2019

Theory of Computation

Kun-Ta Chuang
Department of Computer Science and Information Engineering
National Cheng Kung University

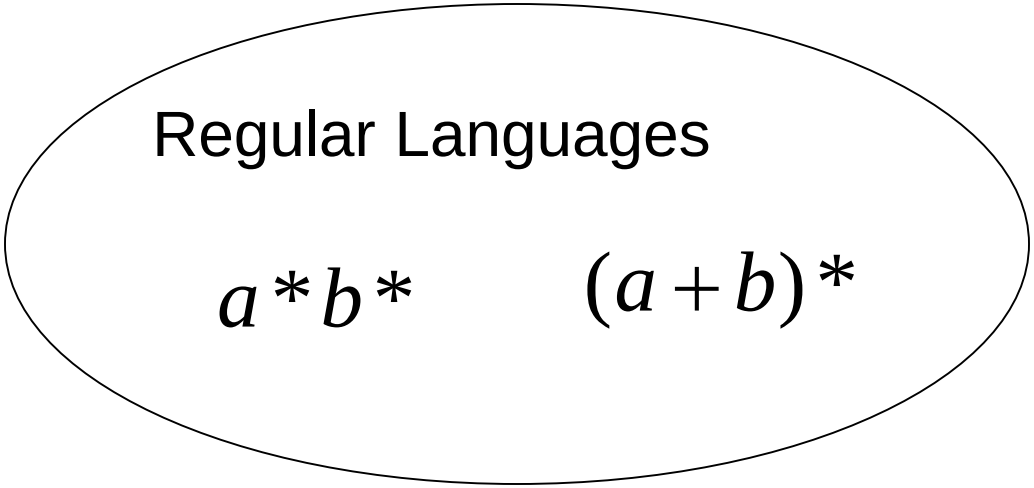
Outline

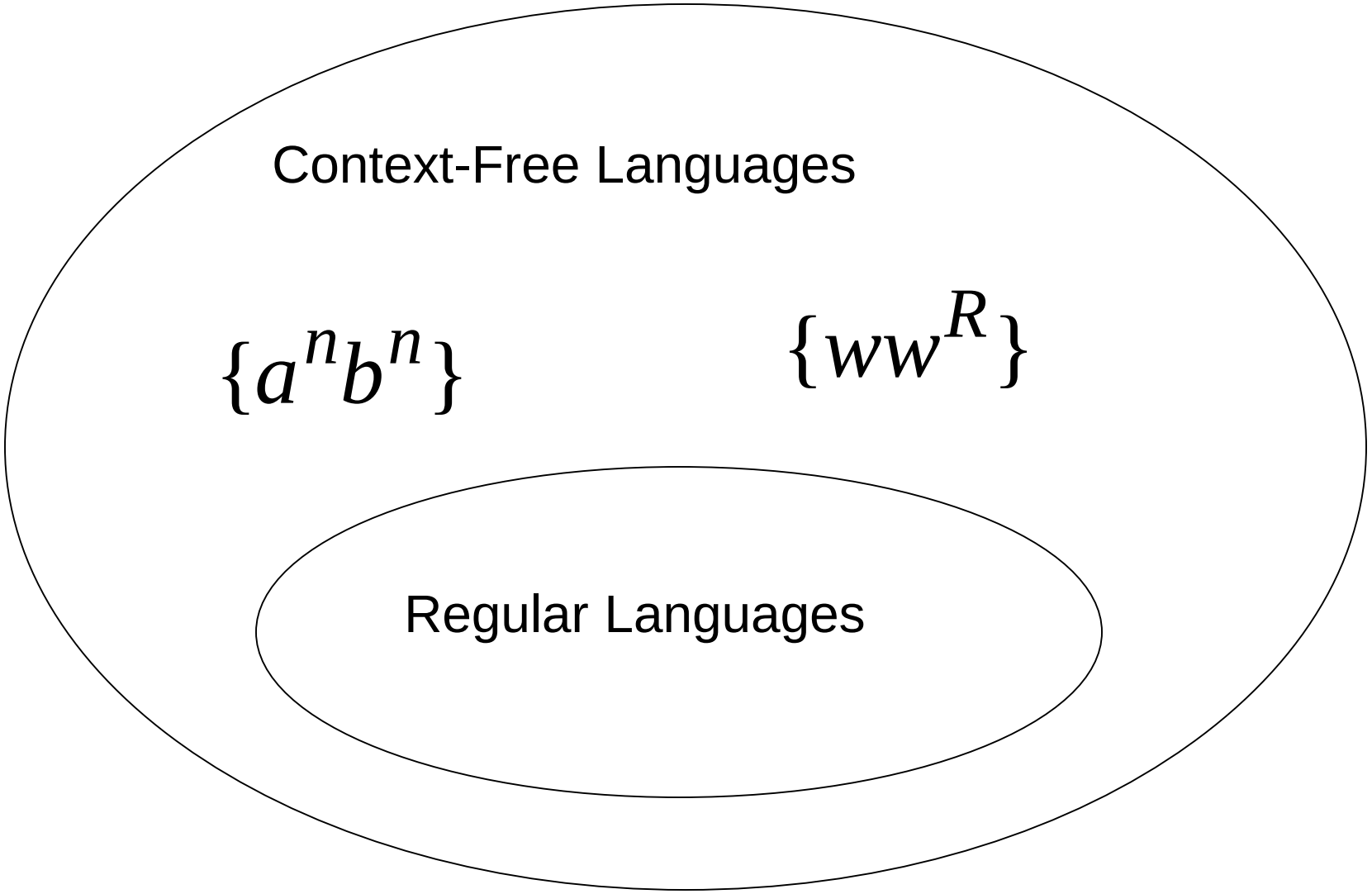
Context-Free Grammars

Parsing and Ambiguity

Context-Free Grammars and Programming Languages

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

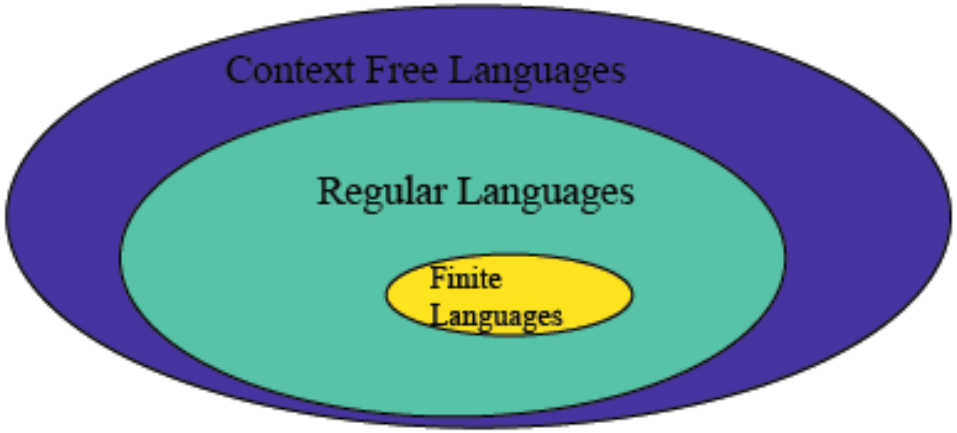


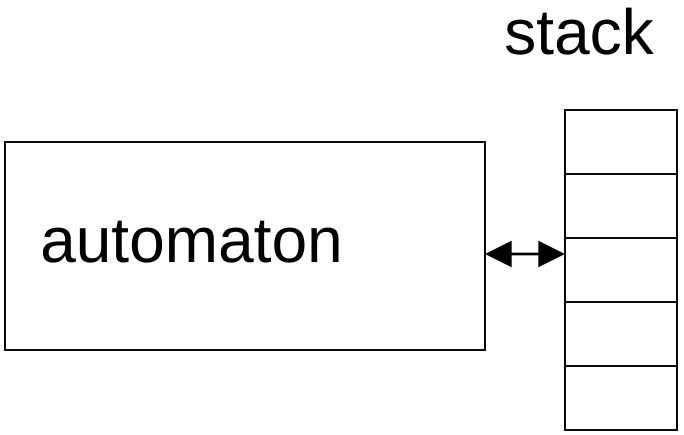

Context-Free Languages
$\{a^n b^n\}$
$\{ww^R\}$
Regular Languages

Context-Free Languages

Context-Free Grammars

Pushdown Automata





Context-Free Grammars

Regular Grammar

$$S \to abS$$

$$S \to a$$

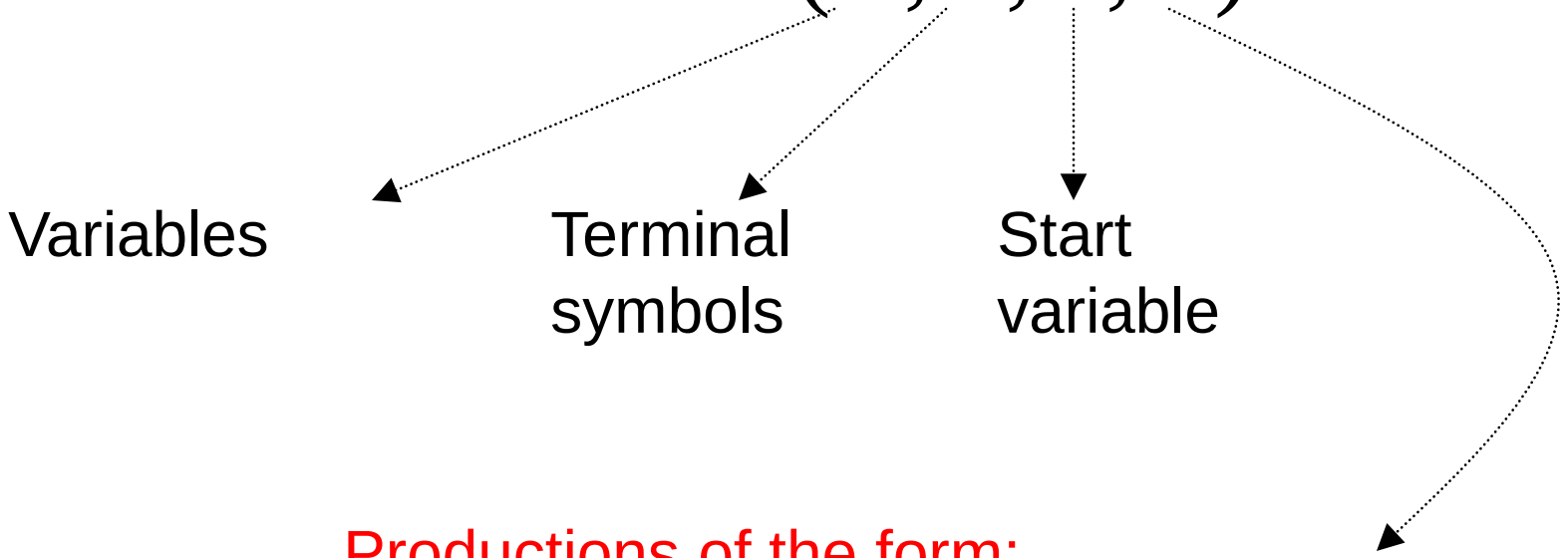
$$S \to Aab$$

$$A \to Aab \mid B$$

$$B \to a$$

Definition 5.1: Context-Free Grammars

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



Productions of the form:

Variable $A \rightarrow x$ String of variables and terminals

A ϵ V

x ϵ (V U T)*

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

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

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

S不斷的轉換，轉換到沒有variables，稱作一個string

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

$$S \to aSb$$

雖然為linear，但是因為Variable放在中間，所以不是regular

$$S \to \lambda$$

A derivation:

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

Example

A context-free grammar : G

$$S \to aSb$$

$$S \to \lambda$$

Another derivation:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

$$S \to aSb$$

$$S \to \lambda$$

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

Describes parentheses: (((( ))))

Example 5.1

A context-free grammar: G

$$S \to aSa$$

$$S \to bSb$$

$$S \to \lambda$$

A derivation:

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

A context-free grammar: G

$$S \to aSa$$
$$S \to bSb$$
$$S \to \lambda$$

Another derivation:

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

$$S \to aSa$$

$$S \to bSb$$

$$S \to \lambda$$

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

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

Example 5.2

A context-free grammar: G

$$S \to abB$$

$$A \to aaBb$$

$$B \to bbAa$$

$$A \to \lambda$$

A derivation:

$$S \Rightarrow abB \Rightarrow abbbAa \Rightarrow abbbaaBba$$

$$\Rightarrow abbbaabbAaba \Rightarrow abbbaabbaba$$

$$S \to abB$$

$$A \to aaBb$$

$$B \to bbAa$$

$$A \to \lambda$$

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

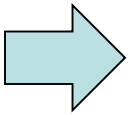
The above grammars are not only context-free, but linear.
Regular and linear grammars are context-free,
But a context-free grammar is not necessarily linear.

Example 5.3

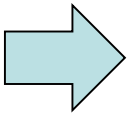
The language $L = \{a^n b^m : n \neq m\}$ is context-free

一開始選要a比較多還是b比較多

$$S \to aSb \mid S \to \lambda$$



$$S \to AS_1$$
$$S_1 \to aS_1b \mid \lambda$$
$$A \to aA \mid a$$



$$S \to AS_1 \mid S_1B$$
$$S_1 \to aS_1b \mid \lambda$$
$$A \to aA \mid a$$
$$B \to bB \mid b$$

多引入了A可以變成aA｜a確保a比b多

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

$n > m$

$n < m$

The resulting grammar is context-free but not linear

Example 5.4

A context-free grammar: G

$$S \to aSb$$

$$S \to SS$$

$$S \to \lambda$$

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

$$S \to aSb$$

$$S \to SS$$

$$S \to \lambda$$

More derivations:

$$S \Rightarrow SS \Rightarrow aSbS \Rightarrow abS \Rightarrow abaS$$

$$\Rightarrow abaaSbb \Rightarrow abaabb$$

$$S \Rightarrow aSb \Rightarrow aSSb \Rightarrow aaSbSb$$

$$\Rightarrow aabSb \Rightarrow aababSbb \Rightarrow aababb$$

$$S \to aSb$$

$$S \to SS$$

$$S \to \lambda$$

$$L(G) = \{w \ : n_a(w) = n_b(w),$$
$$\text{and } n_a(v) \geq n_b(v)$$
$$\text{in any prefix } v\}$$

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. $A \rightarrow aaA$
3. $A \rightarrow \lambda$
4. $B \rightarrow Bb$
5. $B \rightarrow \lambda$

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

Derivation Order

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

Leftmost derivation:

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

Rightmost derivation:

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

Example 5.5

$$1.\ S \to aAB \qquad 2.\ A \to bBb$$

$$3.\ B \to A \qquad 4.\ B \to \lambda$$

Leftmost derivation:

$$S \overset{1}{\Rightarrow} aAB \overset{2}{\Rightarrow} abBbB \overset{3}{\Rightarrow} abAbB \overset{2}{\Rightarrow} abbBbbB$$

$$\overset{4}{\Rightarrow} abbbbB \overset{4}{\Rightarrow} abbbb$$

Rightmost derivation:

