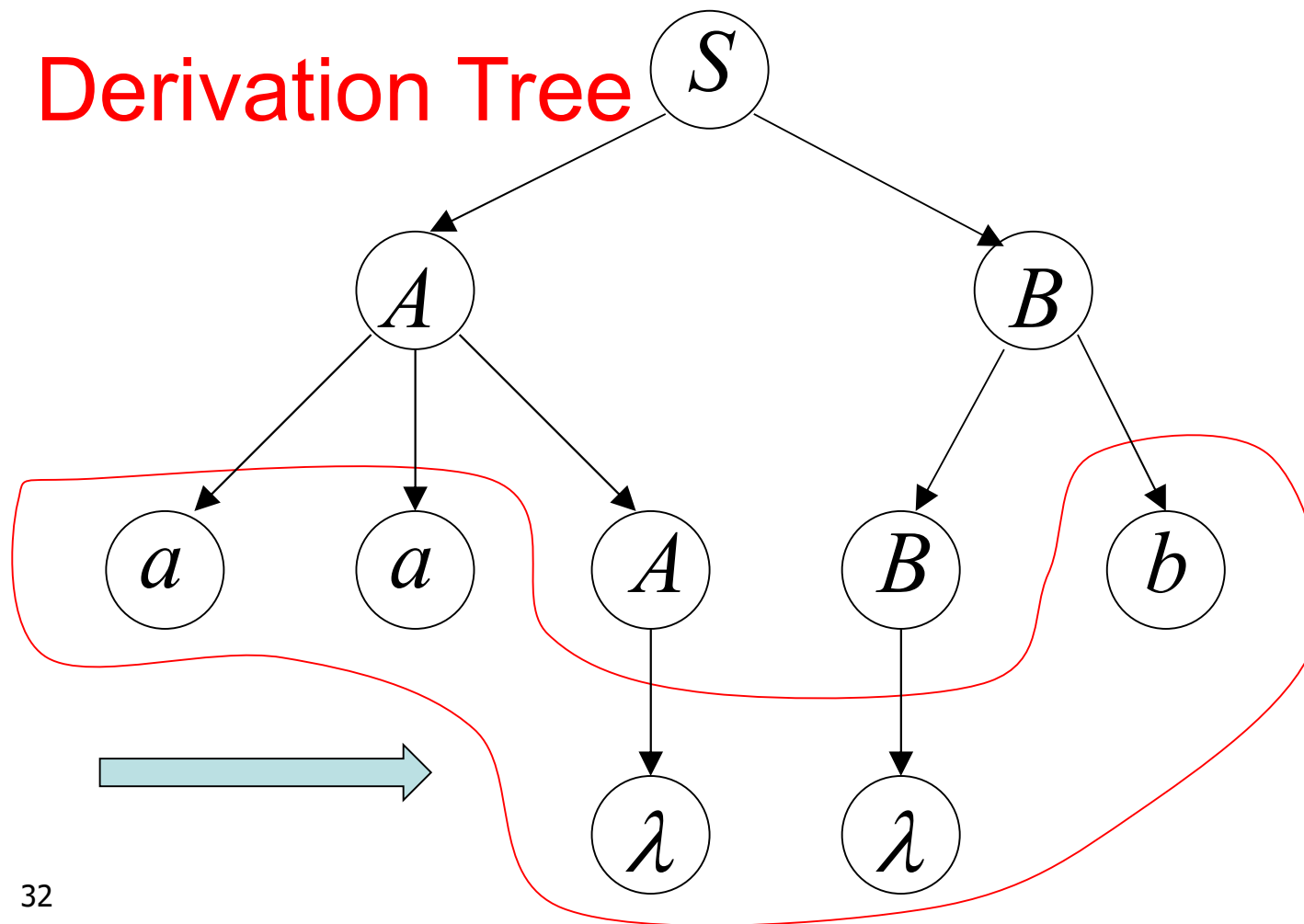
$$S \rightarrow AB$$

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

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

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

Derivation Tree



Yield

$$aa\lambda\lambda b$$

$$= aab$$

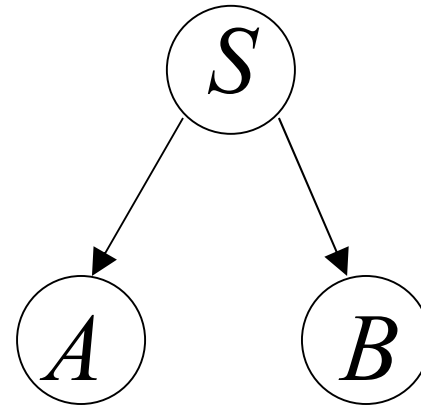
Partial Derivation Trees

$$S \rightarrow AB$$

$$S \Rightarrow AB$$

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

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

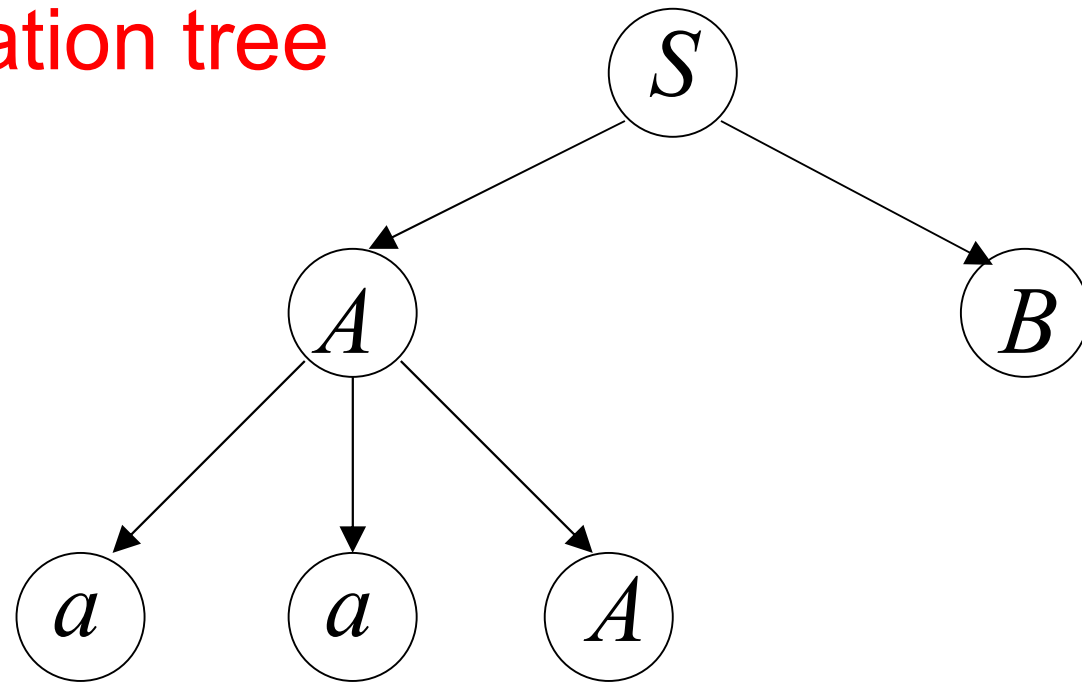


Partial derivation tree

- A tree that has properties 3, 4, and 5.
- 1 does not necessarily hold.
- 2 is replaced by:
 - Every **leaf** has a label from $V \cup T \cup \{\lambda\}$