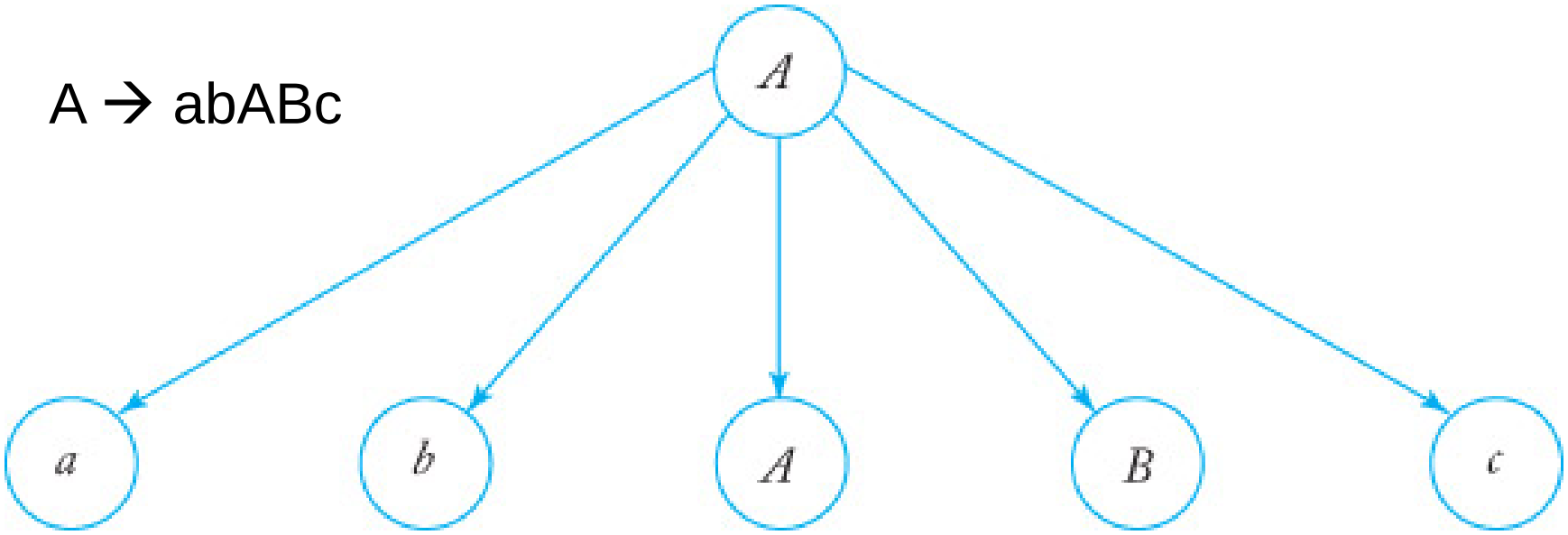
$$S \overset{1}{\Rightarrow} aAB \overset{4}{\Rightarrow} aA \overset{2}{\Rightarrow} abBb \overset{3}{\Rightarrow} abAb$$

$$\overset{2}{\Rightarrow} abbBbb \overset{4}{\Rightarrow} abbbb$$

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.

A → abABc



Definition 5.3

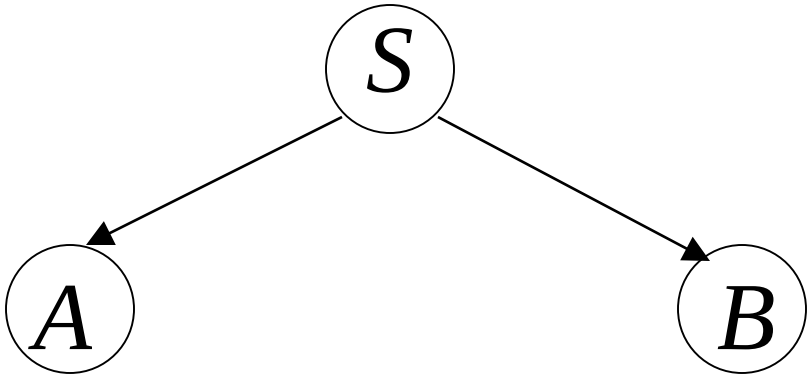
- Let G = (V, T, S, P) be a CFG. An ordered tree is a derivation tree for G iff it has the following properties.
  1. The root is labeled S
  2. Every leaf has a label from T U {λ}
  3. Every internal vertex has a label from V
  4. If a vertex has label A ϵ V, and its children are labeled a_1, a_2, …, a_n, then P must contain a production

     $$A \rightarrow a_1 a_2 \ldots a_n$$

  5. A leaf labeled λ has no sibling

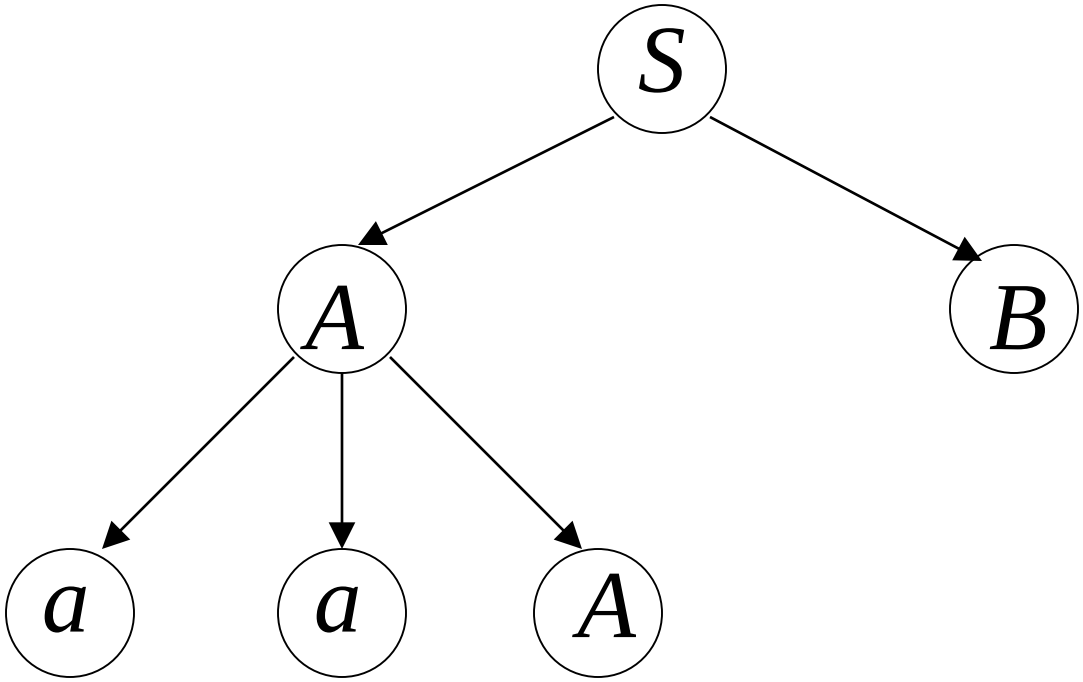
$S \to AB$ $A \to aaA \mid \lambda$ $B \to Bb \mid \lambda$

$S \Rightarrow AB$



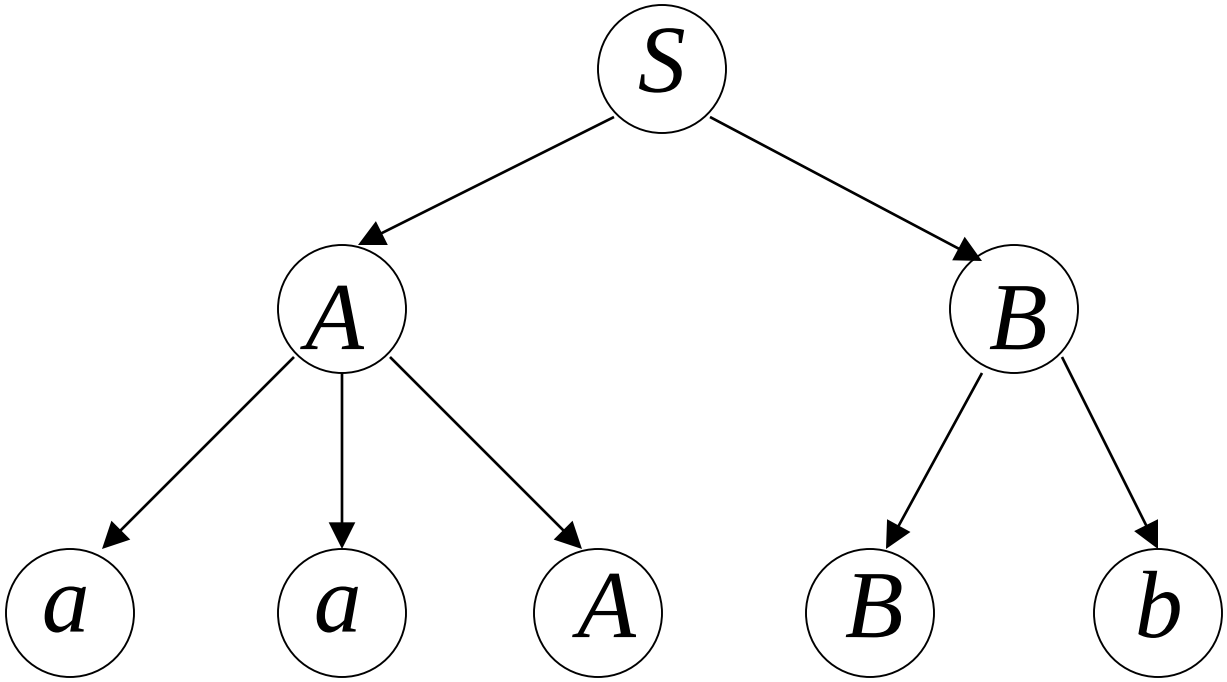
$S \to AB$ $A \to aaA \mid \lambda$ $B \to Bb \mid \lambda$

$$S \Rightarrow AB \Rightarrow aaAB$$



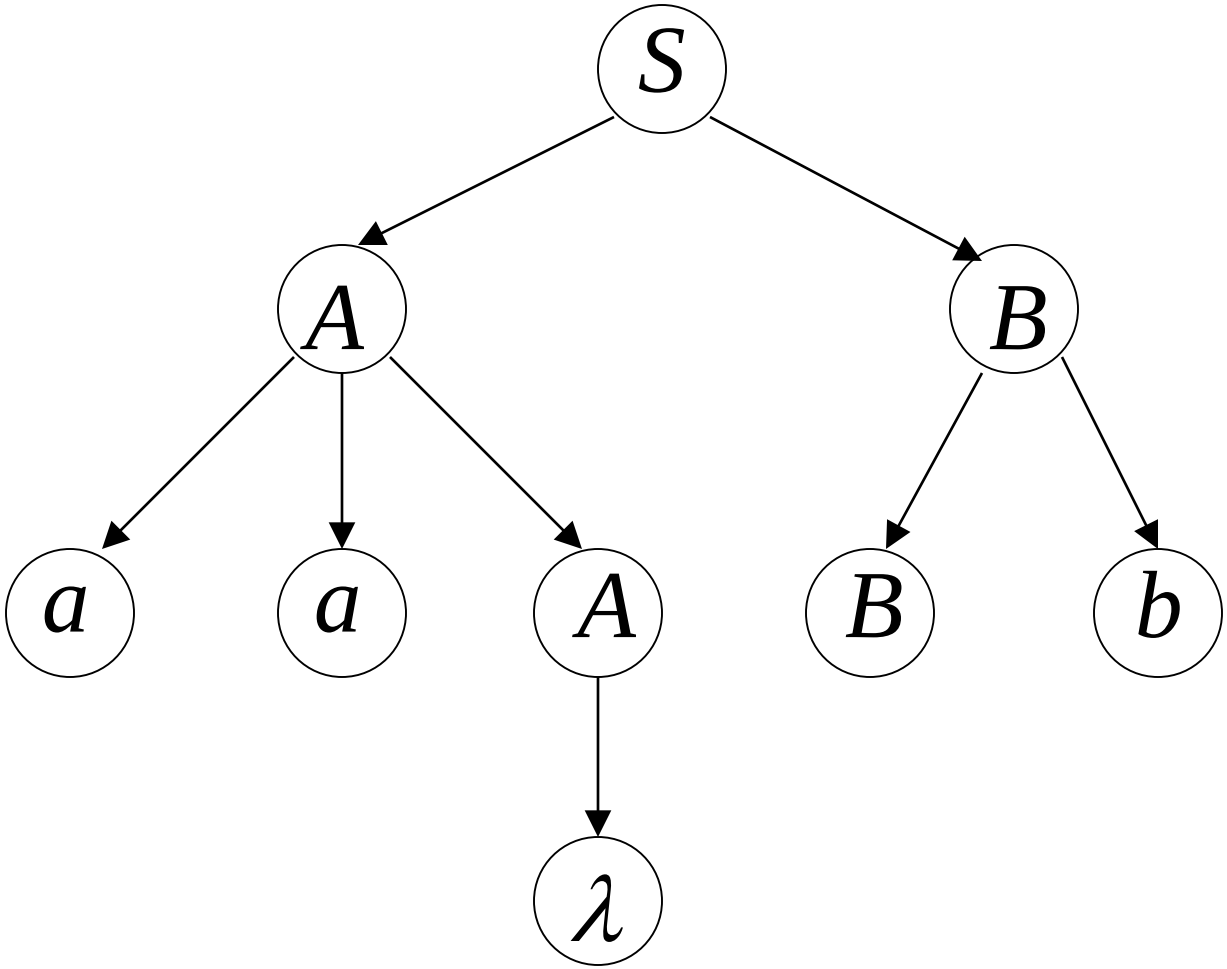
$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$

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



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

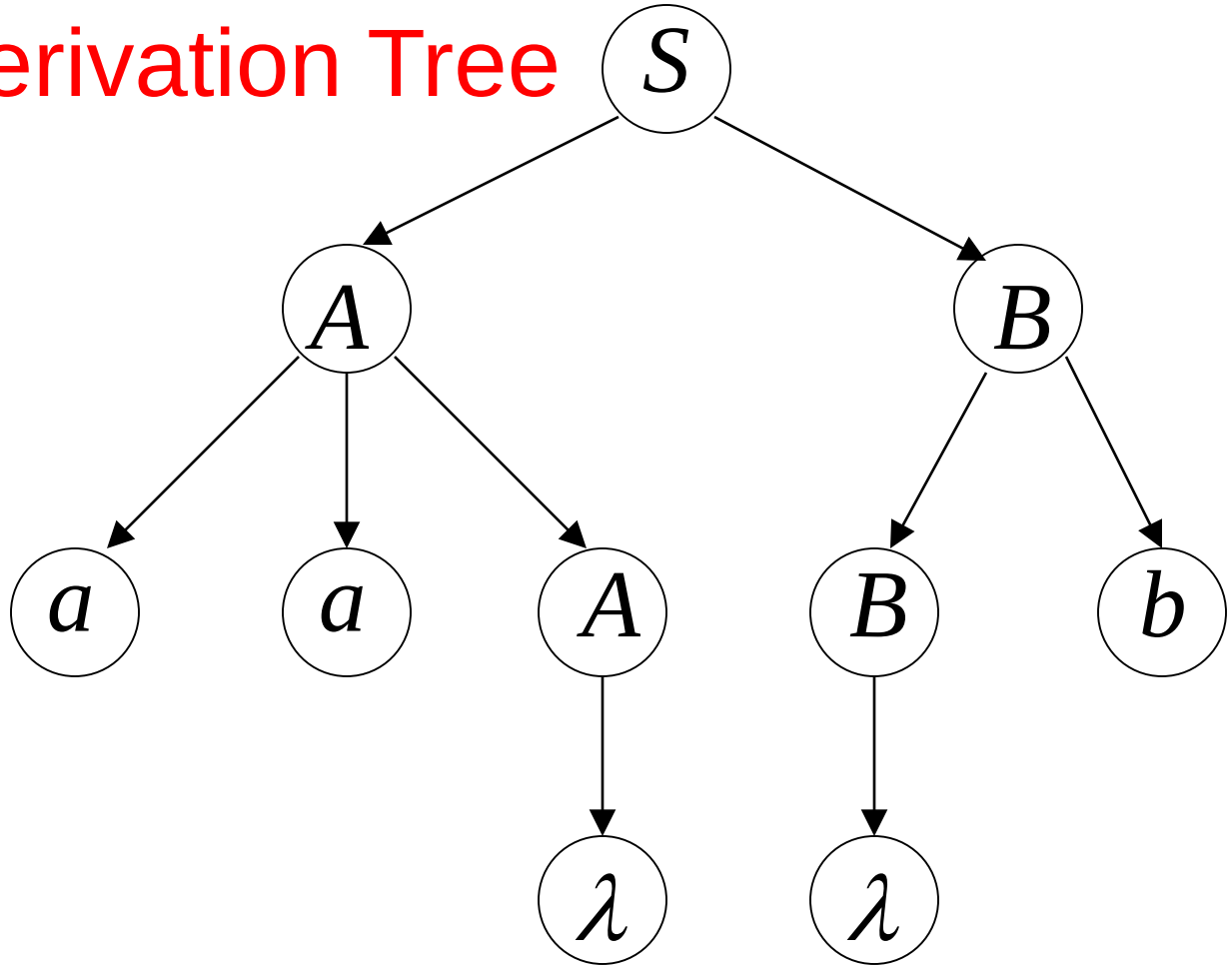
$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb \Rightarrow aaBb$



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

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

Derivation Tree

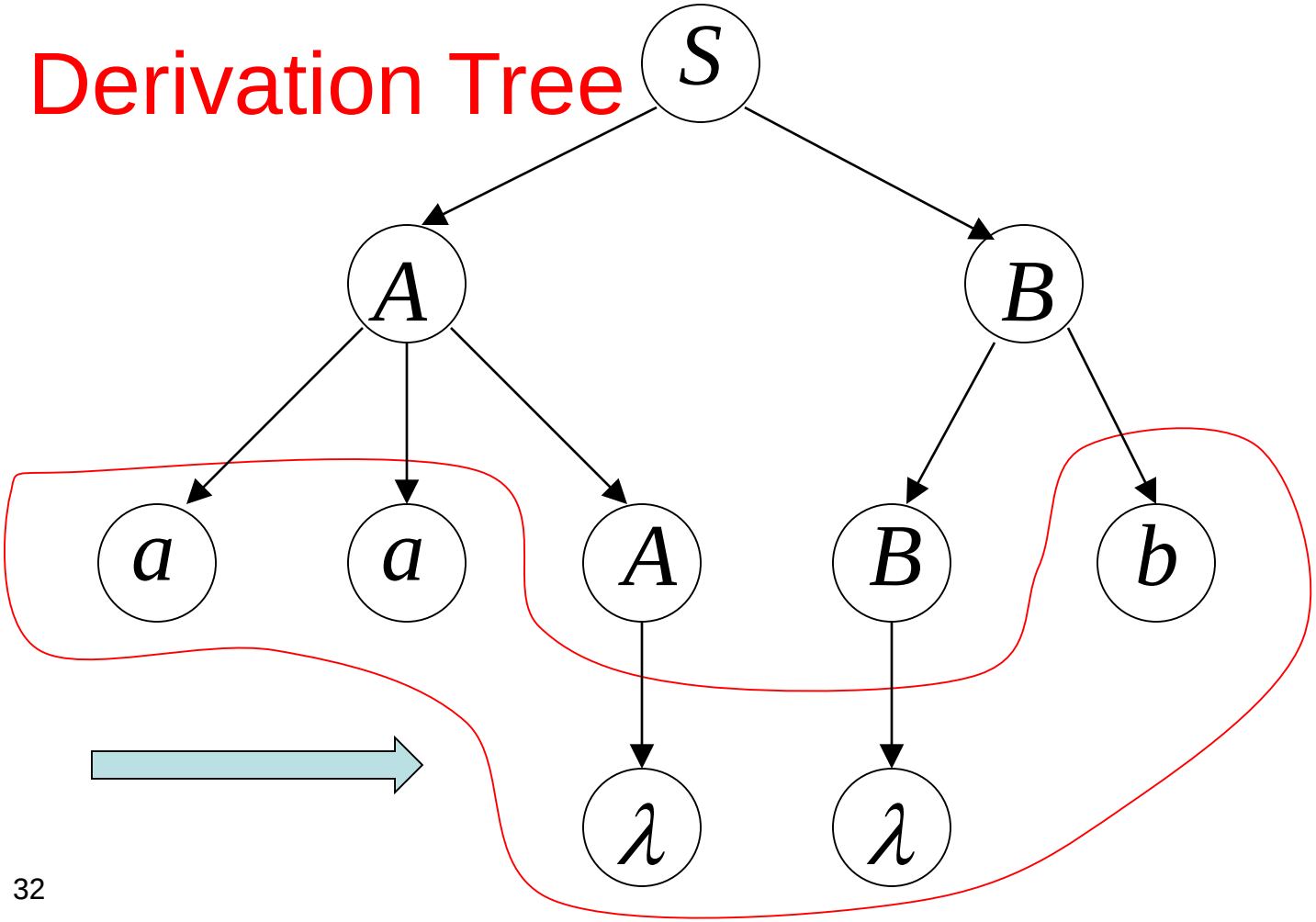


Depth-first search manner from the leftmost unexplored branch

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

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

Derivation Tree



Yield

$aa\lambda\lambda b$

$= aab$

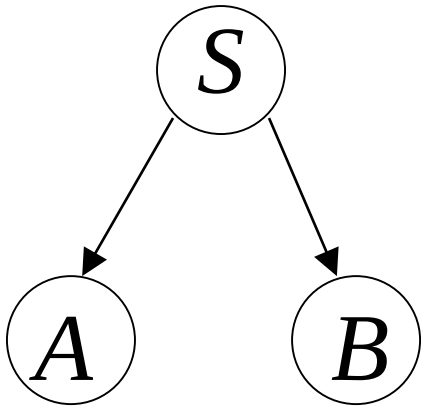
Partial Derivation Trees

$$S \rightarrow AB$$

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

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

$$S \Rightarrow AB$$

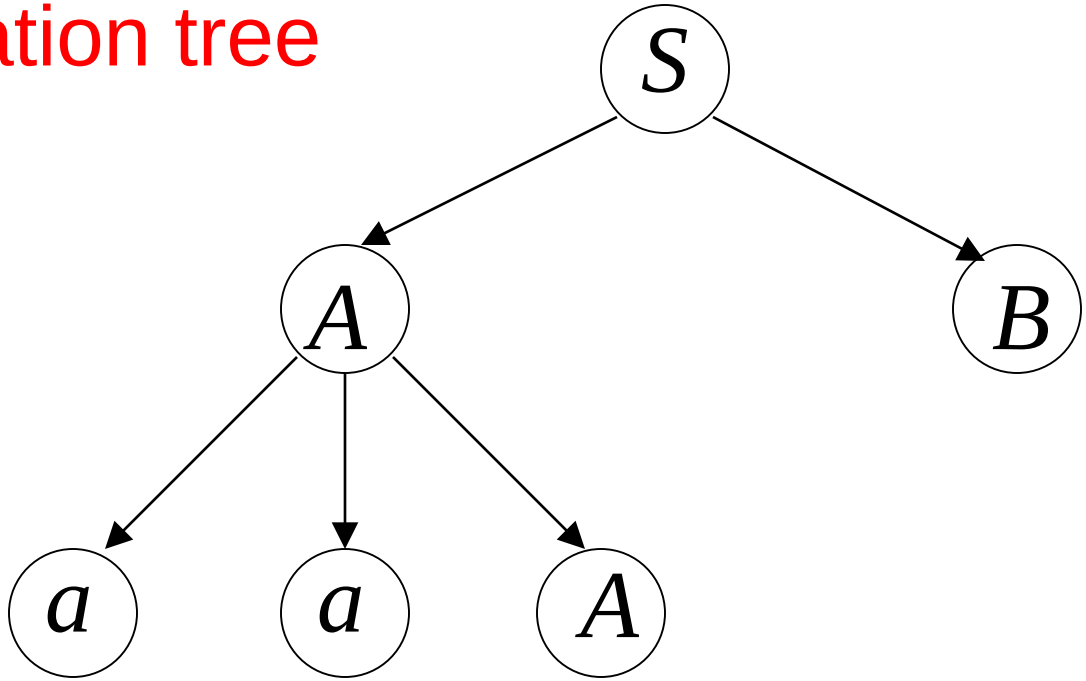


Partial derivation tree

- 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 U T U {λ}

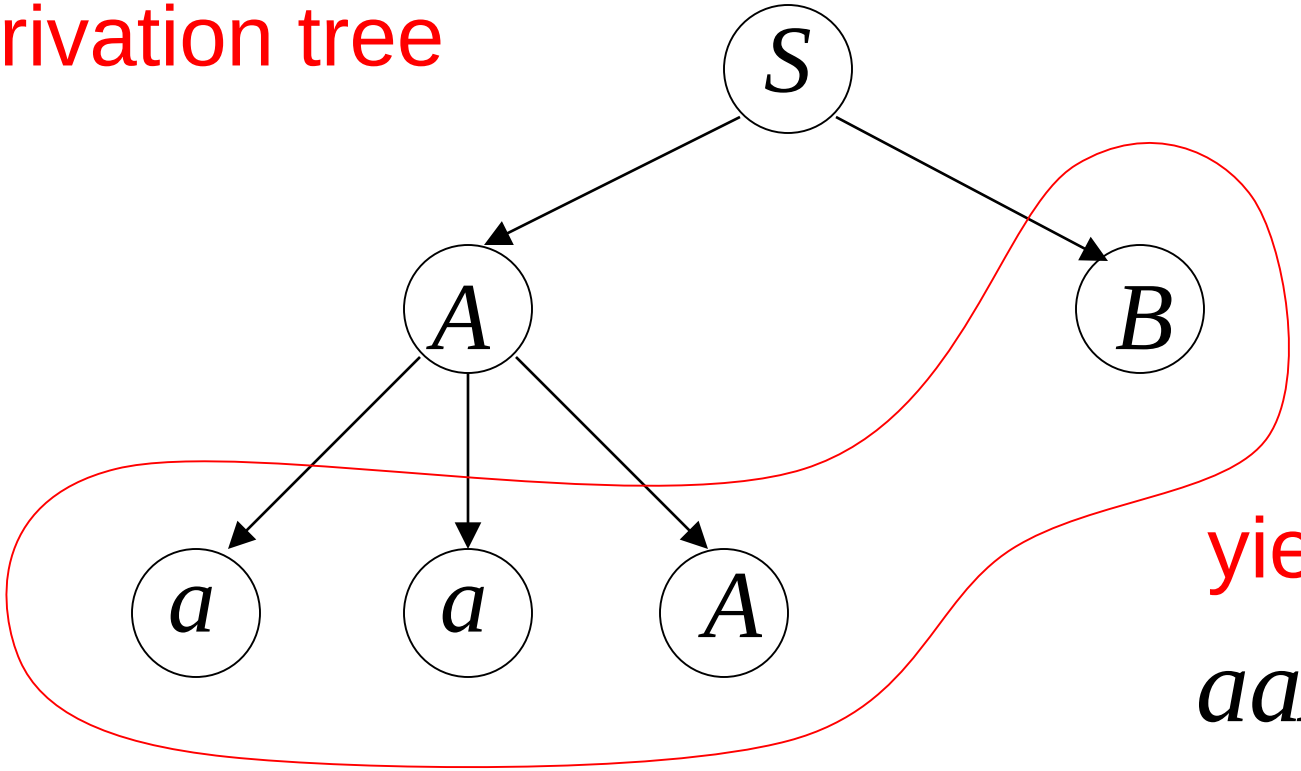
$$S \Rightarrow AB \Rightarrow aaAB$$

Partial derivation tree



$$S \Rightarrow AB \Rightarrow aaAB$$ sentential form

Partial derivation tree



yield

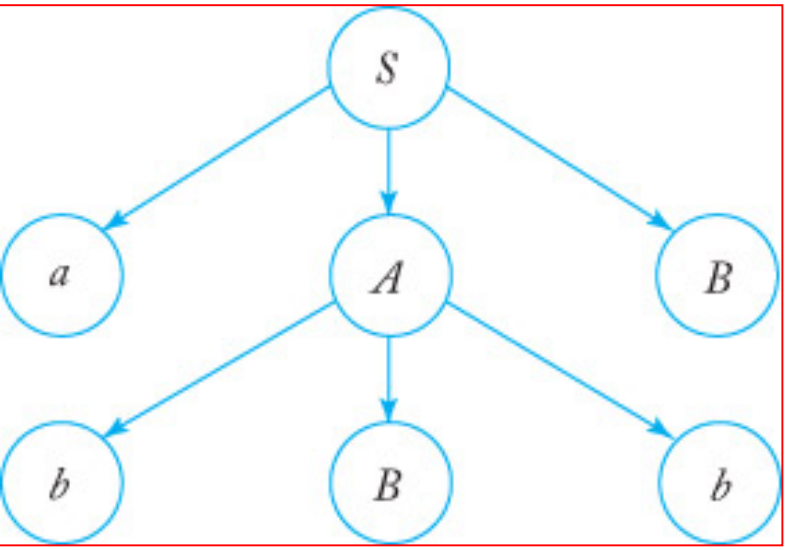
$aaAB$

Example 5.6

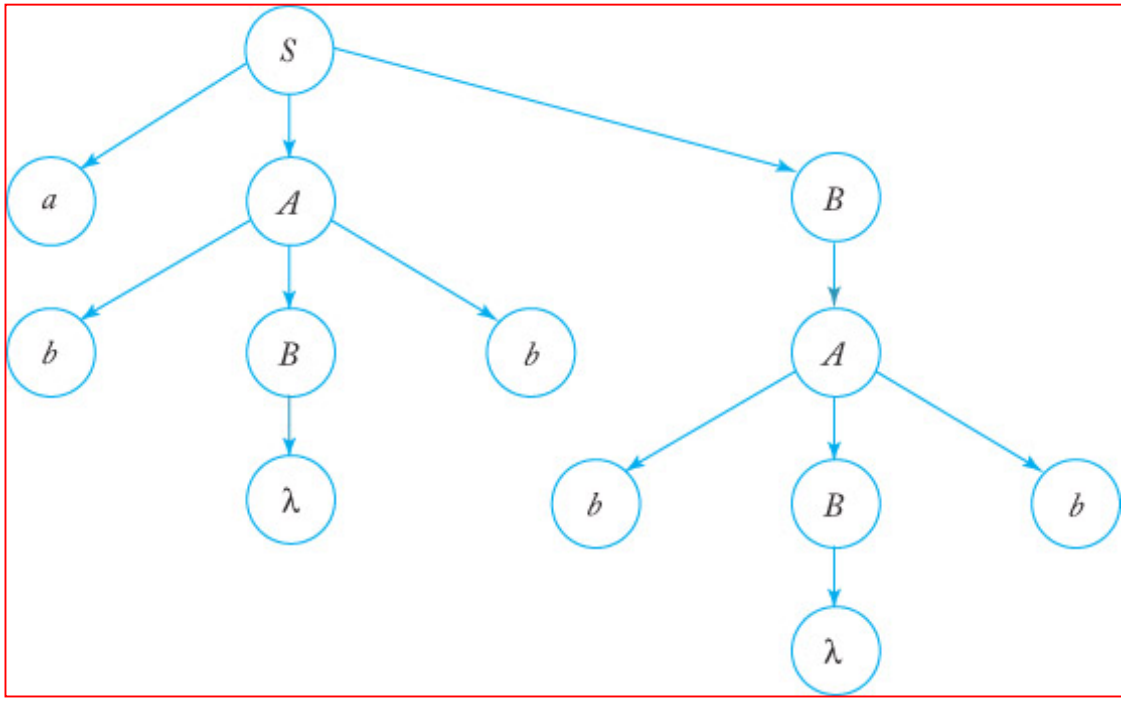
$$S \rightarrow aAB \qquad A \rightarrow bBb \qquad B \rightarrow A \mid \lambda$$