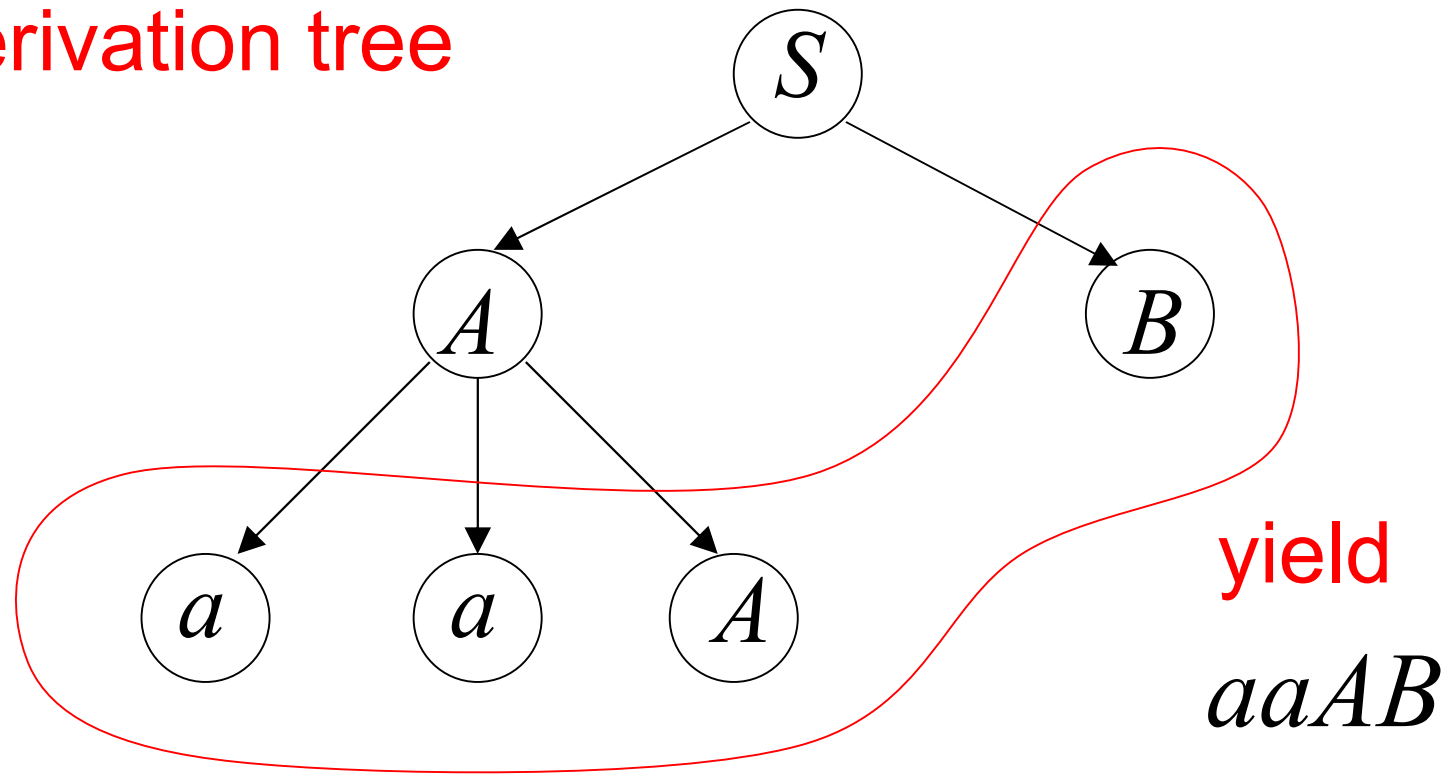
$$S \Rightarrow AB \Rightarrow aaAB$$

Partial derivation tree



$S \Rightarrow AB \Rightarrow aaAB$ sentential
form

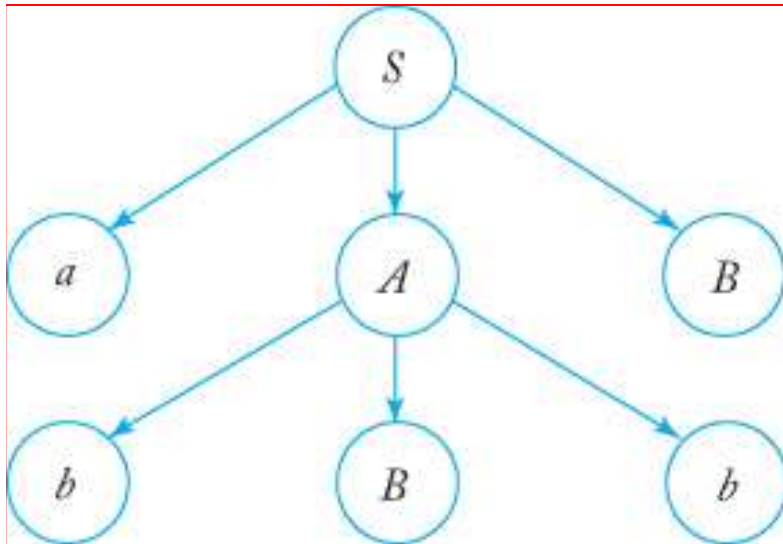
Partial derivation tree



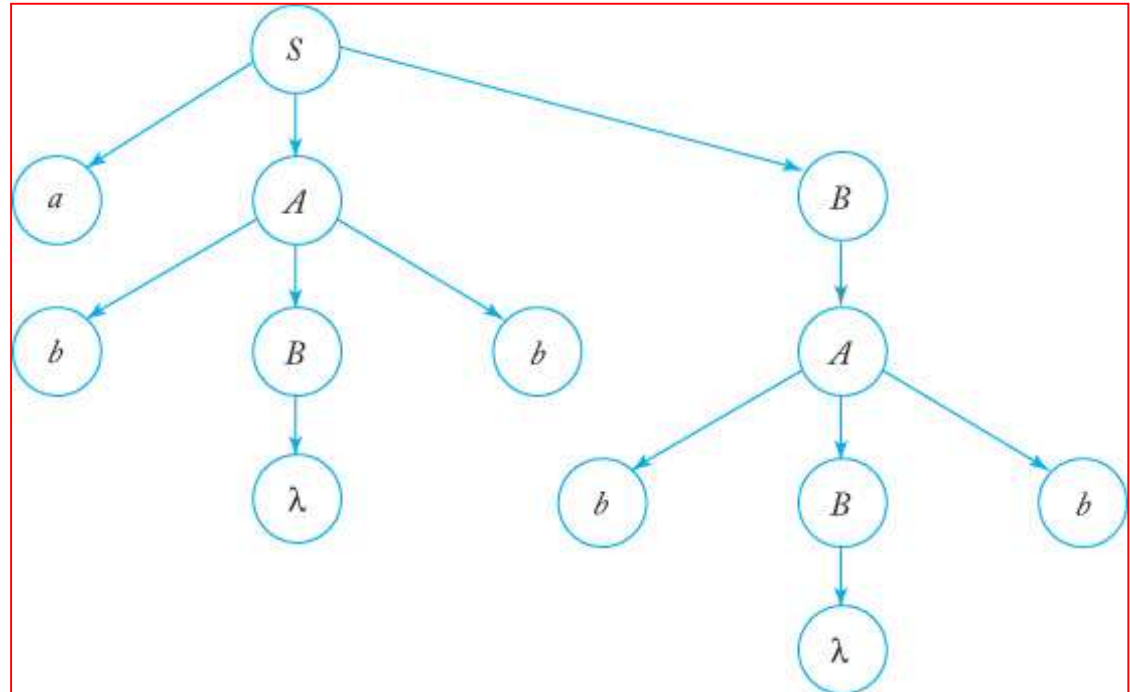
Example 5.6

$$S \rightarrow aAB \quad A \rightarrow bBb \quad B \rightarrow A \mid \lambda$$

Yield: abBbbB is a sentential form of G
 $abbbb \in L(G)$



Partial derivation tree



Derivation tree

Theorem 5.1

- Let $G = (V, T, S, P)$ be a CFG. Then for every $w \in L(G)$, there exists a **derivation tree of G whose yield is w** . Conversely, the yield of any derivation tree is in $L(G)$.
- Also, if t_G is any partial derivation tree for G whose root is labeled S , then the yield of t_G is a **sentential form** of G .

Sometimes, derivation order doesn't matter

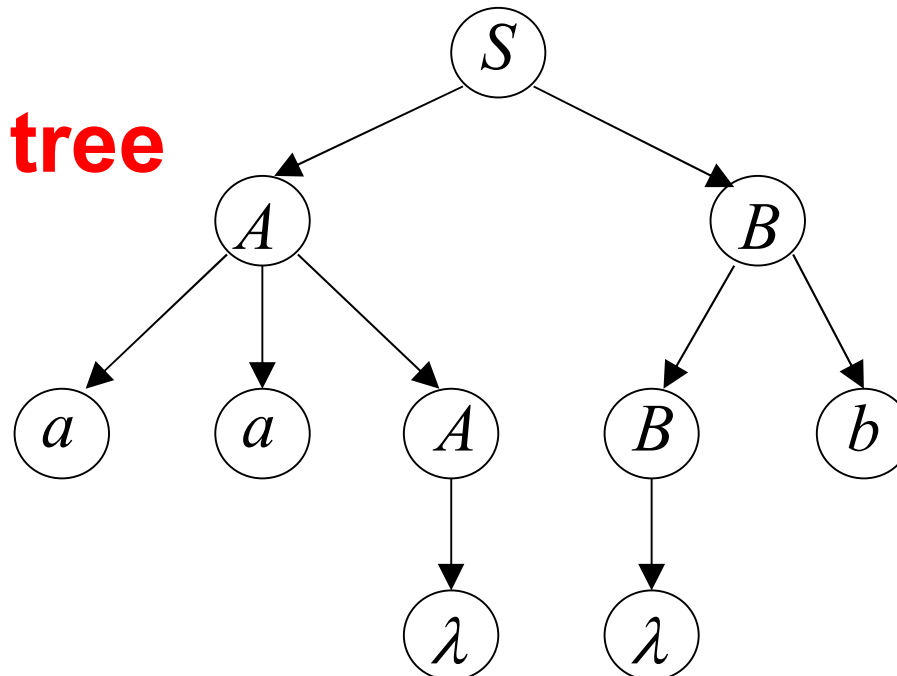
Leftmost:

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

Rightmost:

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

Same derivation tree



Outline



Context-Free Grammars



Parsing and Ambiguity



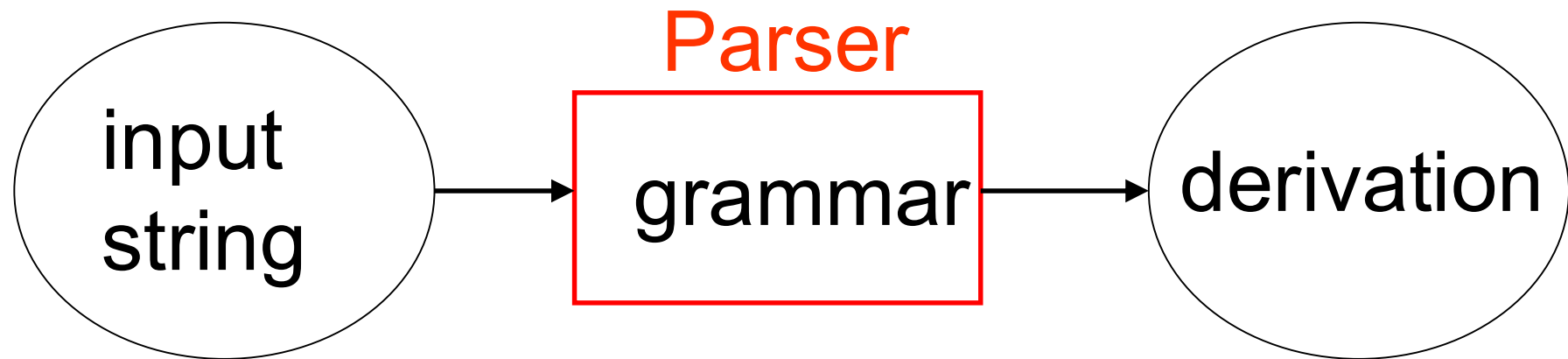
Context-Free Grammars and Programming Languages

Parsing

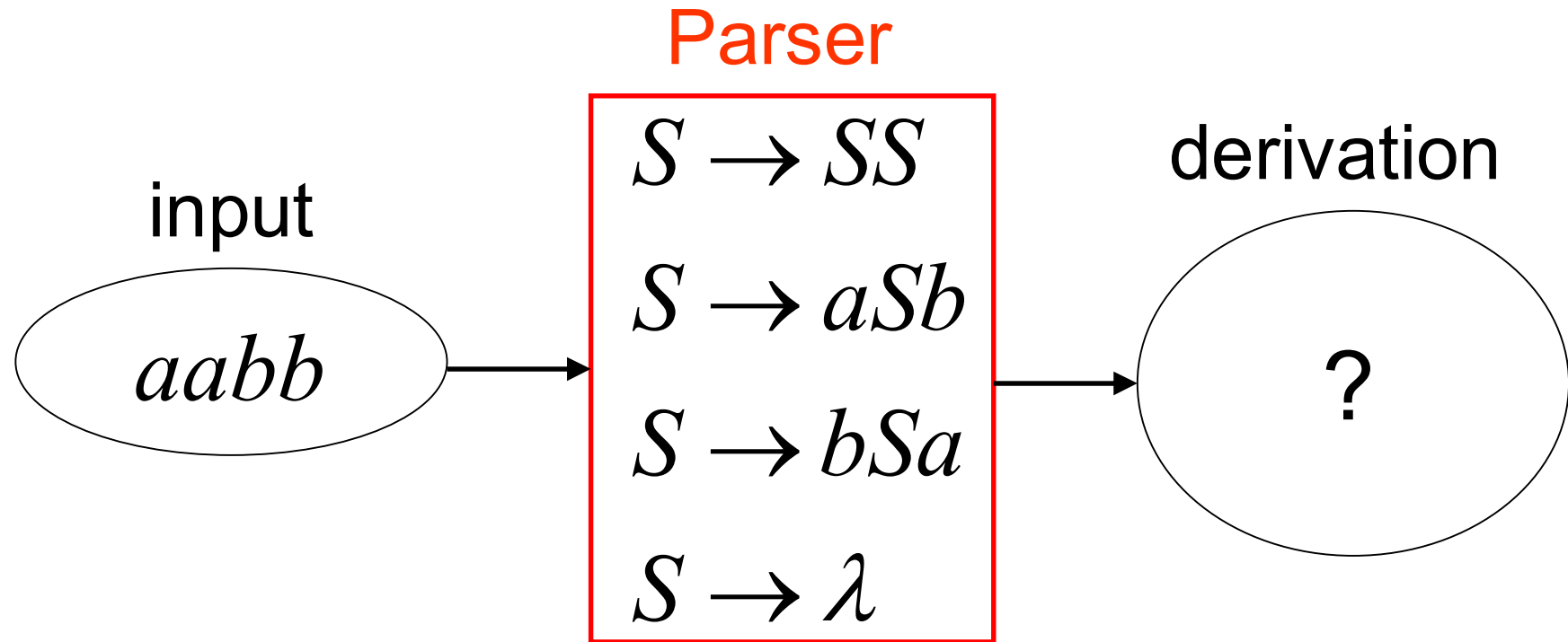
Given a grammar G , we studied the set of strings that can be derived using G , but...

Given a string w of terminals, we want to know whether or not w is in $L(G)$
(membership question)

◆ **Parsing** describes finding a sequence of productions by which a $w \in L(G)$ is derived.