Yield: abBbB is a sentential form of G

abbbb ϵ L(G)



Partial derivation tree



Derivation tree

Theorem 5.1

- Let G = (V, T, S, P) be a CFG. Then for every w ϵ 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

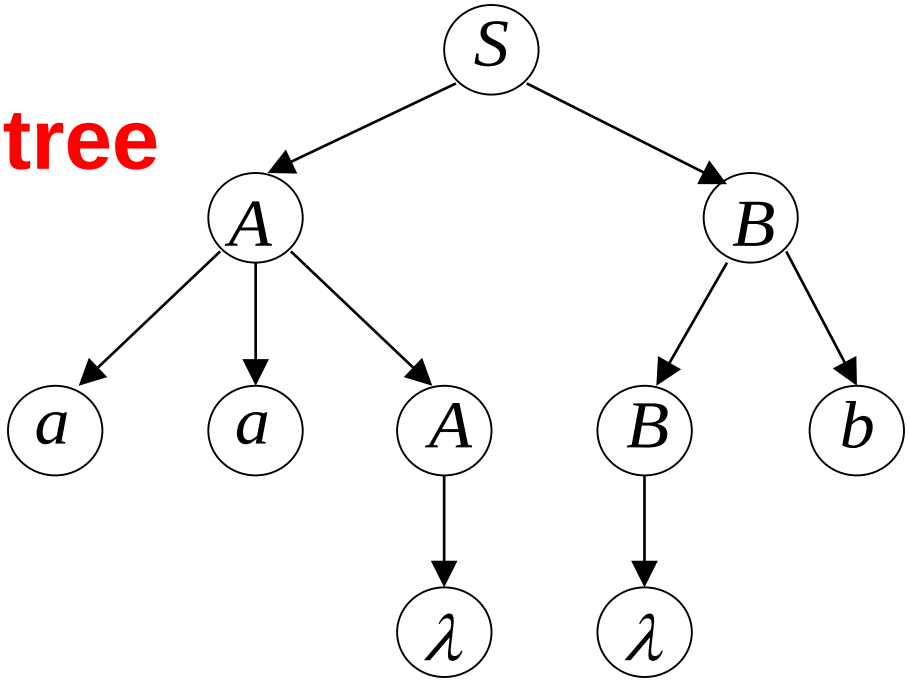
Leftmost:

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

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

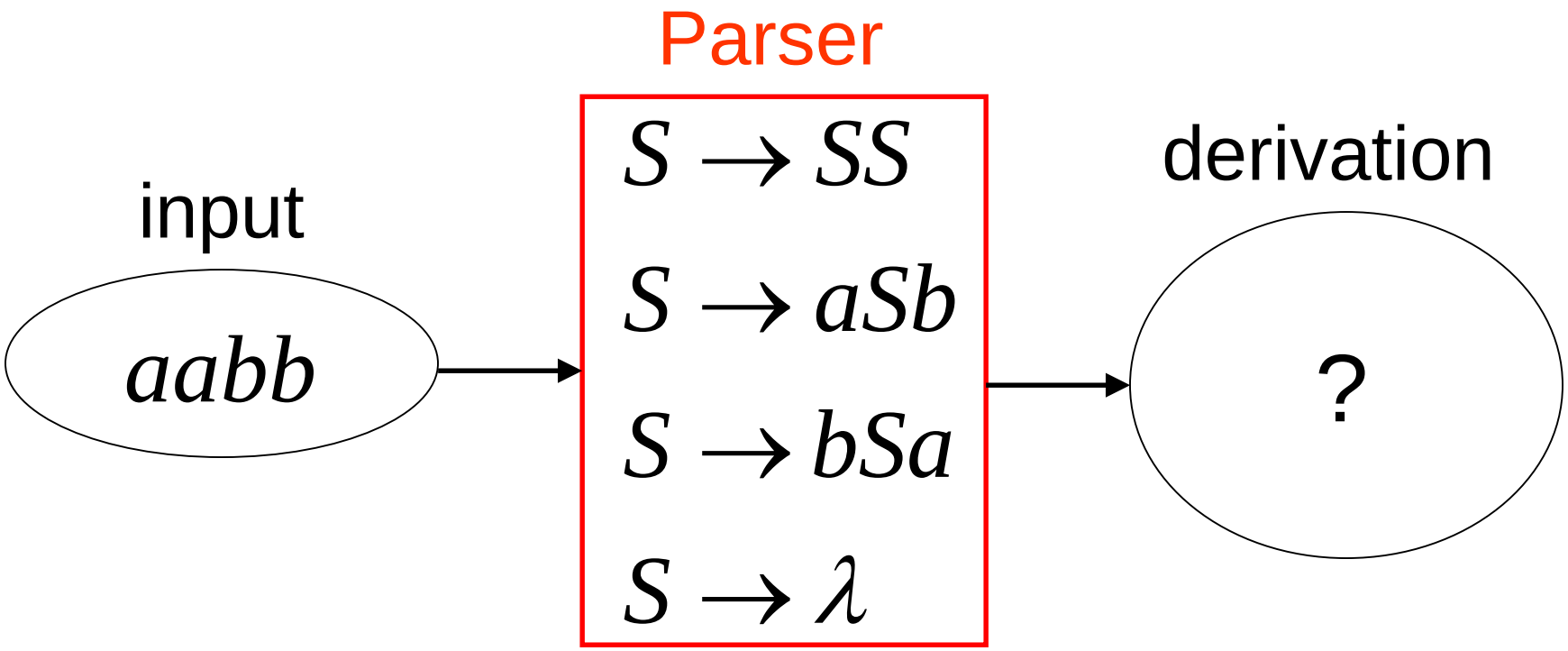
Given a grammar G, we studied the set of strings that can be derived using G, but…

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 ϵ L(G) is derived.


Parser
input
string
grammar
derivation

Example:



Exhaustive Search Parsing
(Brute Force Parsing)

Example 5.7

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

Find derivation of

$aabb$

Phase 1:

$S \Rightarrow SS$

$S \Rightarrow aSb$

~~$S \Rightarrow bSa$~~

~~$S \Rightarrow \lambda$~~

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

$S \Rightarrow SS \Rightarrow SSS$

$S \Rightarrow SS \Rightarrow aSbS$

$aabb$

Phase 1

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

$S \Rightarrow SS$

$S \Rightarrow SS \Rightarrow S$

$S \Rightarrow aSb$

$S \Rightarrow aSb \Rightarrow aSSb$

$S \Rightarrow aSb \Rightarrow aaSbb$

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

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

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

Phase 2

$$S \Rightarrow SS \Rightarrow SSS$$

$$S \Rightarrow SS \Rightarrow aSbS$$

$$S \Rightarrow SS \Rightarrow S$$

$$aabb$$

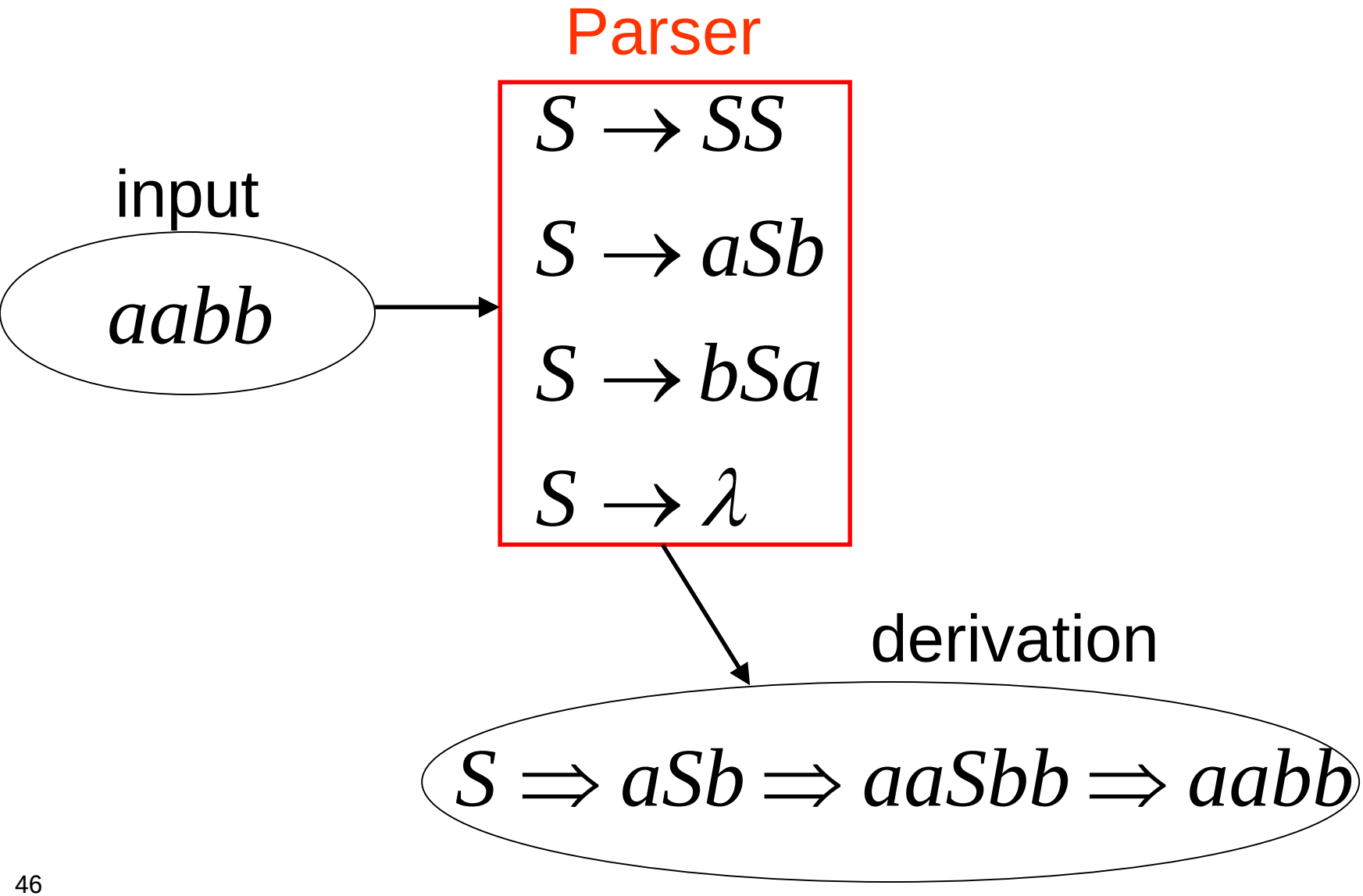
$$S \Rightarrow aSb \Rightarrow aSSb$$

$$S \Rightarrow aSb \Rightarrow aaSbb$$

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?

$$S \to SS$$
$$S \to aSb$$
$$S \to bSa$$
$$S \to \lambda$$

$$A \to \lambda$$
$$A \to B$$

若有這種可能則parsing的string 還有可能會變短(A->lambda)，
很容易在檢測一段string時不停parsing，不會停止

Example 5.8

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

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

If so, given any w ϵ {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

$$A \to \lambda$$

$$A \to B$$

Then the exhaustive search parsing can be made into an algorithm which, for any w ϵ ∑*, 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

$$A \to \lambda$$

$$A \to B$$

∵ Neither the length of a sentential form nor the number of terminals can exceed |w|

∴ Number of phases for string w can not more than: 2|w|

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

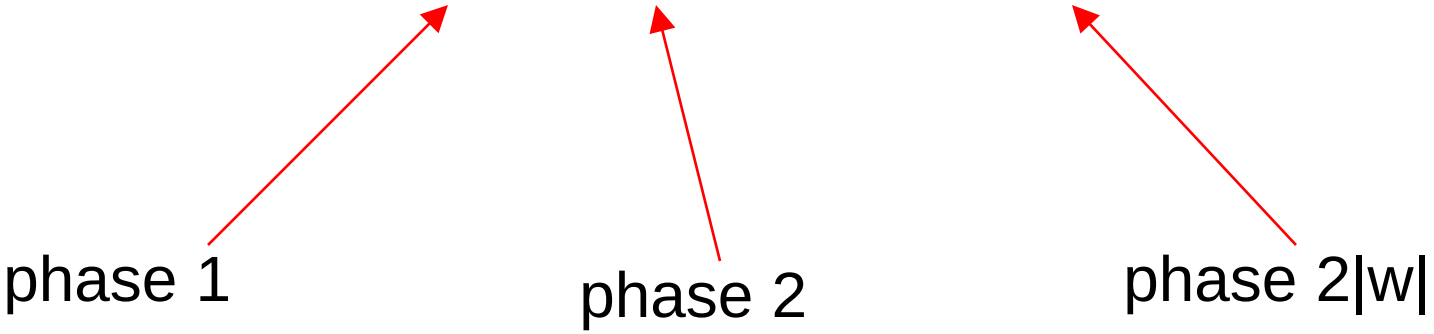
Phase 2:
we have no more than $|P|^2$ sentential forms

⋮

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

Total time needed for string : w

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



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

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):

$$A \to ax$$

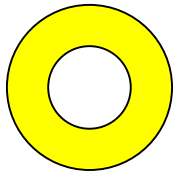
terminal

string of variables

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

Example 5.9

$$S \to aS$$
$$S \to bSS$$
$$S \to c$$



$$S \to aS$$
$$S \to bSS$$
$$S \to aSS$$
$$S \to c$$

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!!!

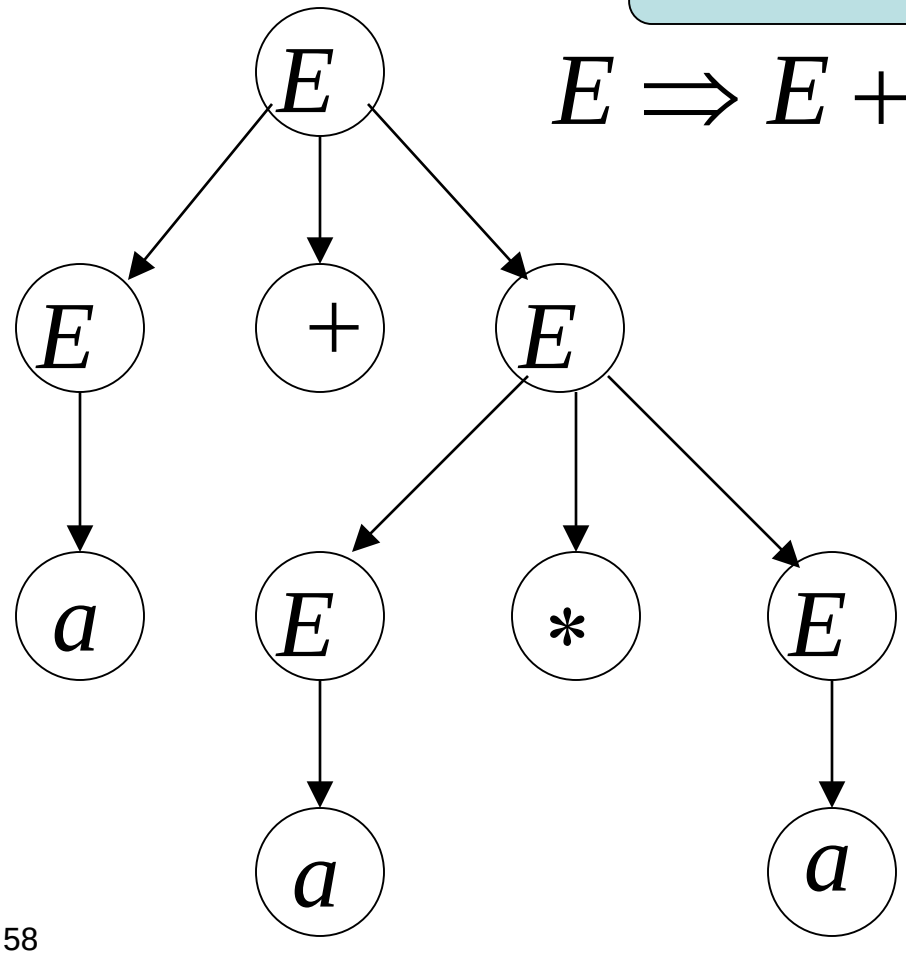
<while_stmt> ::= while <expr><stmt>

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

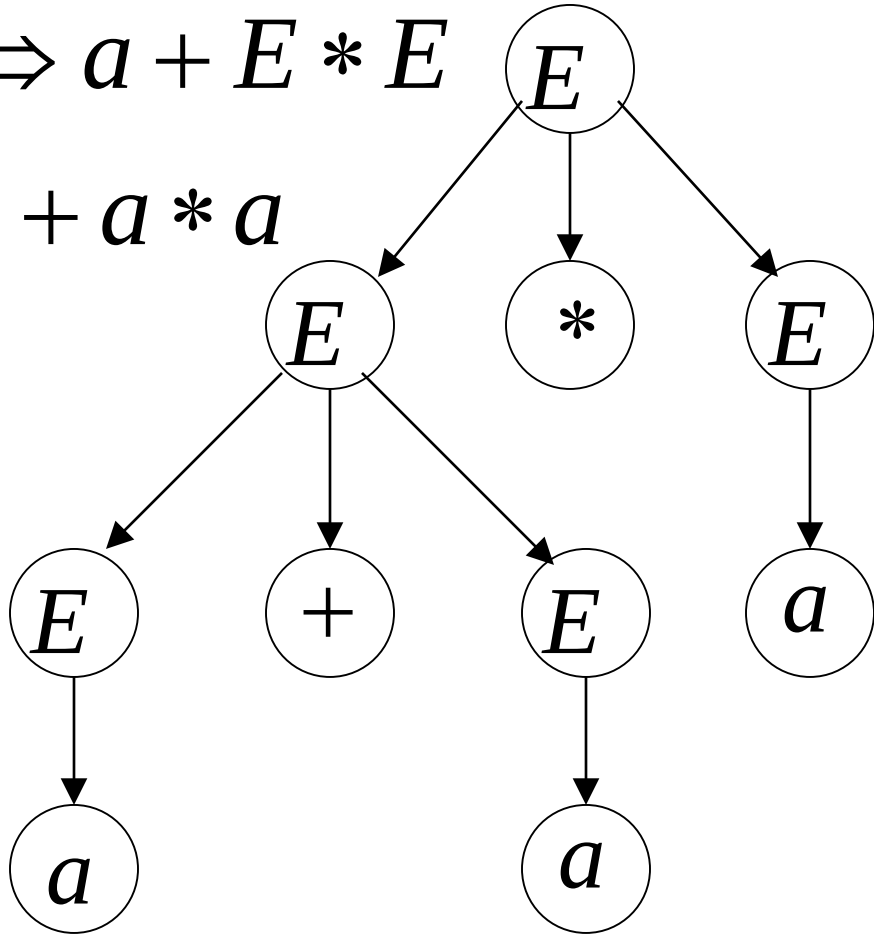
$$E \to E + E \mid E * E \mid (E) \mid a$$

$$a + a * a$$

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

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

leftmost derivation

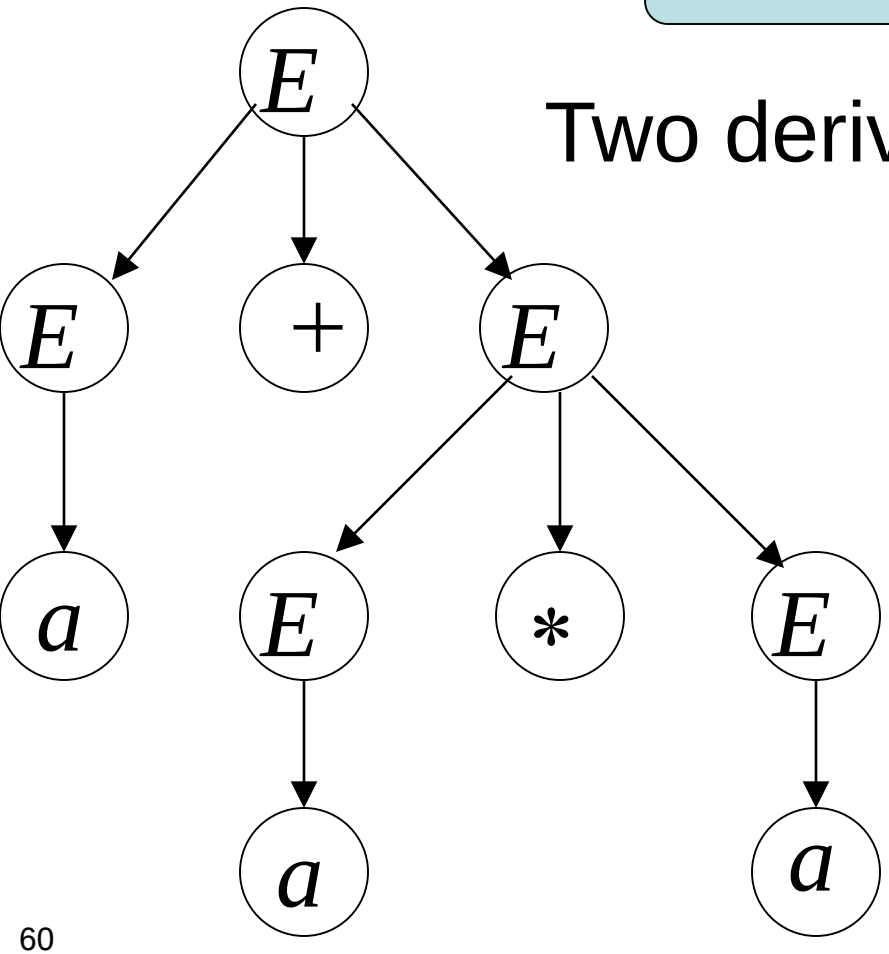


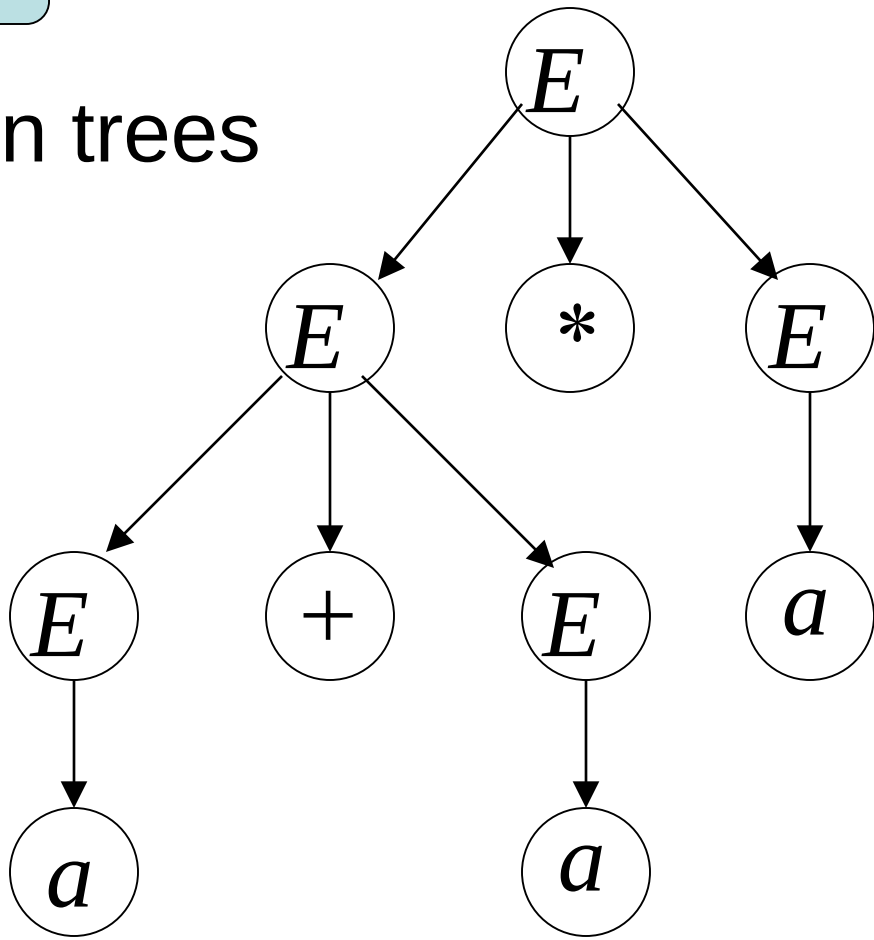
Example 5.11

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

$$a + a * a$$

Two derivation trees

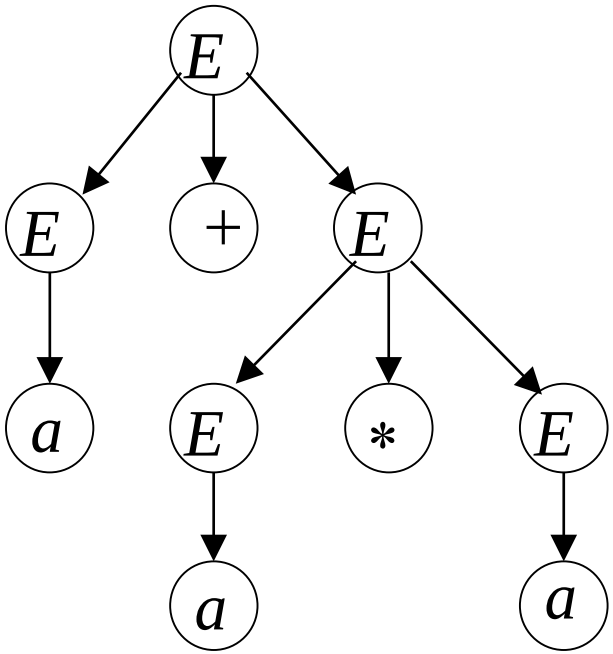


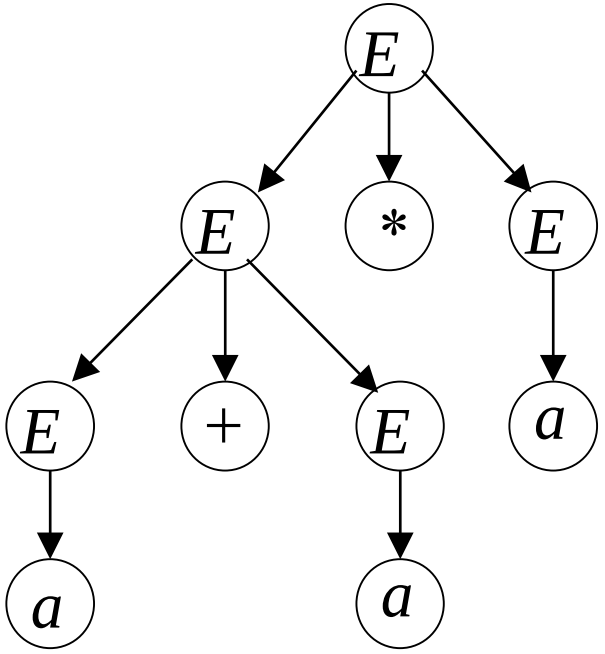


The grammar $E \rightarrow E + E \mid E * E \mid (E) \mid a$

is ambiguous:

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





The grammar $E \to E + E \mid E * E \mid (E) \mid a$

is ambiguous:

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

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

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

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:

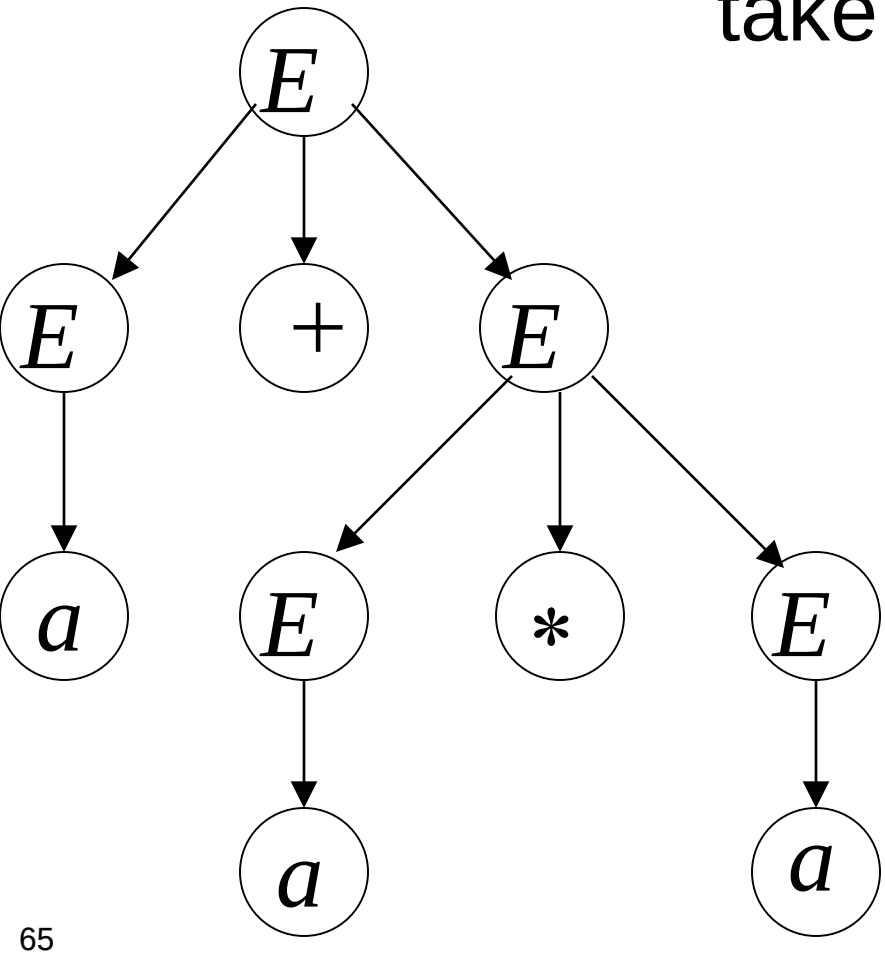
two or more leftmost derivations

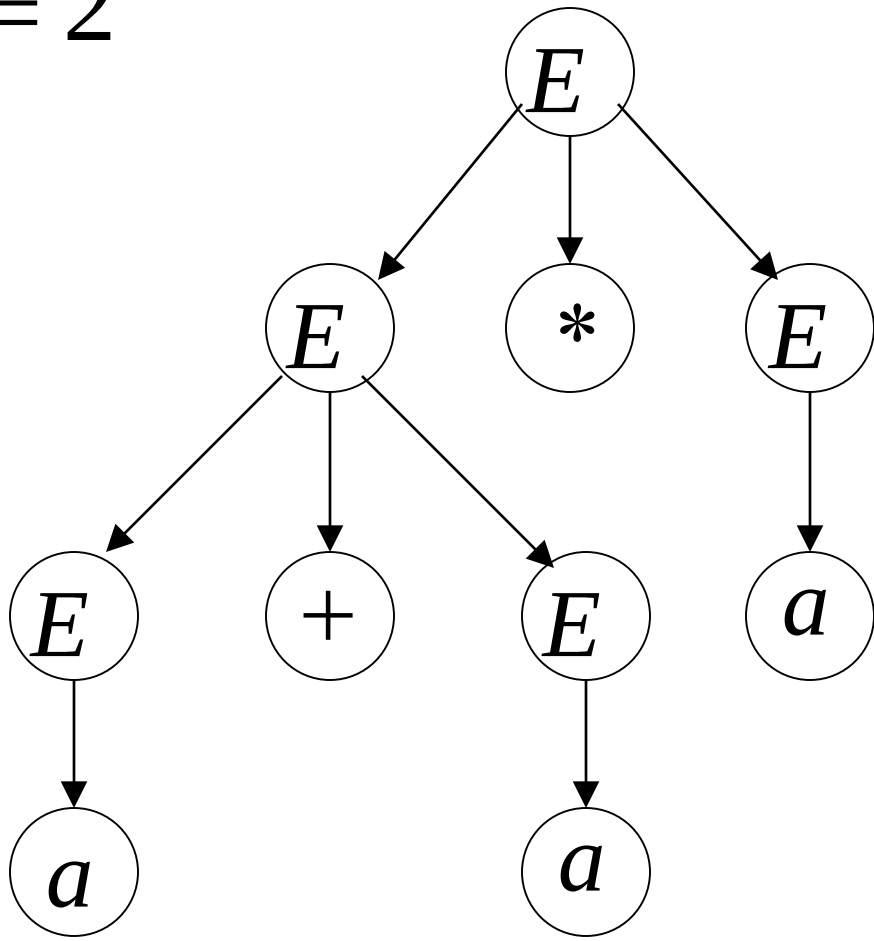
(or rightmost)

Why do we care about ambiguity?