Example:



**Exhaustive Search Parsing
(Brute Force Parsing)**

Example 5.7

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

Find derivation of
 $aabb$

Phase 1:

$$S \Rightarrow SS$$

$$S \Rightarrow aSb$$

~~$$S \Rightarrow bSa$$~~

~~$$S \Rightarrow \lambda$$~~

Phase 2 $S \rightarrow SS \mid aSb \mid bSa \mid \lambda$

$S \Rightarrow SS \Rightarrow SSS$

$S \Rightarrow SS \Rightarrow aSbS$

$aabb$

Phase 1

~~$S \Rightarrow SS \Rightarrow bSaS$~~

$S \Rightarrow SS$

$S \Rightarrow SS \Rightarrow S$

$S \Rightarrow aSb$

$S \Rightarrow aSb \Rightarrow aSSb$

$S \Rightarrow aSb \Rightarrow aaSbb$

~~$S \Rightarrow aSb \Rightarrow abSab$~~

~~$S \Rightarrow aSb \Rightarrow ab$~~

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

Phase 2

$$S \Rightarrow SS \Rightarrow SSS$$

$$S \Rightarrow SS \Rightarrow aSbS$$

$$aabb$$

$$S \Rightarrow SS \Rightarrow S$$

$$S \Rightarrow aSb \Rightarrow aSSb$$

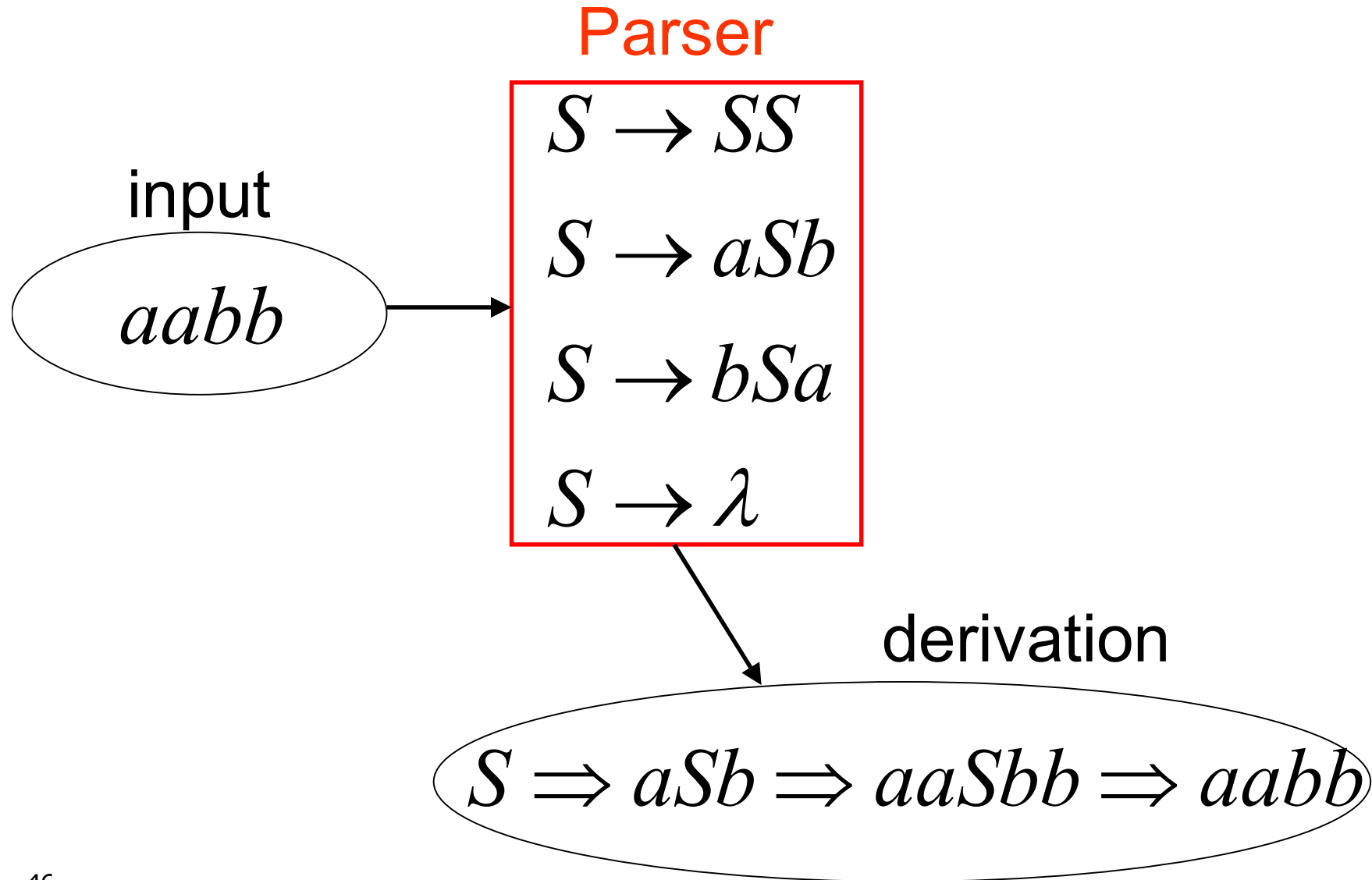
$$S \Rightarrow aSb \Rightarrow aaSbb$$

Phase 3



$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

Final result of exhaustive search (top-down parsing)



Flaws of Exhaustive Search Parsing

- ◆ Tediousness (bad for efficiency)
- ◆ It is possible that it never terminates for strings not in $L(G)$

$w=abb?$

$$S \rightarrow SS$$

$$S \rightarrow aSb$$

$$S \rightarrow bSa$$

$$S \rightarrow \lambda$$

$$A \rightarrow \lambda$$

$$A \rightarrow B$$

Example 5.8

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

by $S \rightarrow SS \mid aSb \mid bSa \mid ab \mid ba$

If so, given any $w \in \{a,b\}^+$,
the exhaustive search parsing will always
terminate in no more than $|w|$ rounds.

It is trivial because the length of the sentential
form grows by at least one symbol in each round

Theorem 5.2

Suppose a CFG does **not** have any rules of the form

$$A \rightarrow \lambda$$

$$A \rightarrow B$$

Then the exhaustive search parsing can be made into an algorithm which, for any $w \in \Sigma^*$, either produces a parsing of **w** or tells us that no parsing is possible

Theorem 5.2

Suppose a CFG does not have any rules of the form

$$A \rightarrow \lambda$$

$$A \rightarrow B$$

\therefore Neither the length of a sentential form nor the number of terminals can exceed $|w|$

\therefore Number of phases for string w can not more than: $2|w|$

$$\text{Ex: } S \rightarrow SS \mid aSb \mid bSa \mid ab \mid ba$$

For grammar with P rules

Phase 1:

we have no more than $|P|$ sentential forms

Phase 2:

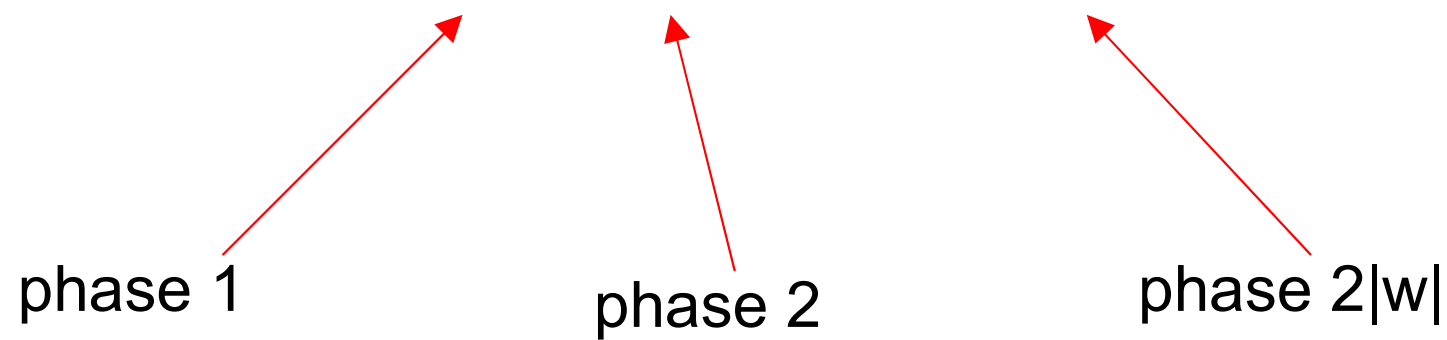
we have no more than $|P|^2$ sentential forms

⋮

Phase $2|w|$:

we have no more than $|P|^{2|w|}$ sentential forms

Total time needed for string : w

$$|P| + |P|^2 + \dots + |P|^{2^{|w|}}$$


phase 1 phase 2 phase $2^{|w|}$

$$O(P^{2^{|w|}+1})$$

Extremely bad!!!

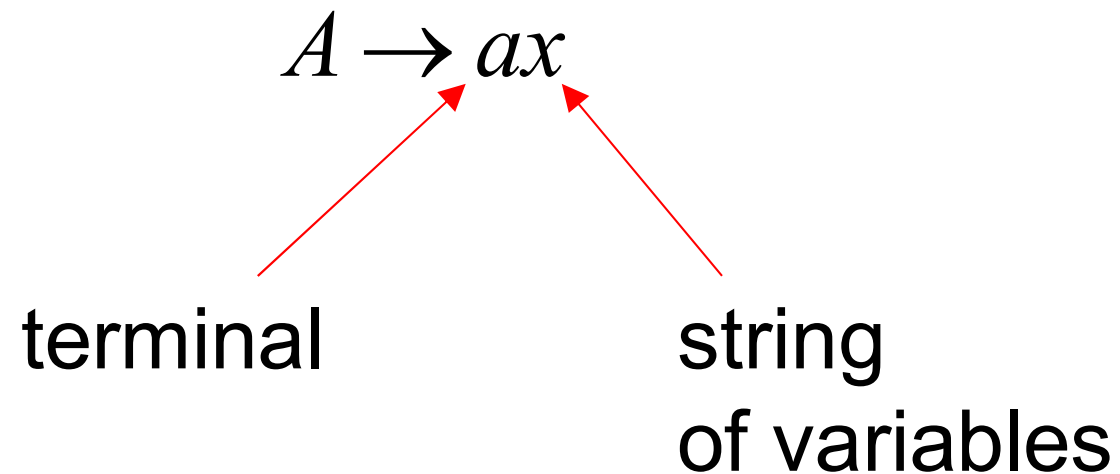
For general context-free grammars:

There exists a parsing algorithm
that parses a string $|w|$
in time $|w|^3$

The construction of more efficient parsing methods for CFGs is a complicated matter that belongs to a course on compilers

There exist faster algorithms
for specialized grammars

Simple grammar (s-grammar):



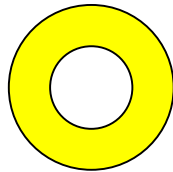
Pair (A, a) appears at most once in P

Example 5.9

$$S \rightarrow aS$$

$$S \rightarrow bSS$$

$$S \rightarrow c$$



$$S \rightarrow aS$$

$$S \rightarrow bSS$$

$$S \rightarrow aSS$$

$$S \rightarrow c$$



Each string has a unique derivation

$$S \Rightarrow aS \Rightarrow abSS \Rightarrow abcS \Rightarrow abcc$$

For S-grammars:

In the exhaustive search parsing
there is only one choice in each phase

Time for a phase: 1

Total time for parsing string w : $|w|$

Linear time!!!

$\langle \text{while_stmt} \rangle ::= \text{while } \langle \text{expr} \rangle \langle \text{stmt} \rangle$

Ambiguity

Example 5.11

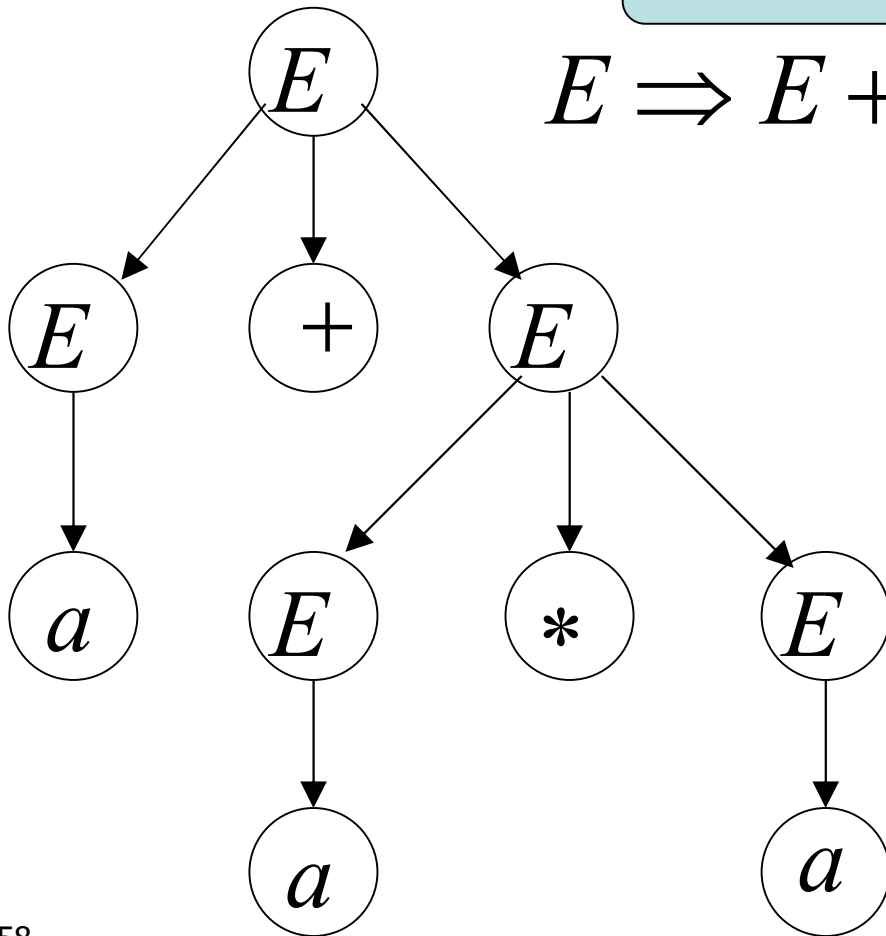
$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