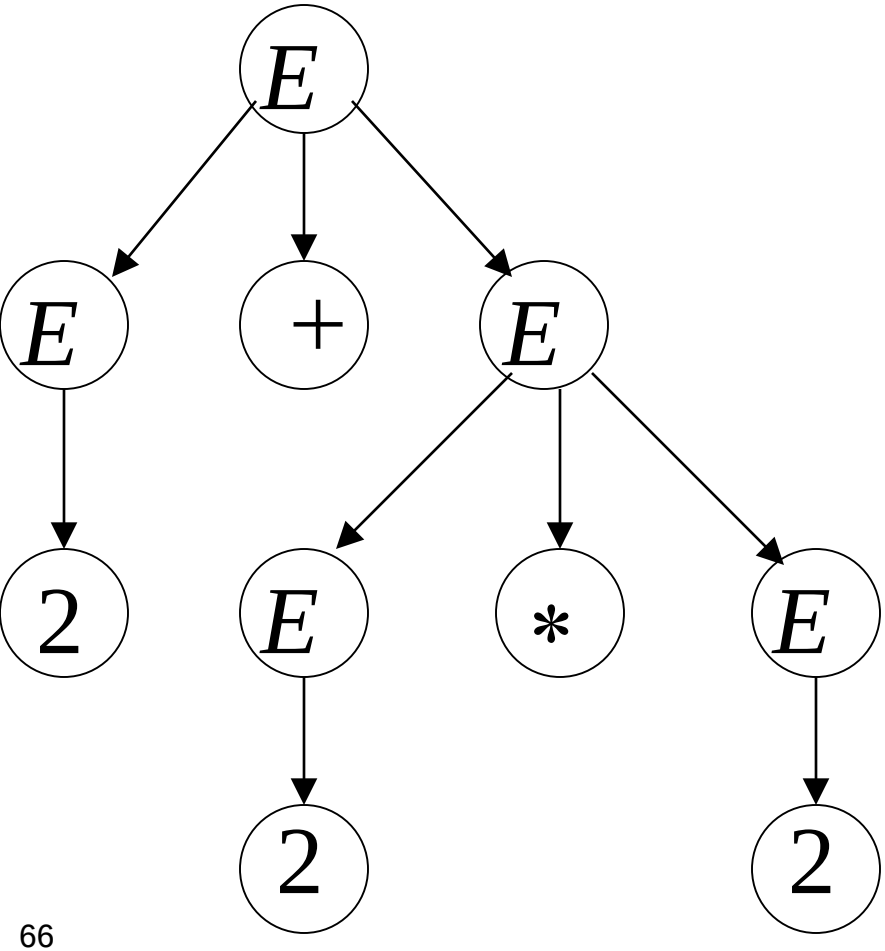
$$a + a * a$$

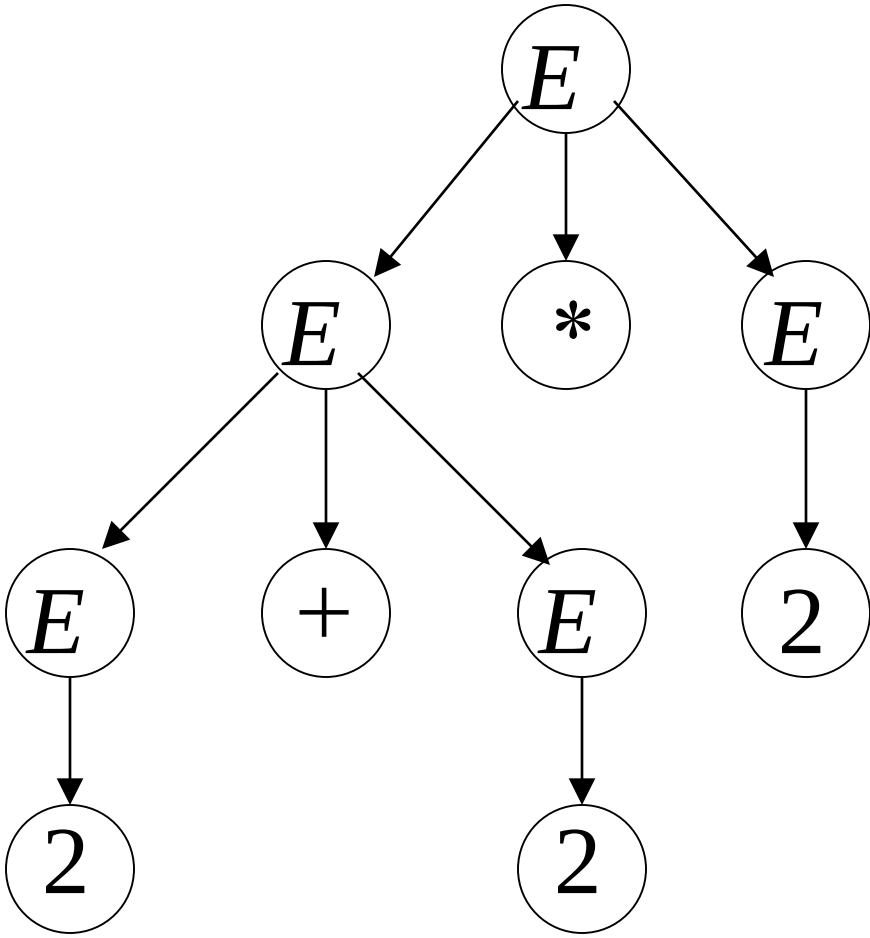
take $a = 2$



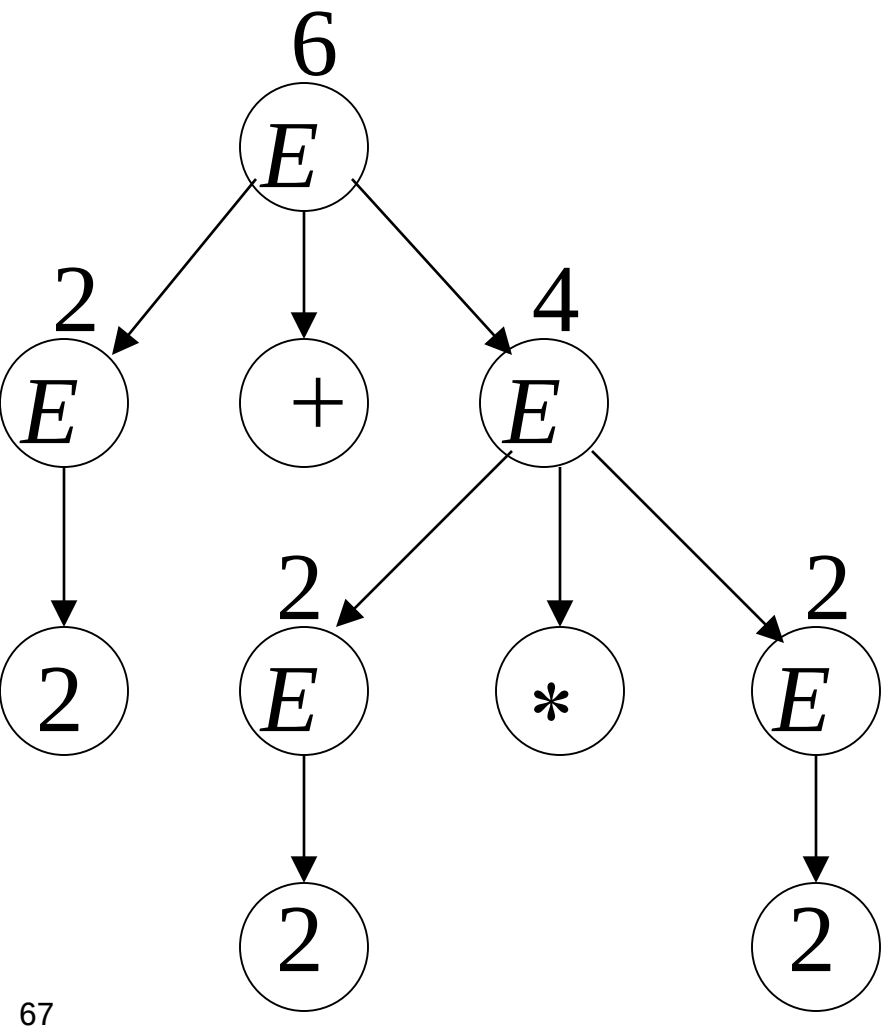


$$2 + 2 * 2$$

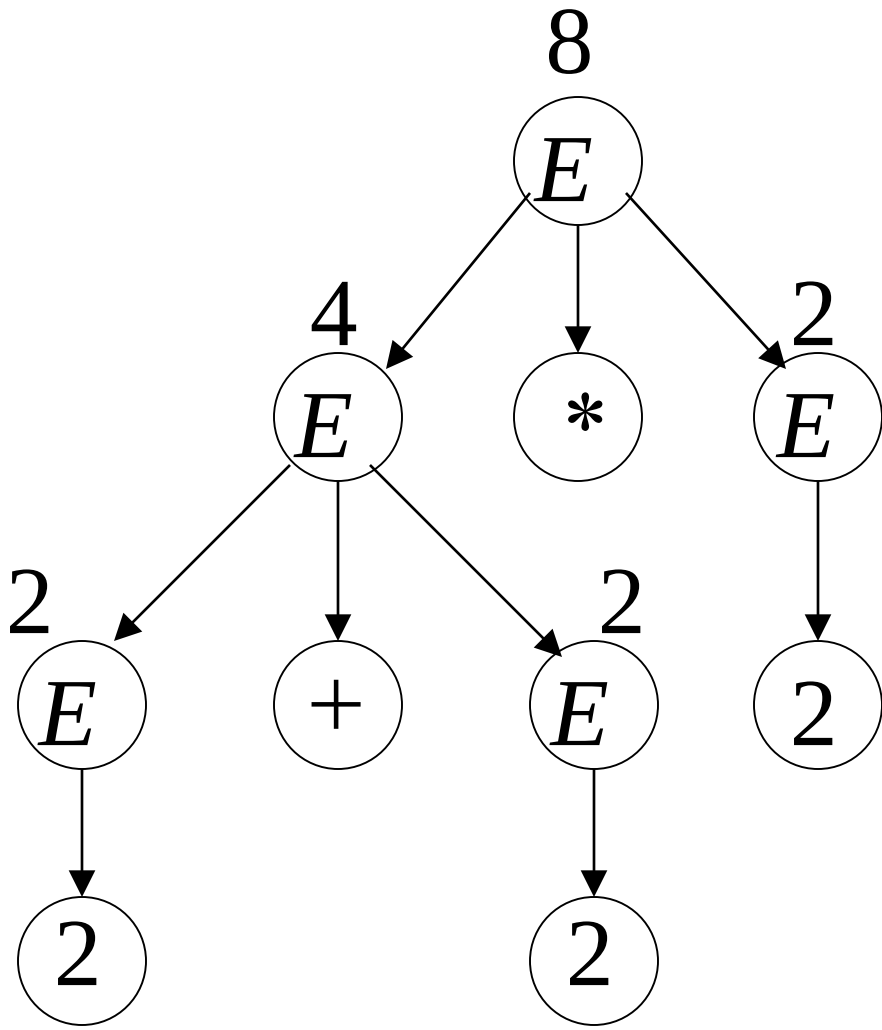




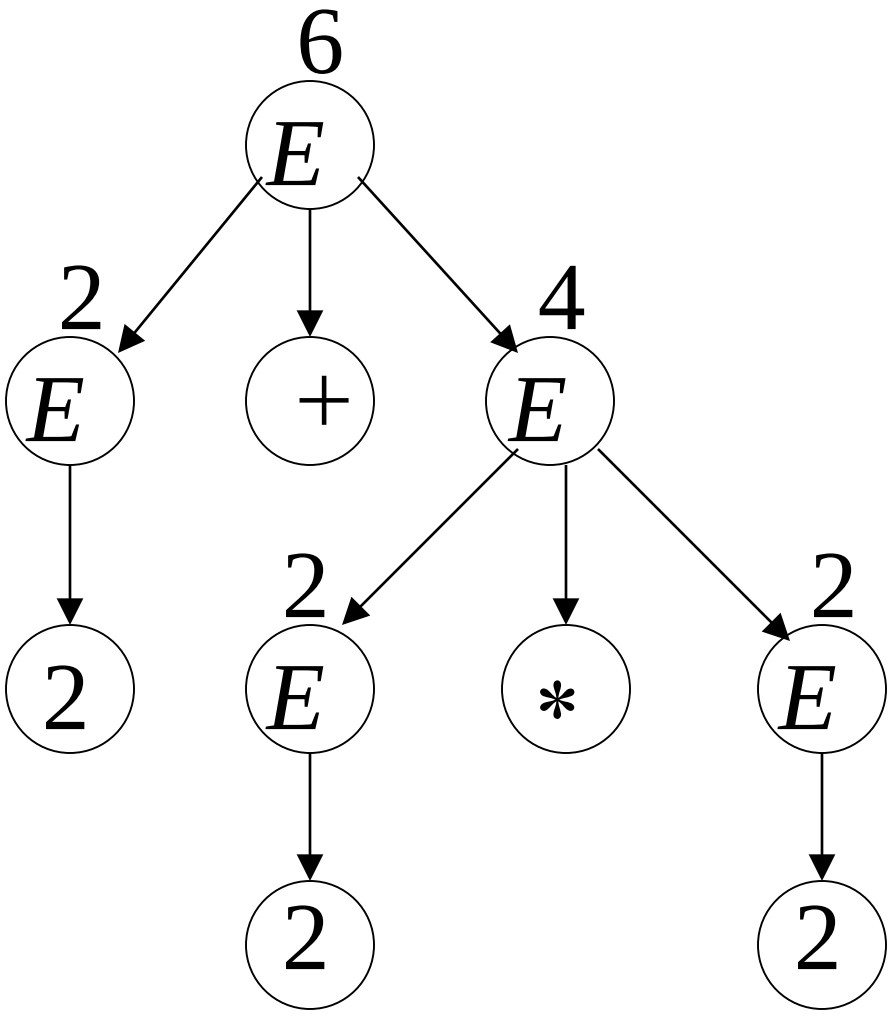
$2+2*2=6$



$2+2*2=8$



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} ϵ V

priority

$$E \rightarrow E + T$$

$$E \rightarrow T$$

$$T \rightarrow T * F$$

$$T \rightarrow F$$

$$F \rightarrow (E)$$

$$F \rightarrow a$$

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

$E \rightarrow E+T$

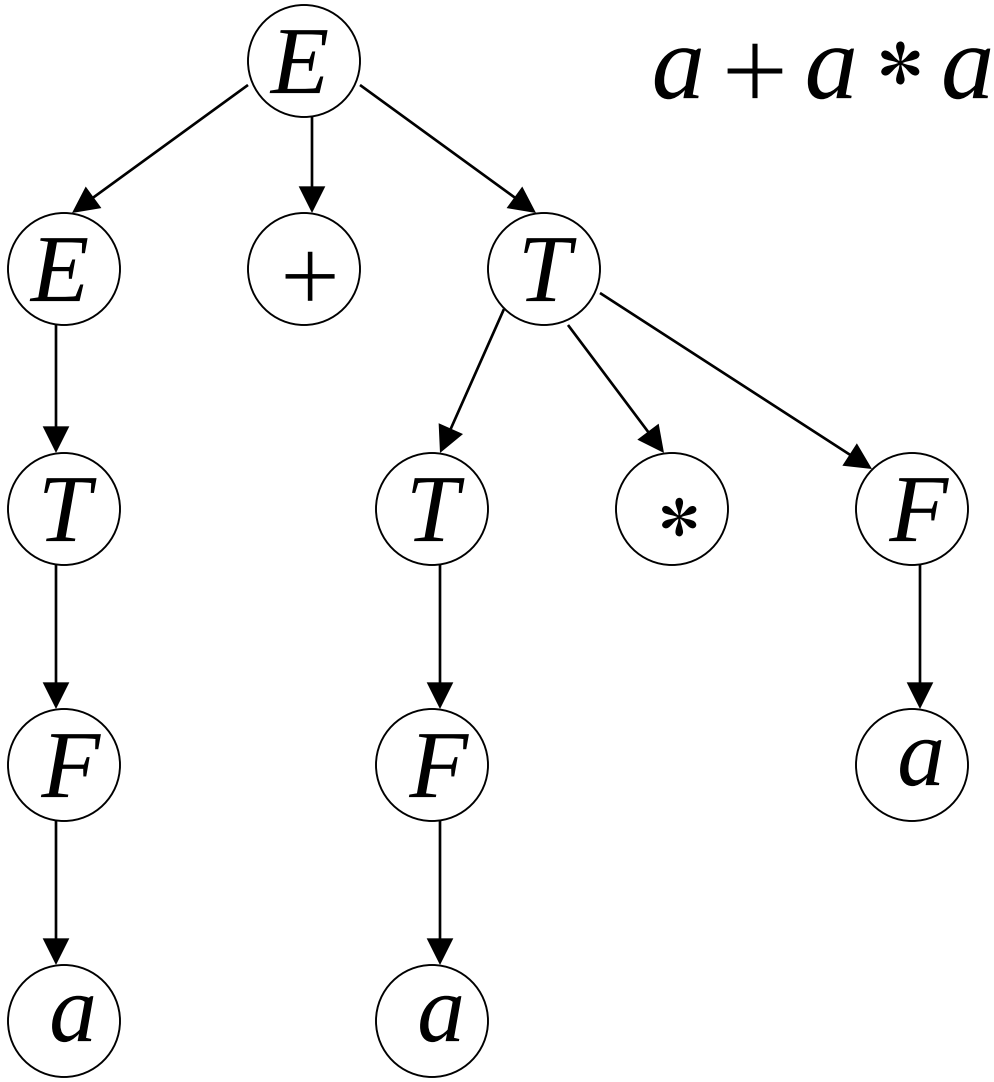
$E \rightarrow T$

$T \rightarrow T*F$

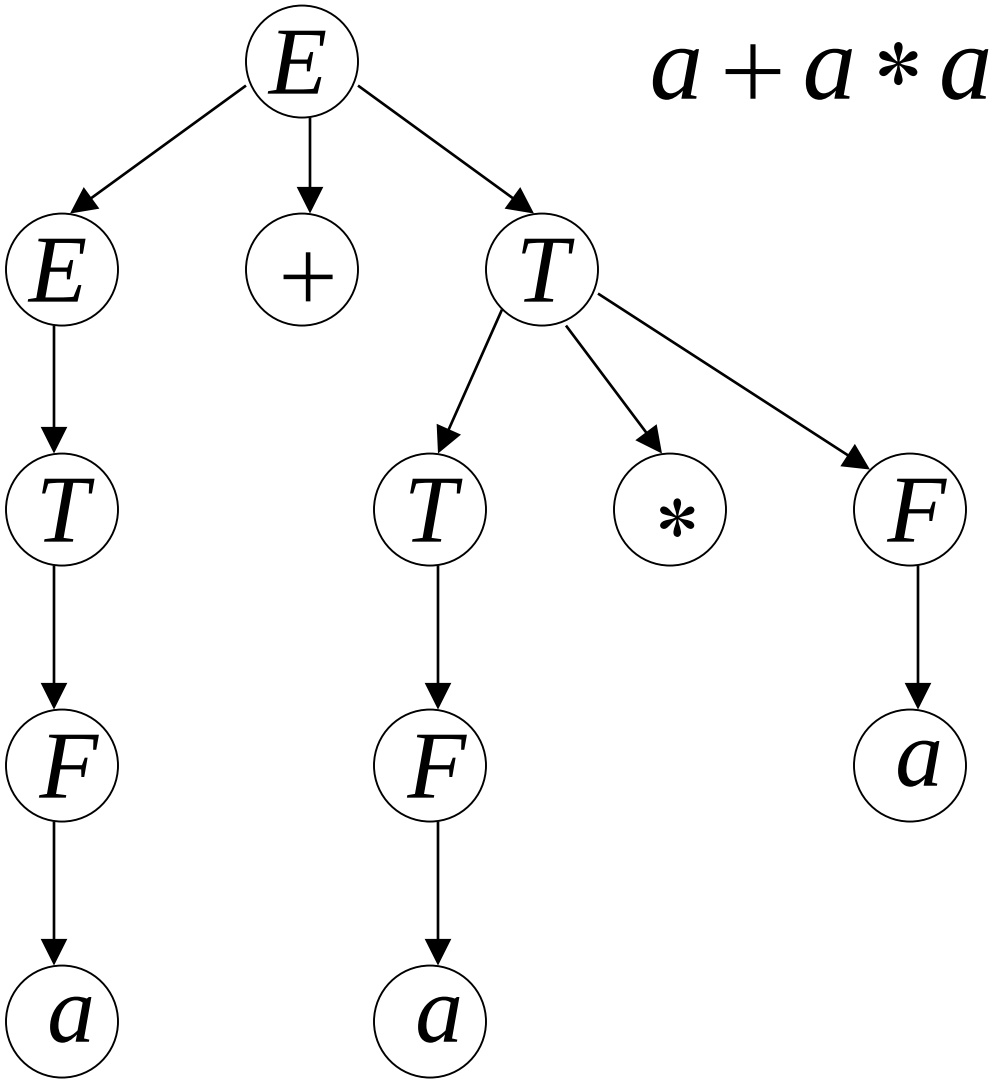
$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow a$



Unique derivation tree



The grammar: G

$$E \to E + T$$

$$E \to T$$

$$T \to T * F$$

$$T \to F$$

$$F \to (E)$$

$$F \to a$$

is unambiguous:

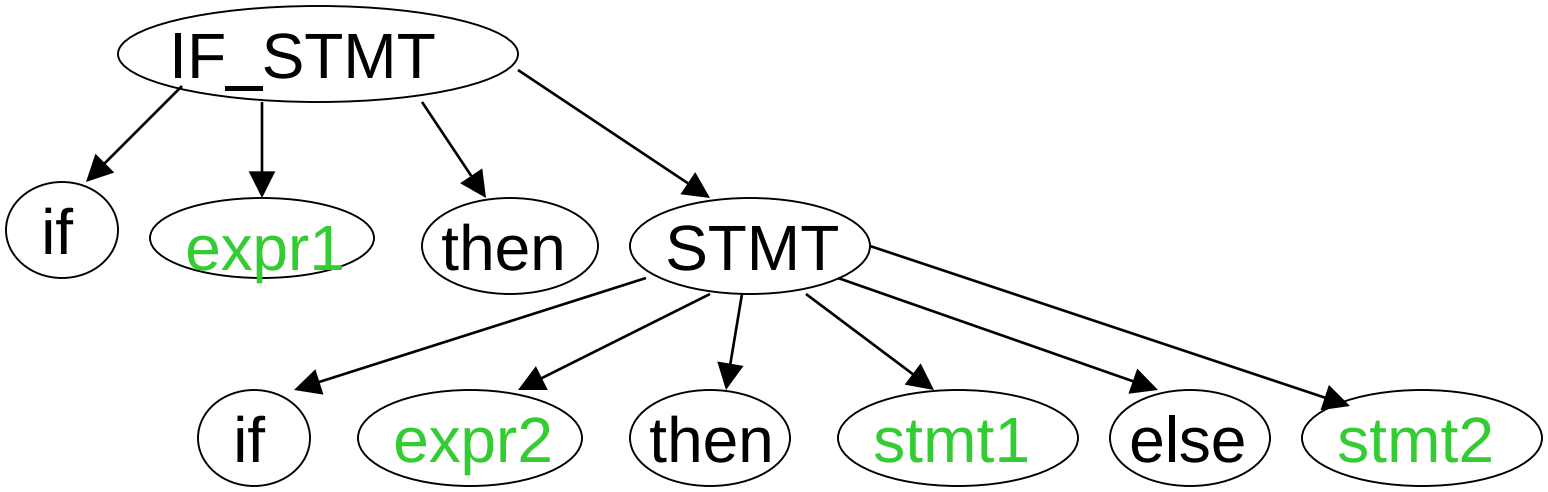
Every string $w \in L(G)$ has a unique derivation tree

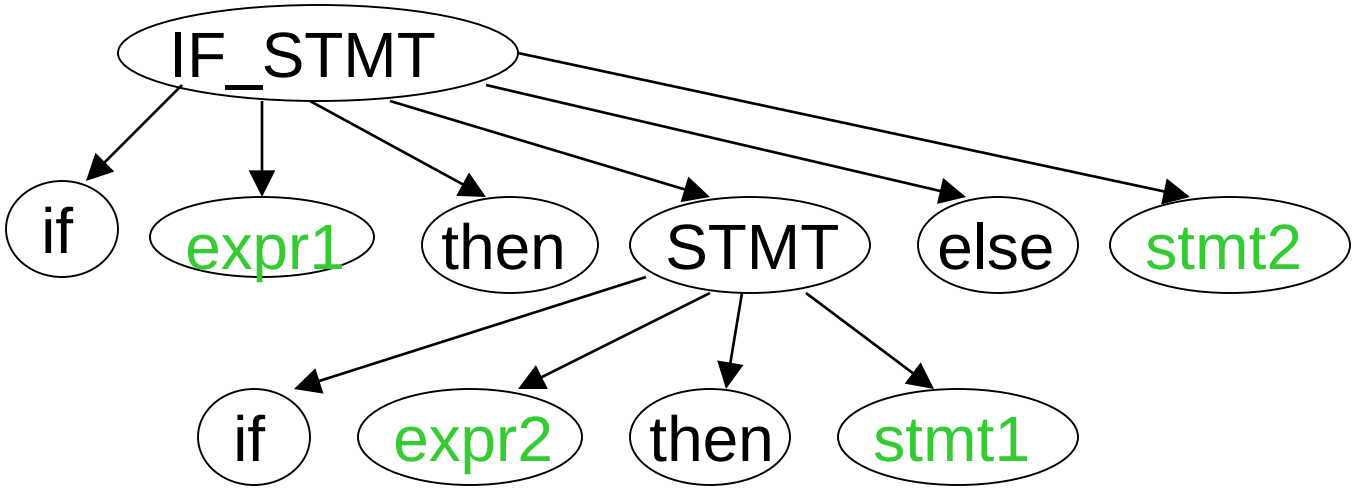
Another Ambiguous Grammar (Dangling Else)