$$a + a * a$$

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

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

leftmost derivation



Example 5.11

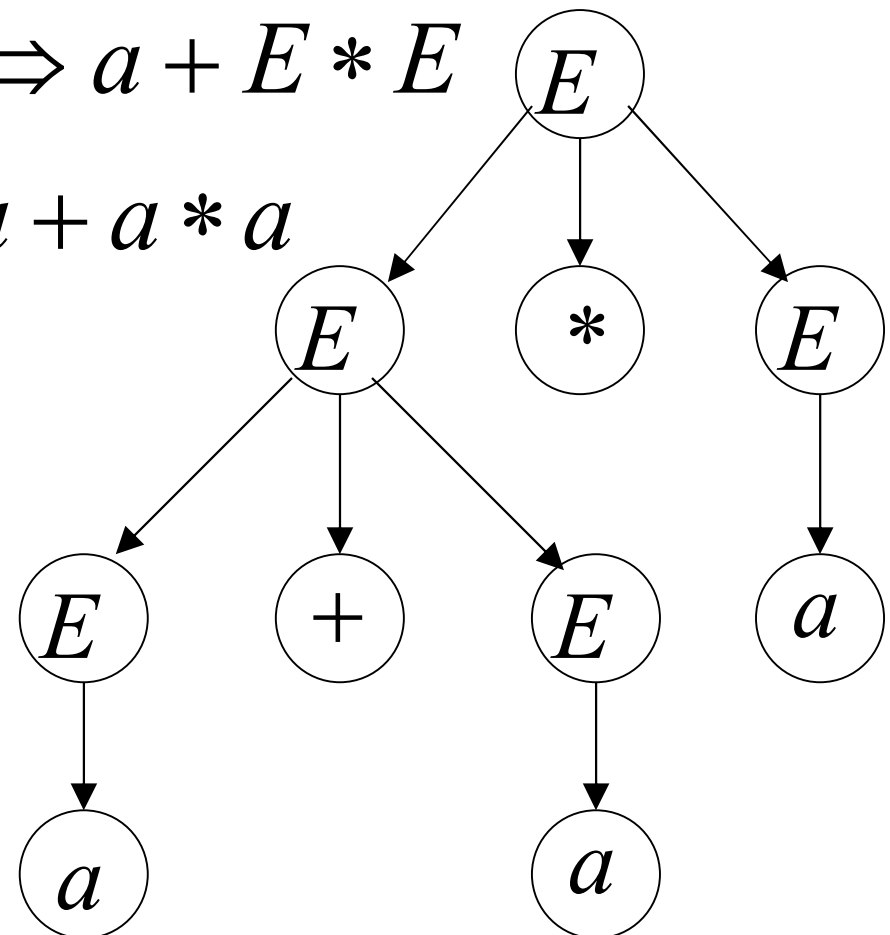
$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

$$a + a * a$$

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

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

leftmost derivation

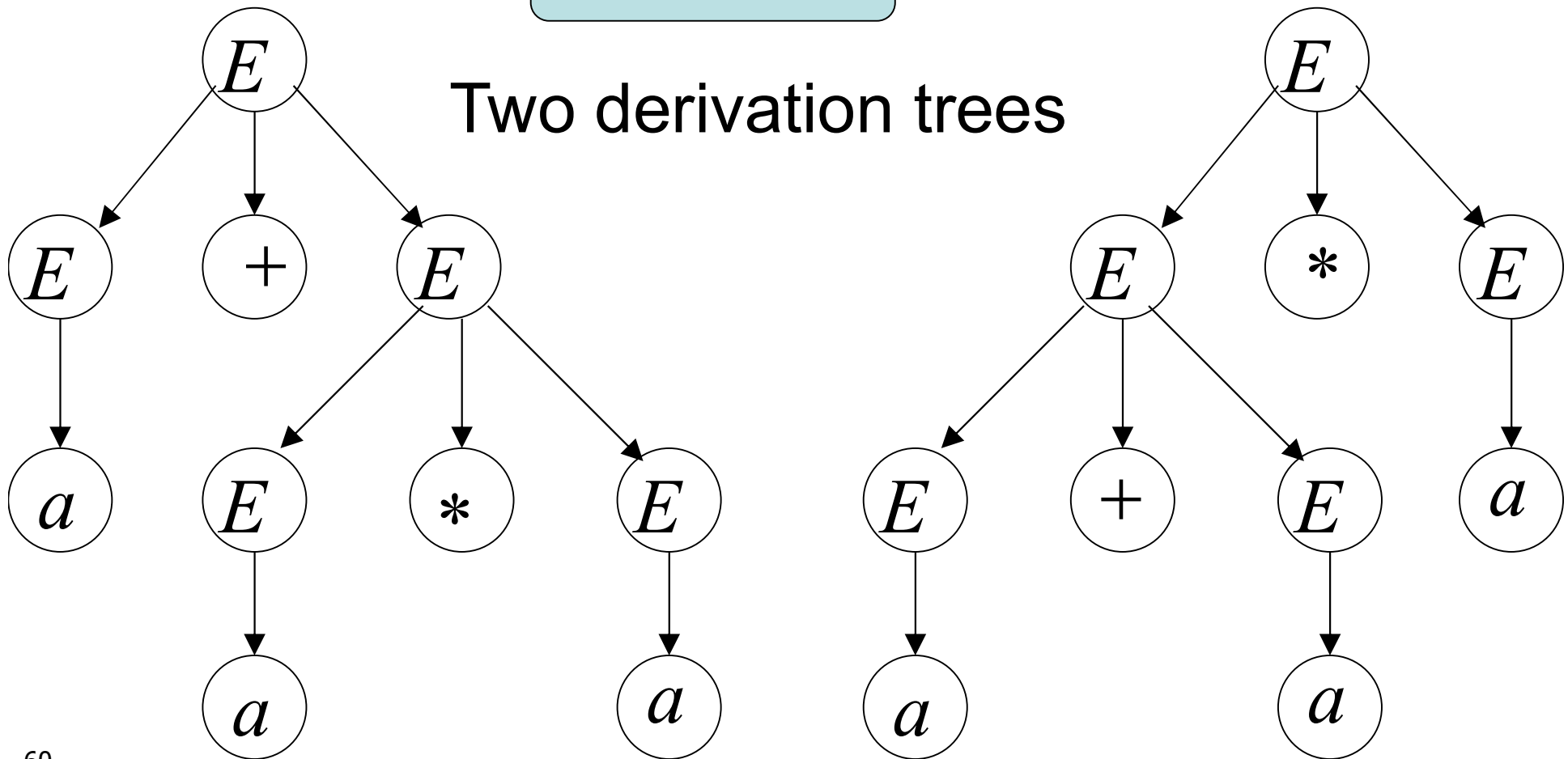


Example 5.11

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

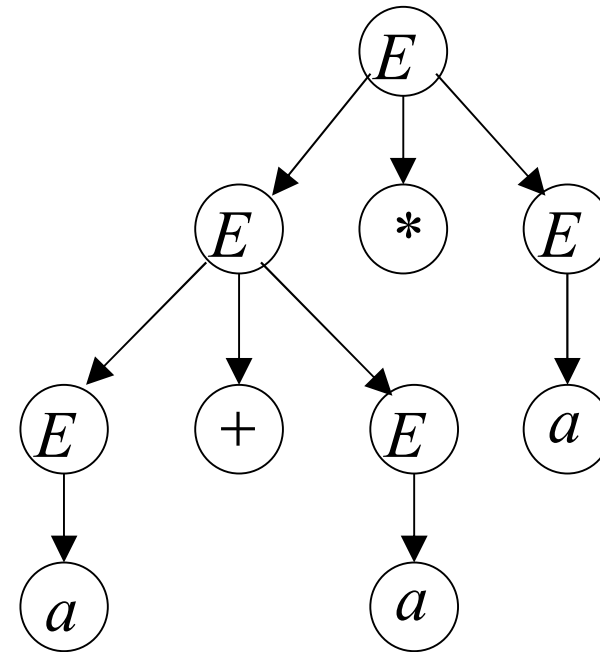
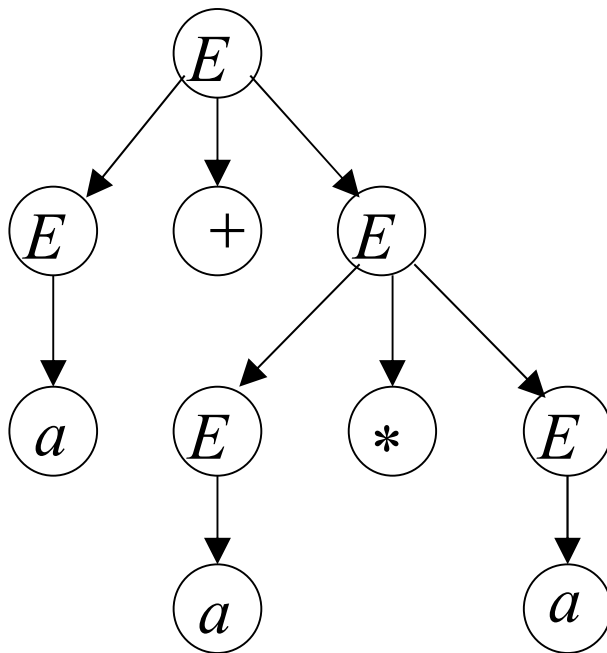
$$a + a * a$$

Two derivation trees



The grammar $E \rightarrow E + E \mid E * E \mid (E) \mid a$
is **ambiguous**:

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



The grammar $E \rightarrow E + E \mid E * E \mid (E) \mid a$
is **ambiguous**:

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

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

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

Definition 5.5:

A context-free grammar G is **ambiguous**

if some string $w \in L(G)$ has:

two or more derivation trees

In other words:

A context-free grammar G is **ambiguous**

if some string $w \in L(G)$ has:

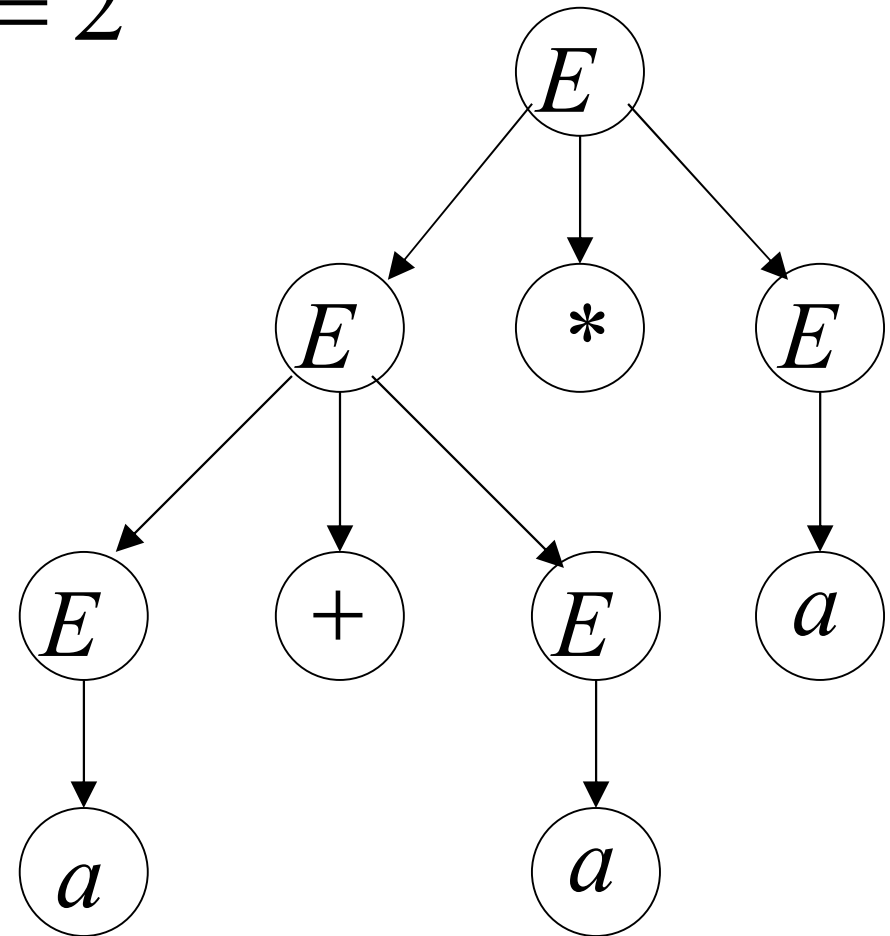
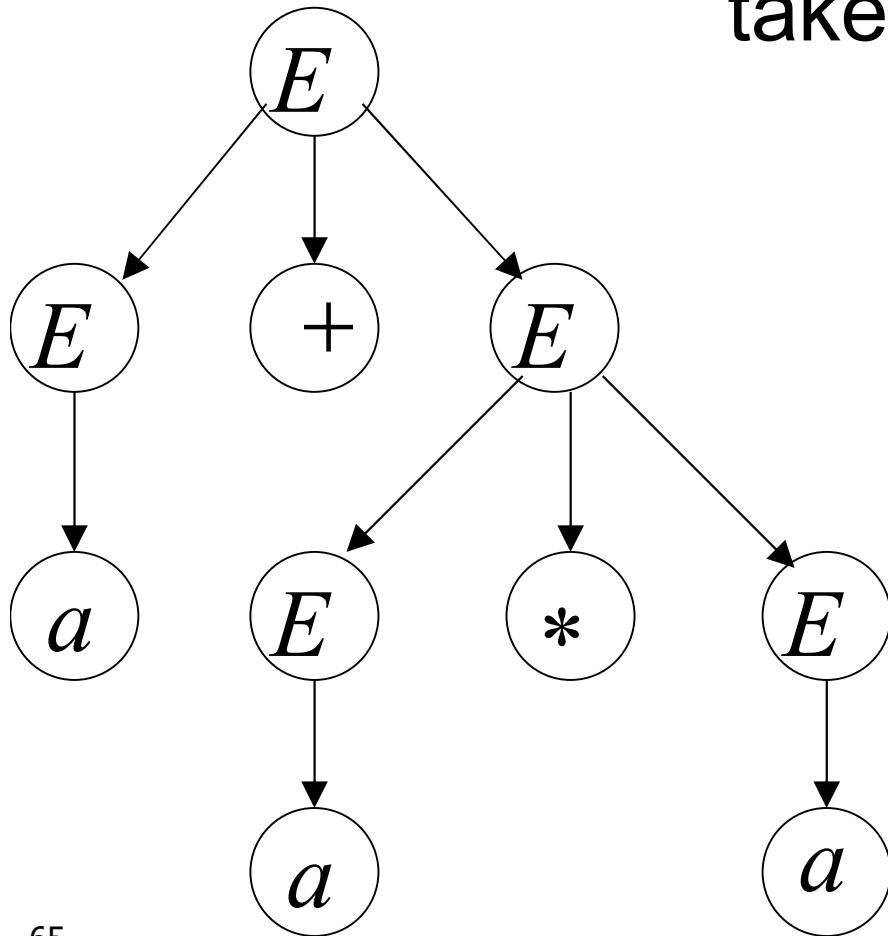
two or more leftmost derivations

(or rightmost)

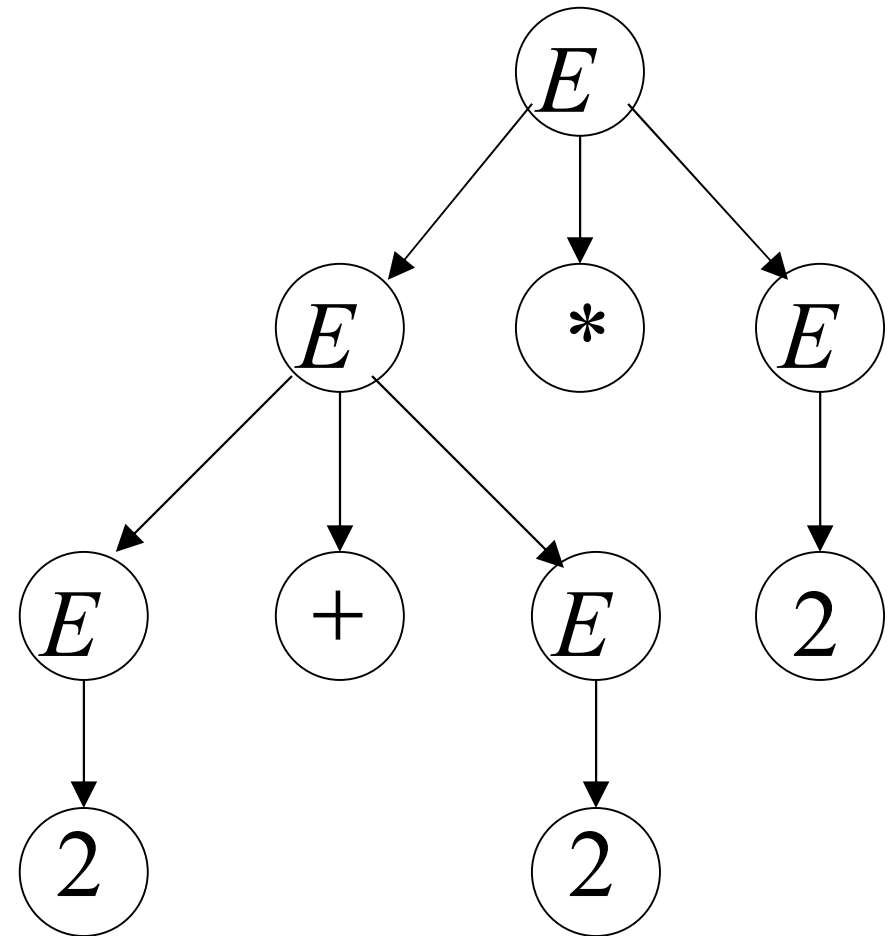
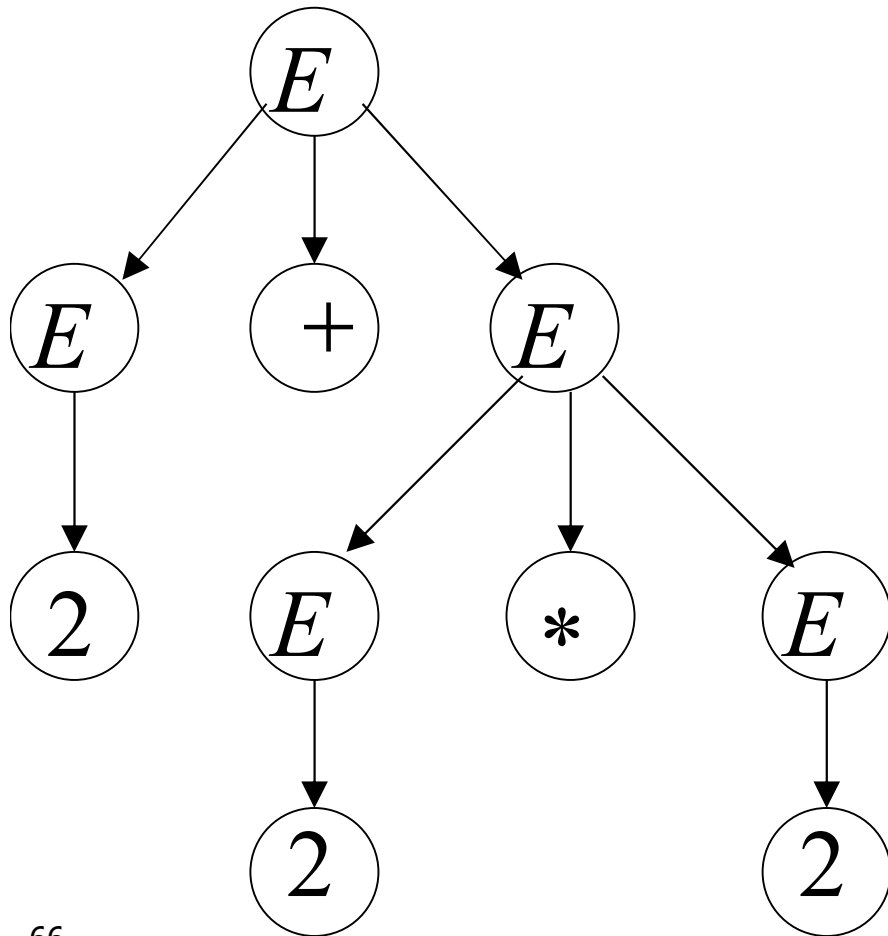
Why do we care about ambiguity?

$$a + a * a$$

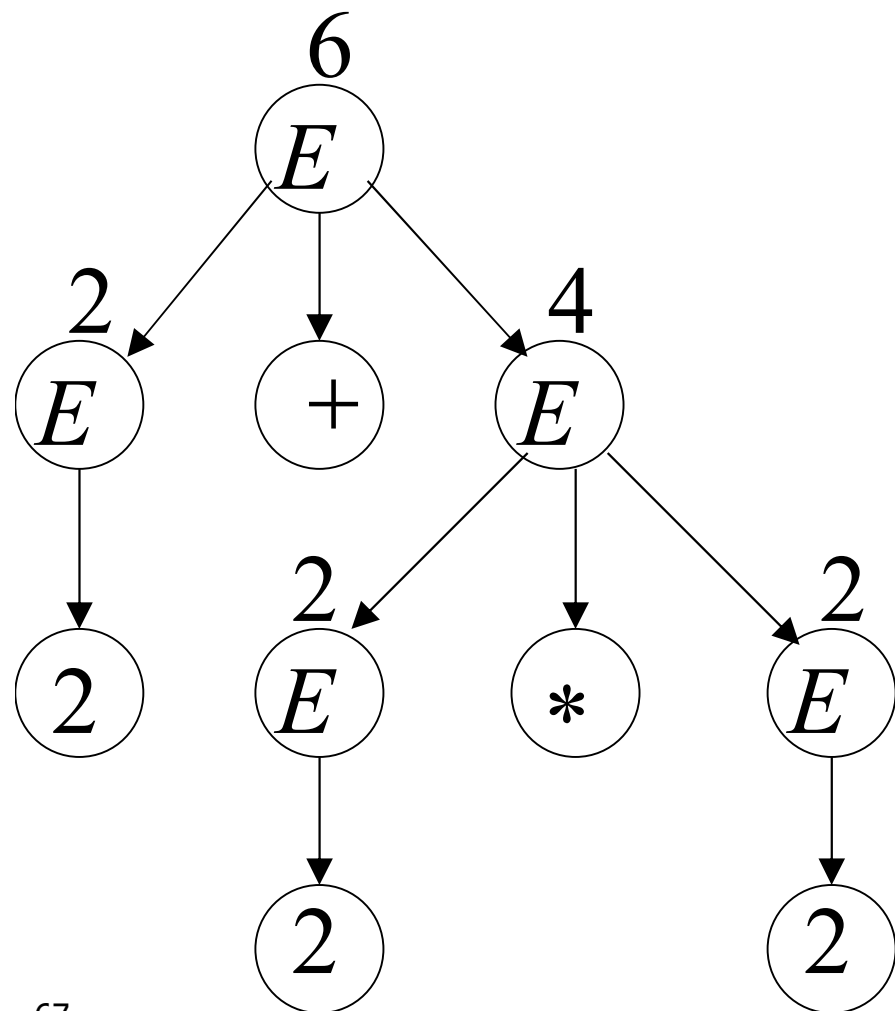
take $a = 2$



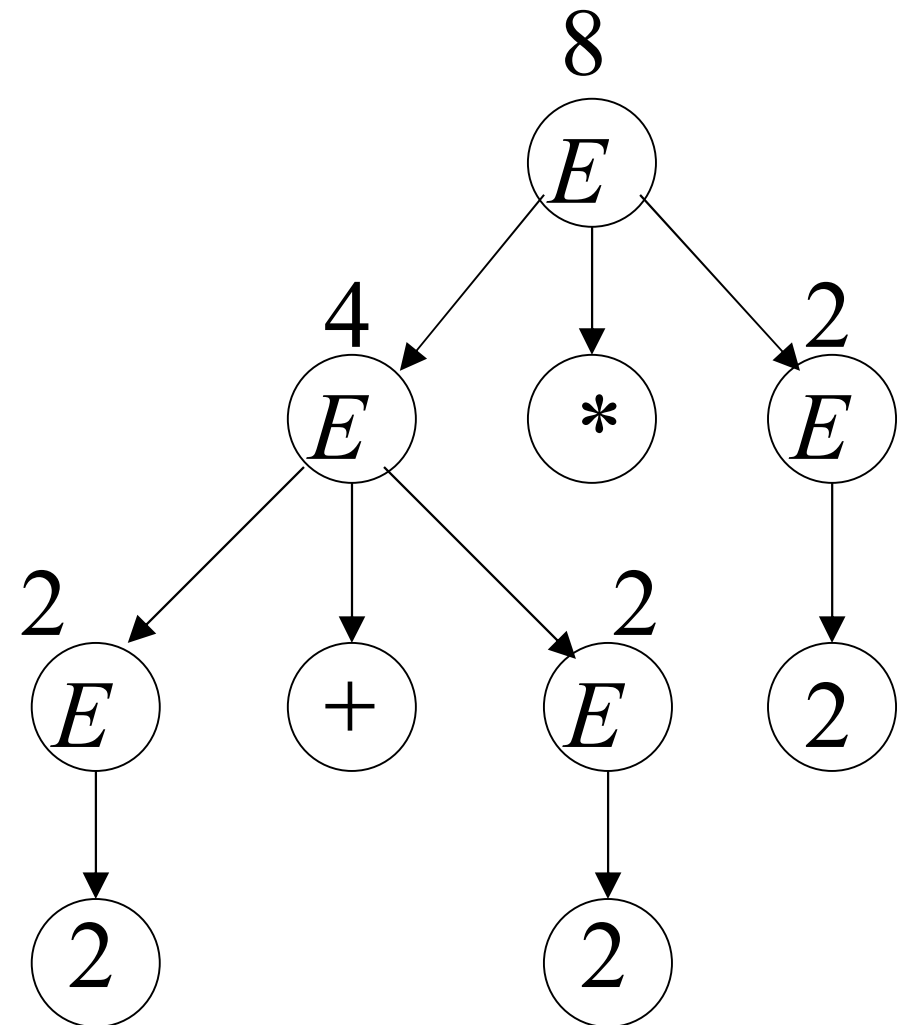
$$2 + 2 * 2$$



$$2 + 2 * 2 = 6$$

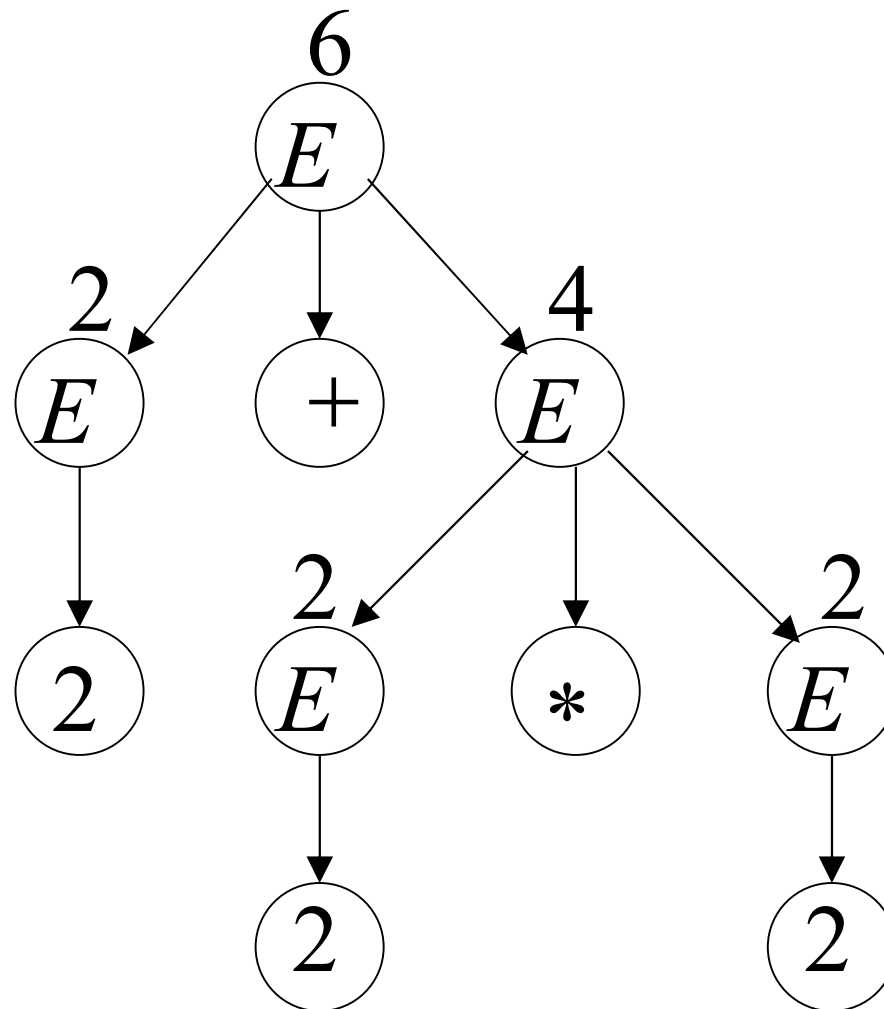


$$2 + 2 * 2 = 8$$



Correct result:

$$2 + 2 * 2 = 6$$



- Ambiguity is **bad** for programming languages
- We want to remove ambiguity

We fix the **ambiguous** grammar:

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

New **non-ambiguous** grammar:

$$\{E, T, F\} \in V$$

priority



$$E \rightarrow E + T$$

$$E \rightarrow T$$

$$T \rightarrow T * F$$

$$T \rightarrow F$$

$$F \rightarrow (E)$$

$$F \rightarrow a$$

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

$$E \rightarrow E + T$$

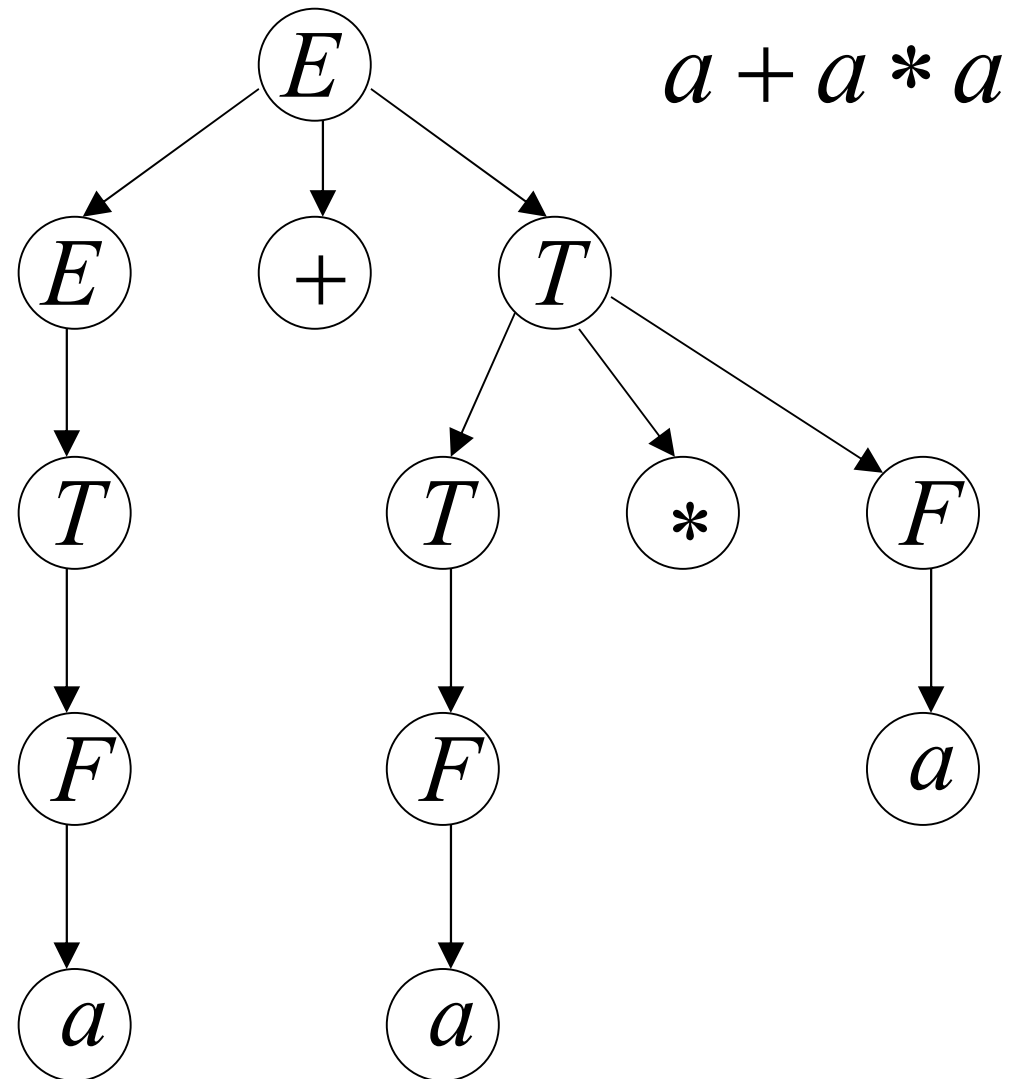
$$E \rightarrow T$$

$$T \rightarrow T * F$$

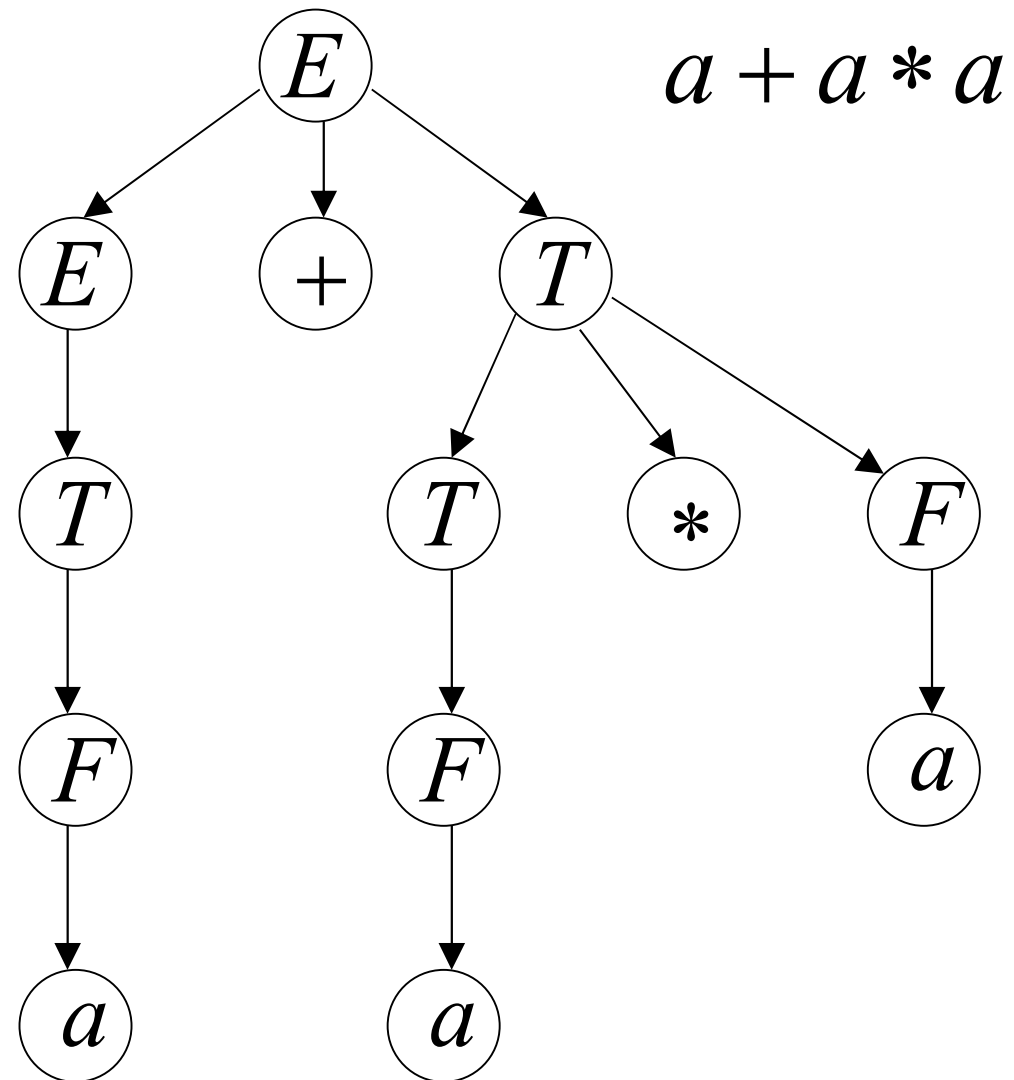
$$T \rightarrow F$$

$$F \rightarrow (E)$$

$$F \rightarrow a$$



Unique derivation tree



The grammar: G $E \rightarrow E + T$

$$E \rightarrow T$$

$$T \rightarrow T * F$$

$$T \rightarrow F$$

$$F \rightarrow (E)$$

$$F \rightarrow a$$

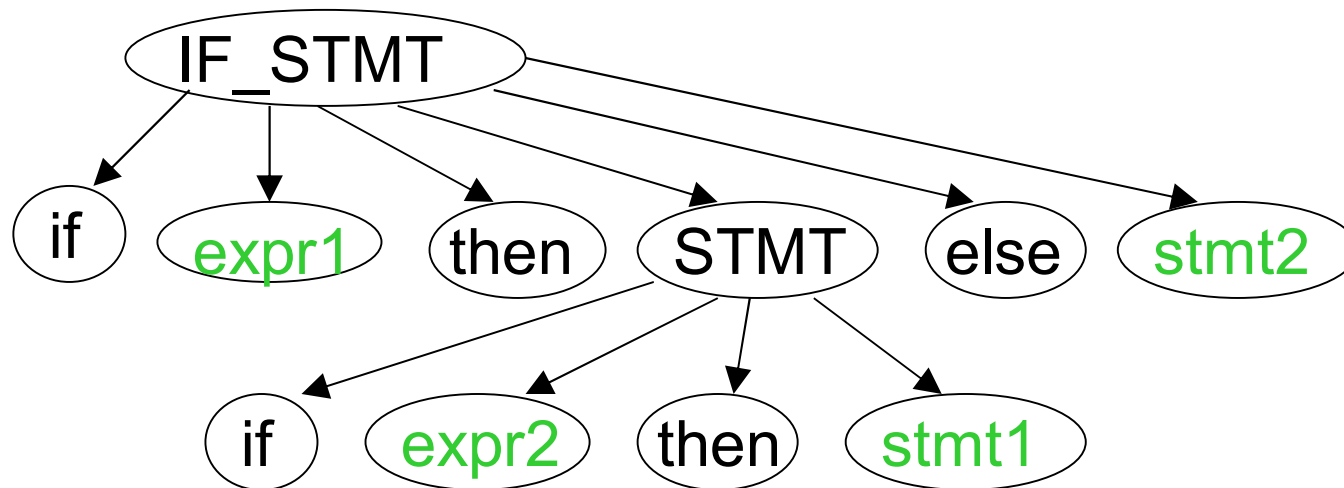
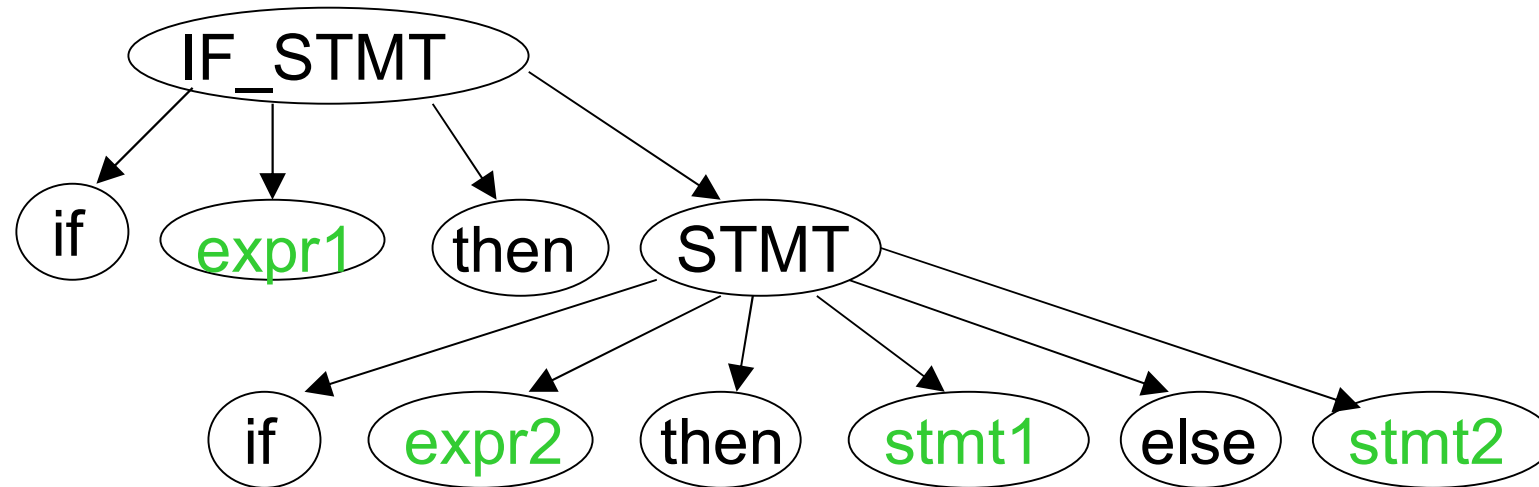
is **unambiguous**:

Every string $w \in L(G)$ has
a unique derivation tree

Another Ambiguous Grammar (Dangling Else)

IF_STMT \rightarrow if EXPR then STMT
 | if EXPR then STMT1 else STMT2

If **expr1** then if **expr2** then **stmt1** else **stmt2**



Inherent Ambiguity

Some context free languages
have only ambiguous grammars

Example:

$$L = \{a^n b^n c^m\} \cup \{a^n b^m c^m\}$$


$$S \rightarrow S_1 \mid S_2$$

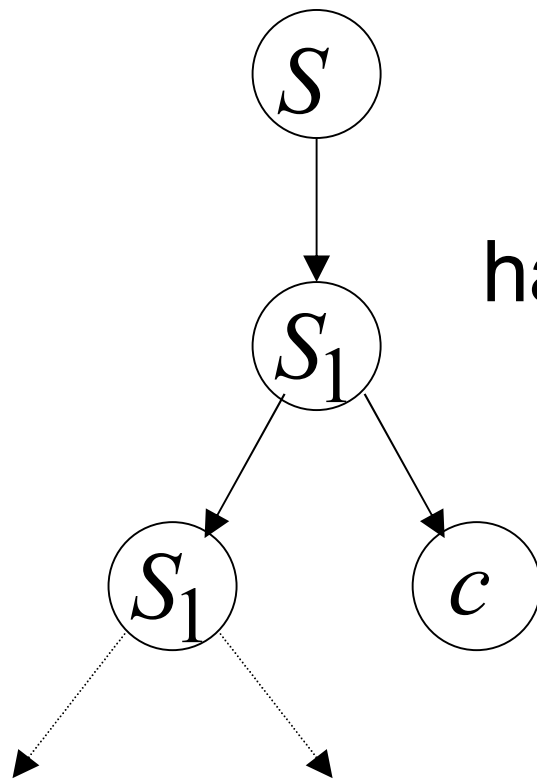

$$S_1 \rightarrow S_1 c \mid A$$

$$A \rightarrow aAb \mid \lambda$$


$$S_2 \rightarrow aS_2 \mid B$$

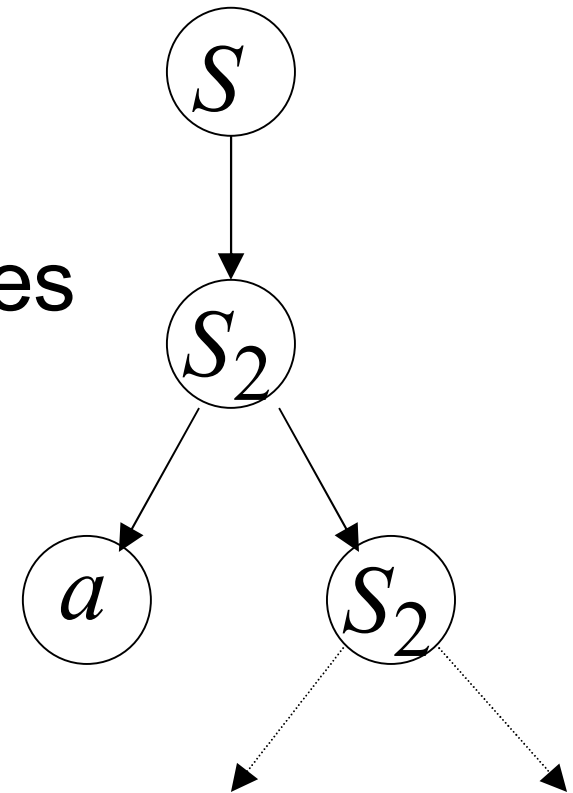
$$B \rightarrow bBc \mid \lambda$$

$$\begin{array}{lll}
 S \rightarrow S_1 \mid S_2 & S_1 \rightarrow S_1 c \mid A & S_2 \rightarrow a S_2 \mid B \\
 & A \rightarrow a A b \mid \lambda & B \rightarrow b B c \mid \lambda
 \end{array}$$



$a^n b^n c^n$
 has two derivation trees

Proof by
Harrison, 1978



Exercise 5.3.6

- Show that the following grammar is ambiguous.

$$S \rightarrow AB \mid aaB$$

$$A \rightarrow a \mid Aa$$

$$B \rightarrow b$$

There are two leftmost derivations for $w = aab$

$$S \Rightarrow aaB \Rightarrow aab,$$

$$S \Rightarrow AB \Rightarrow AaB \Rightarrow aaB \Rightarrow aab$$

Outline



Context-Free Grammars



Parsing and Ambiguity



Context-Free Grammars and Programming Languages

Program

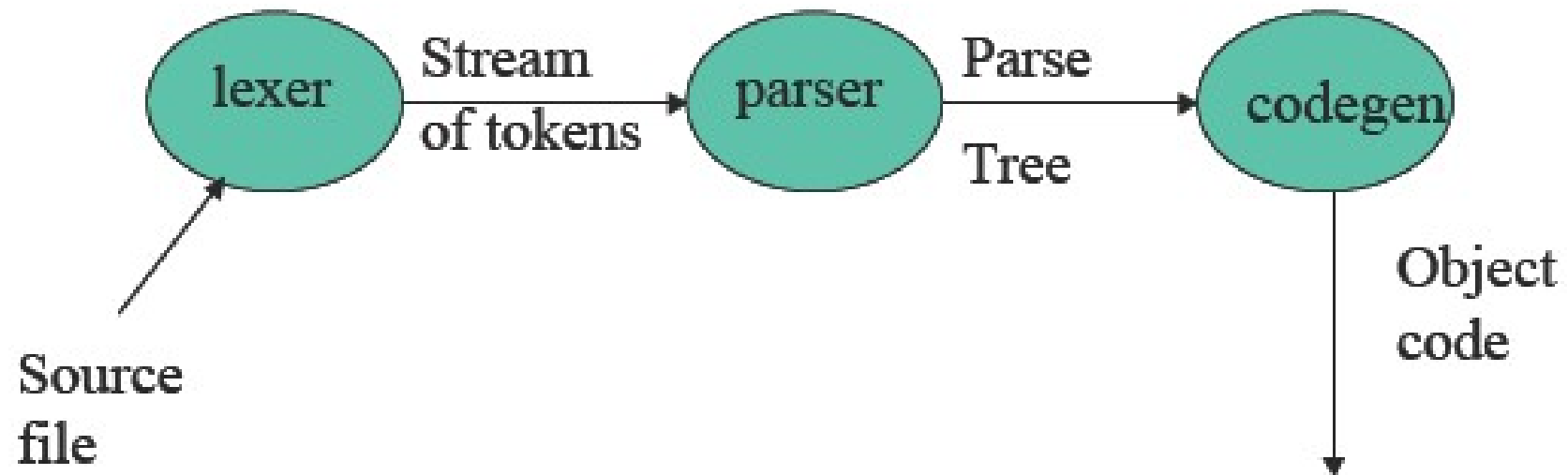
```
v = 5;  
if (v>5)  
    x = 12 + v;  
while (x !=3) {  
    x = x - 3;  
    v = 10;  
}  
.....
```