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

$S \to S_1 \mid S_2$

$S_1 \to S_1 c \mid A$

$A \to aAb \mid \lambda$

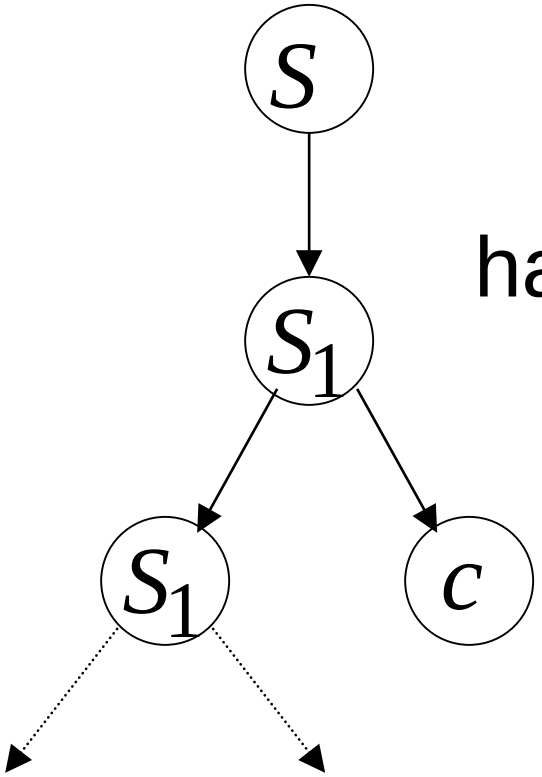
$S_2 \to aS_2 \mid B$

$B \to bBc \mid \lambda$

$$S \to S_1 \mid S_2$$

$$S_1 \to S_1 c \mid A \qquad S_2 \to a S_2 \mid B$$

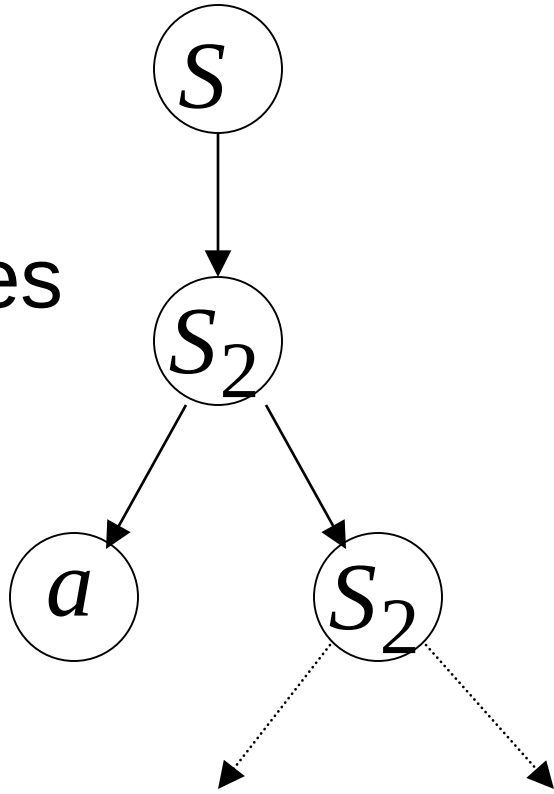
$$A \to aAb \mid \lambda \qquad B \to bBc \mid \lambda$$



$$a^n b^n c^n$$

has two derivation trees

Proof by
Harrison, 1978



Exercise 5.3.6

- Show that the following grammar is ambiguous.

$$S \to AB \mid aaB$$

$$A \to a \mid Aa$$

$$B \to b$$

There are two leftmost derivations for w = aab

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

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

Outline

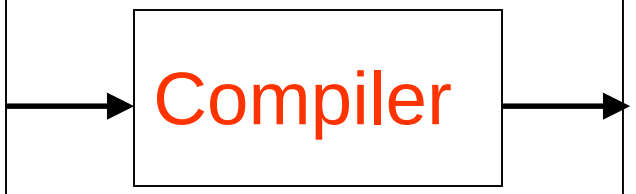
Context-Free Grammars

Parsing and Ambiguity

Context-Free Grammars and Programming Languages

Program

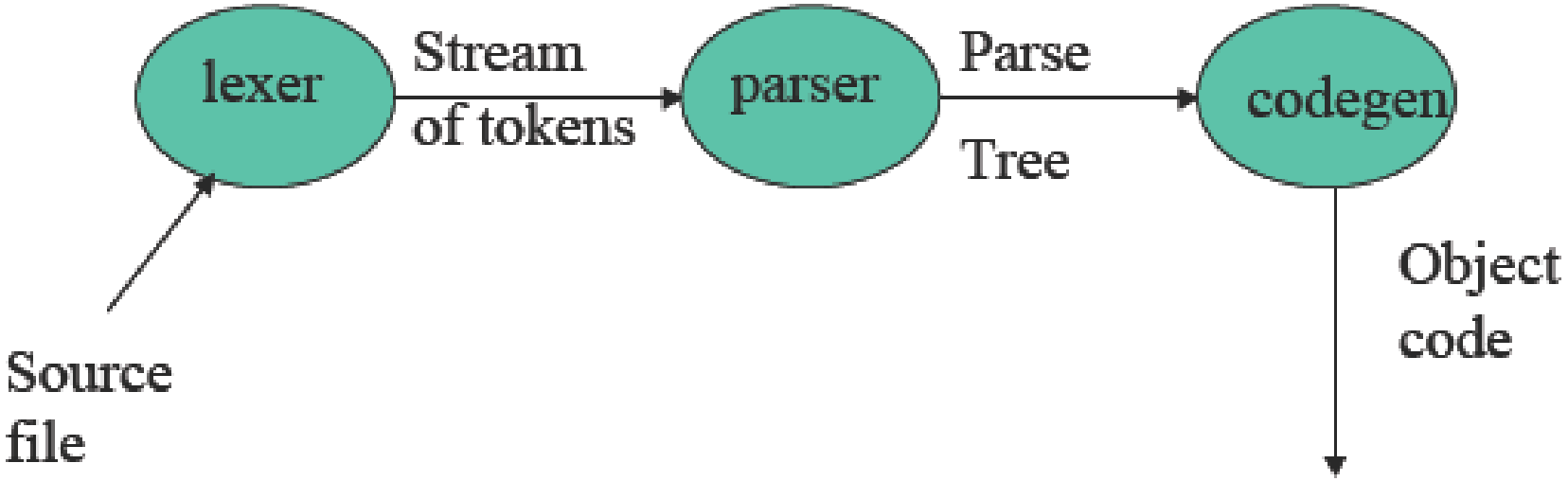
```
v = 5;
if (v>5)
   x = 12 + v;
while (x !=3) {
  x = x - 3;
  v = 10;
}
......
```



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

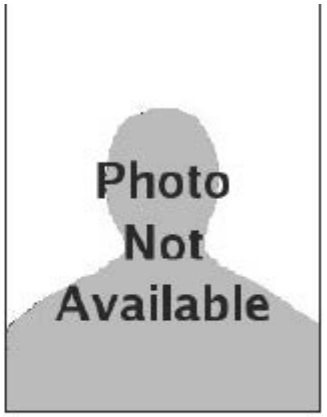
Regular expressions in

UNIX / grep / vi

Eric E. Schmidt

Mike Lesk

lex



Stephen C Johnson

yacc

A parser knows the grammar
of the programming language

Parser

PROGRAM → STMT_LIST

STMT_LIST → STMT; STMT_LIST | STMT;

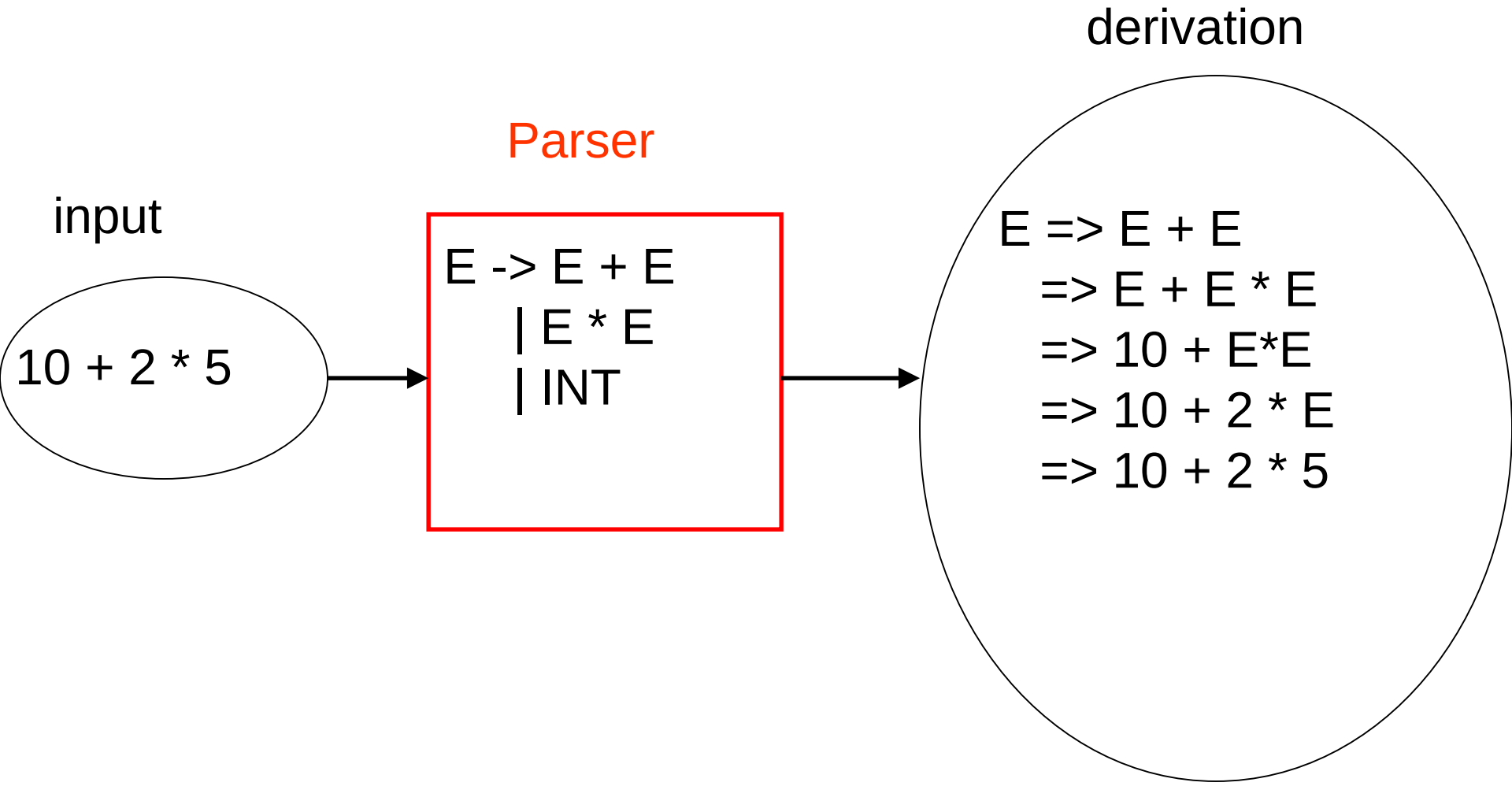
STMT → EXPR | IF_STMT | WHILE_STMT
| { STMT_LIST }

EXPR → EXPR + EXPR | EXPR - EXPR | ID

IF_STMT → if (EXPR) then STMT
| if (EXPR) then STMT else STMT

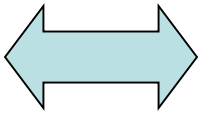
WHILE_STMT → while (EXPR) do STMT

The parser finds the derivation of a particular input

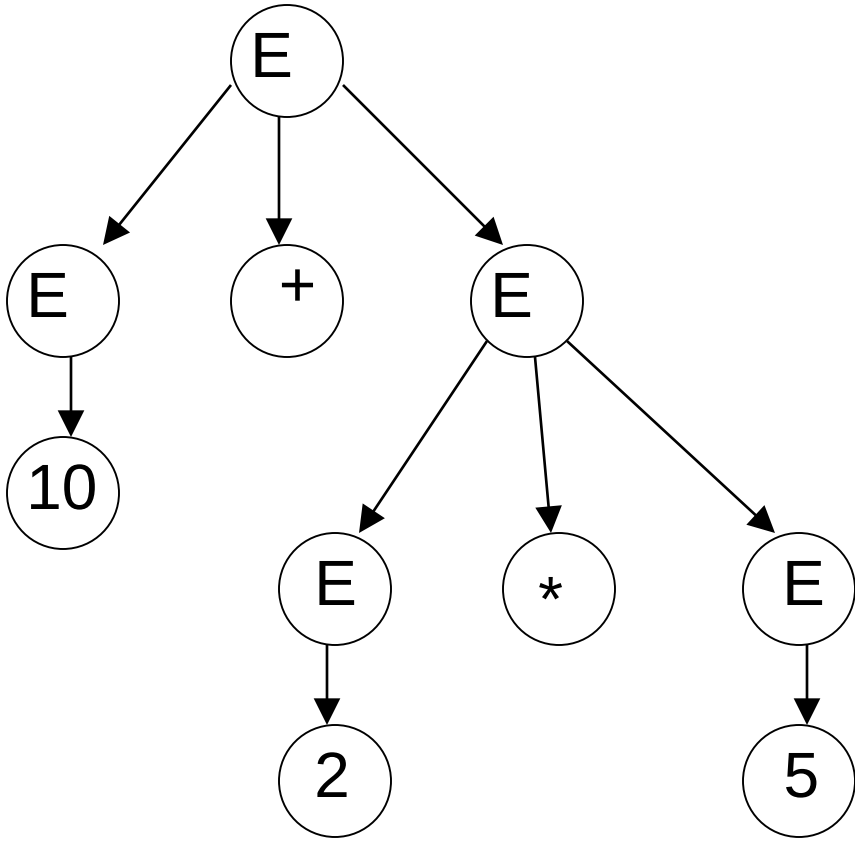


derivation

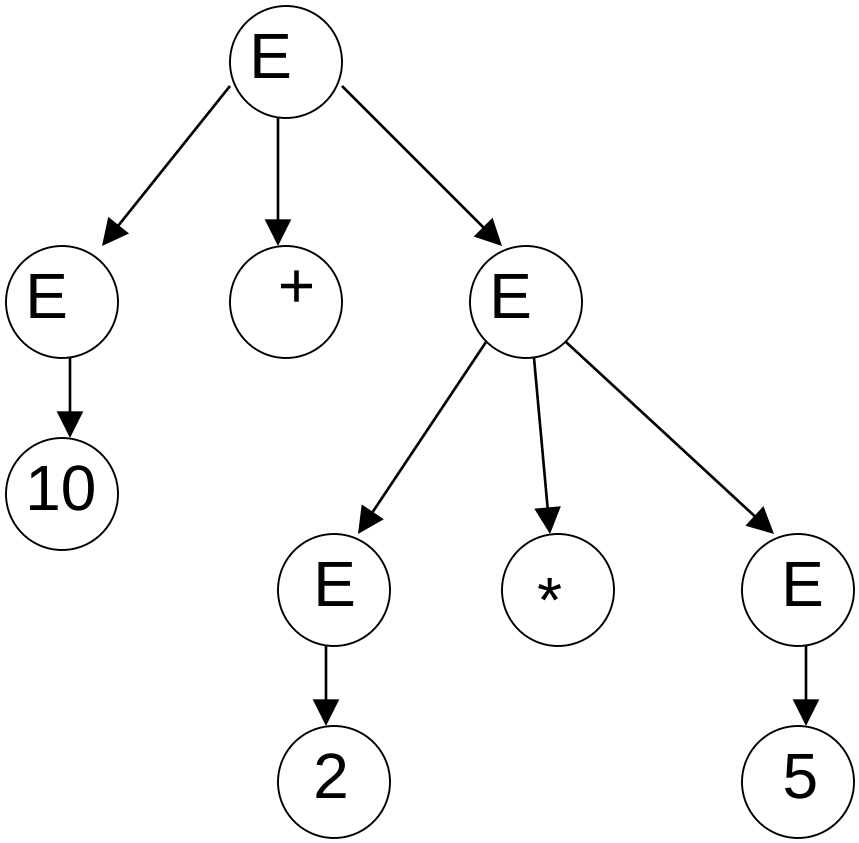
E => E + E
=> E + E * E
=> 10 + E*E
=> 10 + 2 * E
=> 10 + 2 * 5



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:
    <expr> ::= <term> | <expr> + <term>
    <term> ::= <factor> | <term> * <factor>
- LL and LR grammar: parse in linear time

Questions?