Compiler

Machine Code

```
Add v,v,0  
cmp v,5  
jmplt ELSE  
THEN:  
    add x, 12,v  
ELSE:  
WHILE:  
    cmp x,3  
...
```

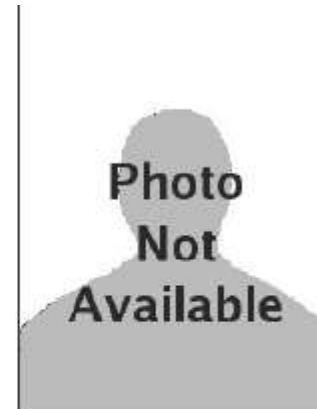

■ How a compiler works



Ken Thompson
Regular expressions in
UNIX / grep / vi



Eric E. Schmidt
Mike Lesk
lex



Stephen C Johnson
yacc

A **parser** knows the grammar
of the programming language

Parser

PROGRAM \rightarrow STMT_LIST

STMT_LIST \rightarrow STMT; STMT_LIST | STMT;

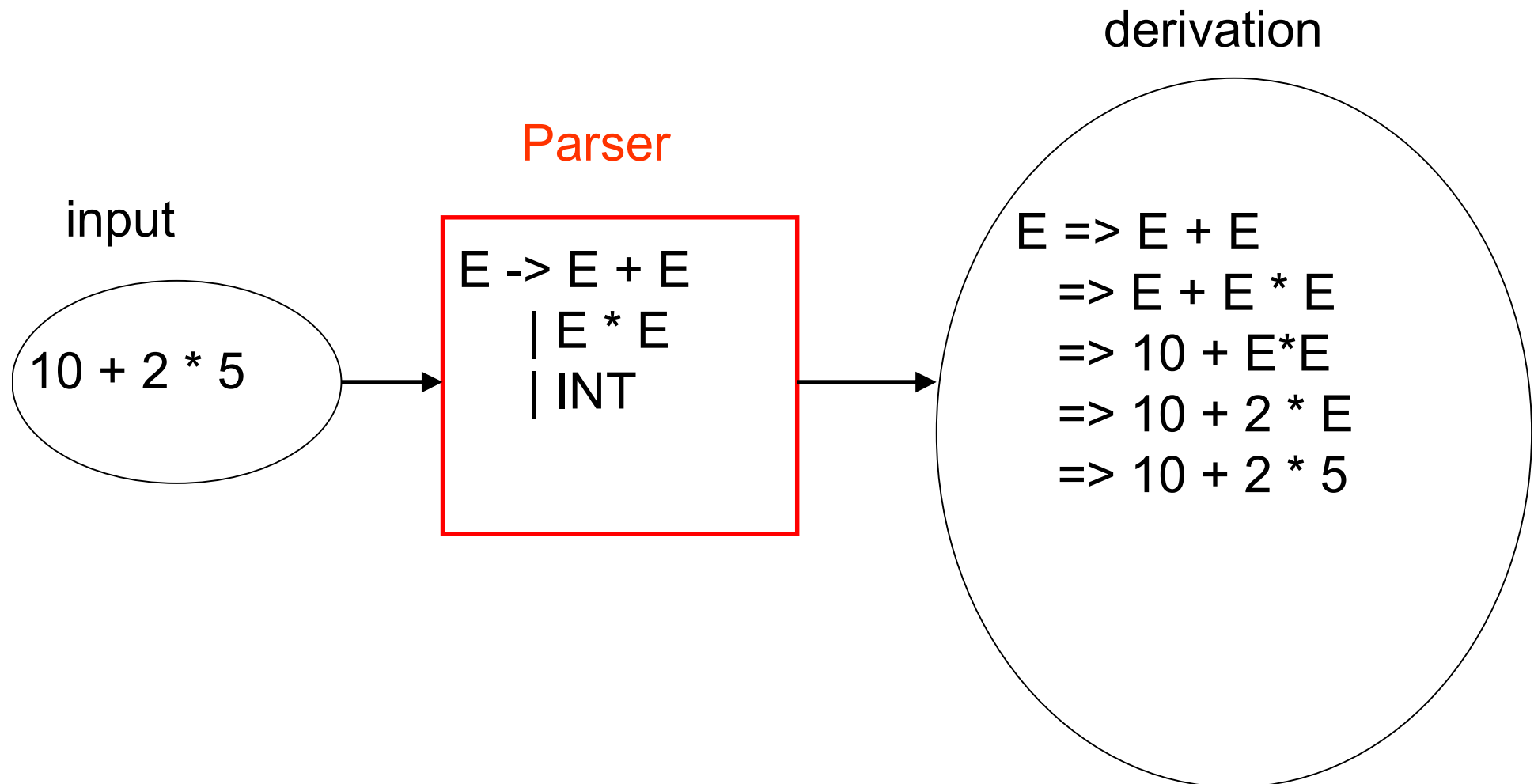
STMT \rightarrow EXPR | IF_STMT | WHILE_STMT
| { STMT_LIST }

EXPR \rightarrow EXPR + EXPR | EXPR - EXPR | ID

IF_STMT \rightarrow if (EXPR) then STMT
| if (EXPR) then STMT else STMT

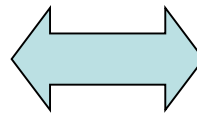
WHILE_STMT \rightarrow while (EXPR) do STMT

The parser finds the derivation
of a particular input

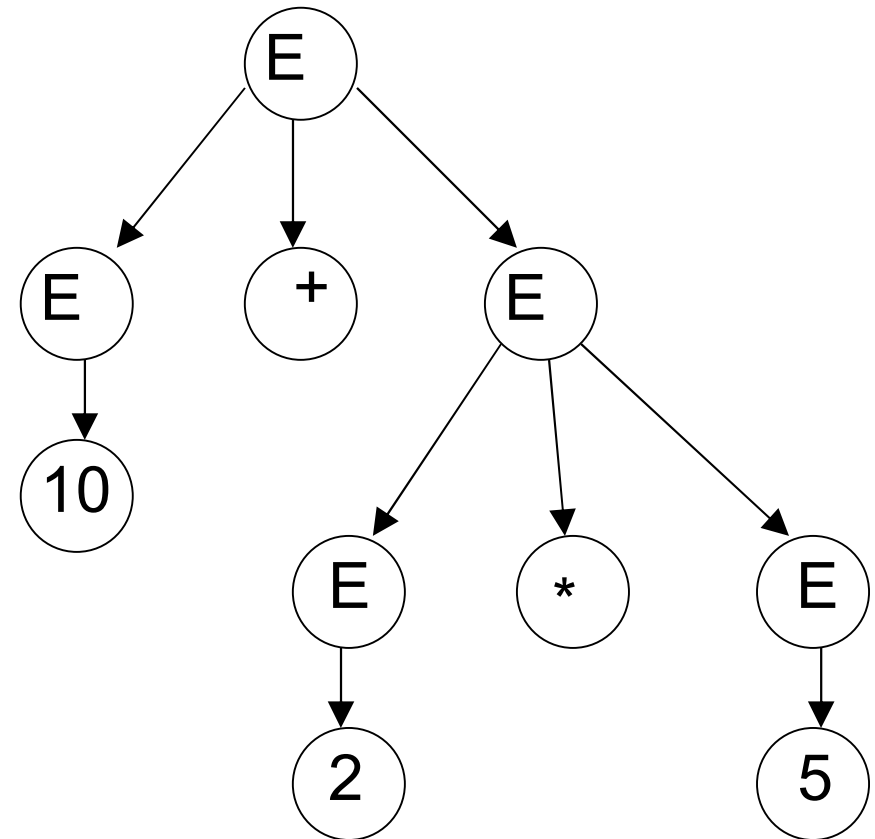


derivation

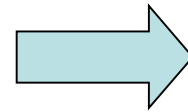
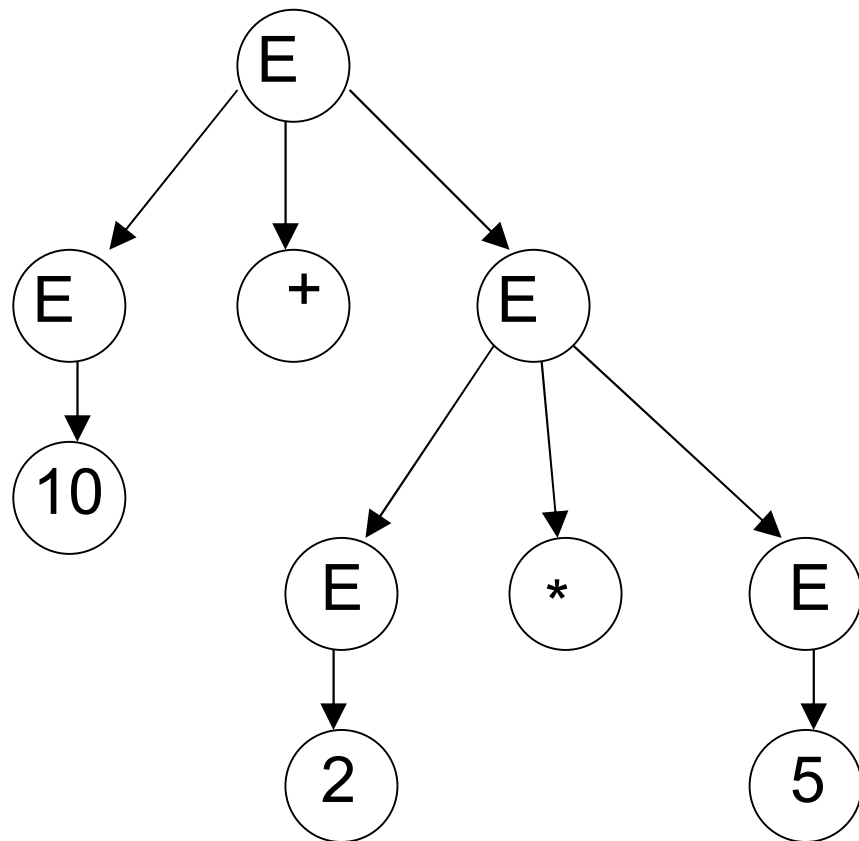
$E \Rightarrow E + E$
 $\Rightarrow E + E * E$
 $\Rightarrow 10 + E * E$
 $\Rightarrow 10 + 2 * E$
 $\Rightarrow 10 + 2 * 5$



derivation tree



derivation tree



machine code

mult a, 2, 5
add b, 10, a

Programming Language and Grammar

- Backus-Naur Form (BNF)
 - Common used grammar for programming languages
 - Ex:
 $\langle \text{expr} \rangle ::= \langle \text{term} \rangle \mid \langle \text{expr} \rangle + \langle \text{term} \rangle$
 $\langle \text{term} \rangle ::= \langle \text{factor} \rangle \mid \langle \text{term} \rangle * \langle \text{factor} \rangle$
- LL and LR grammar: parse in linear time

Questions?