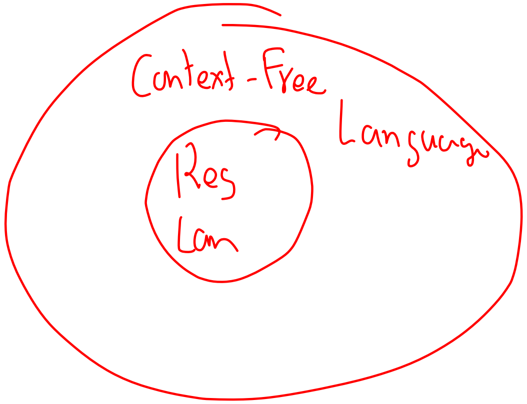
ออกมาจากภาษา regular แล้ว

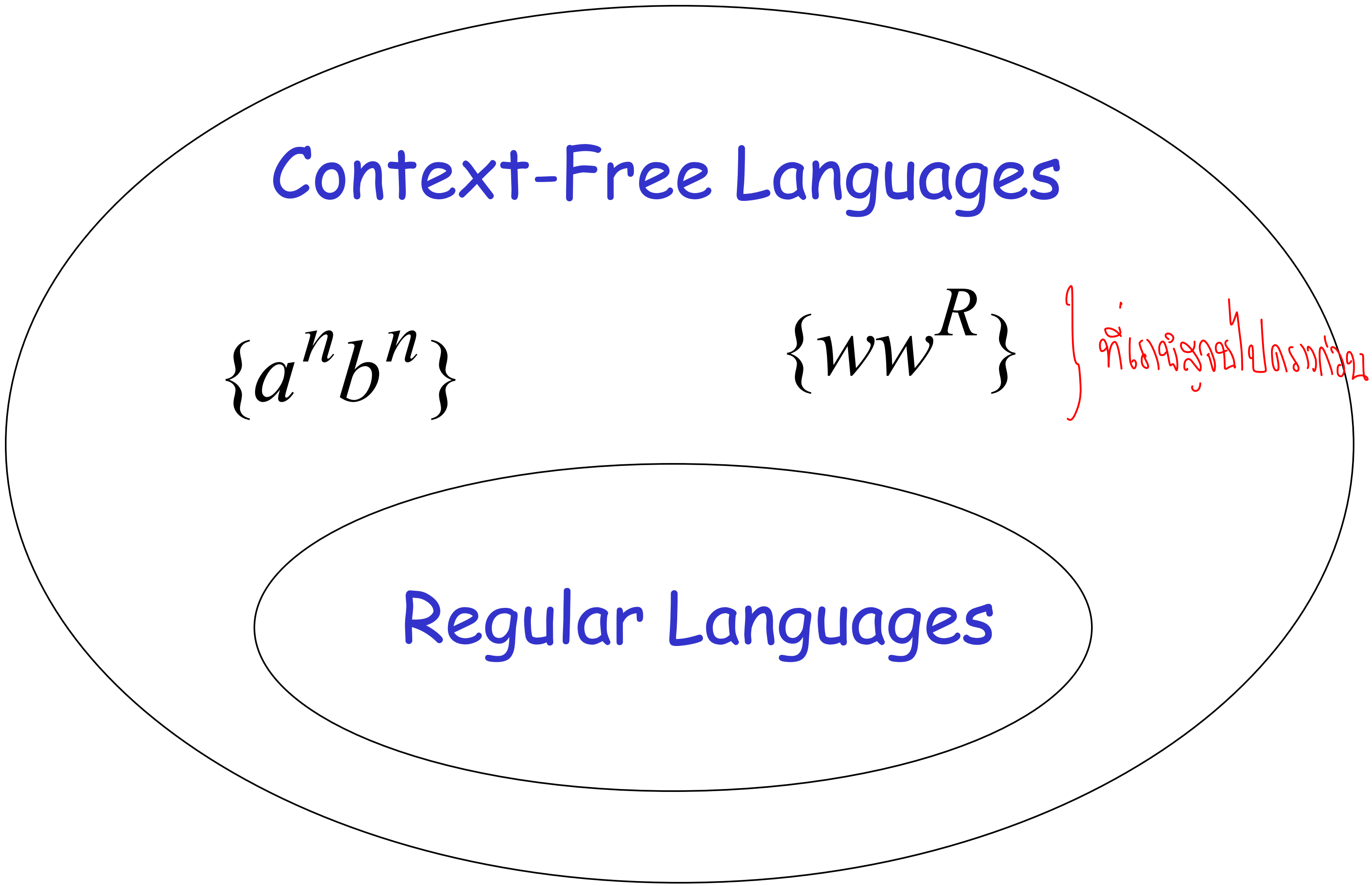
Context-Free Languages



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

Regular Languages

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


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

Context-Free Languages

เรียน ๒ อัน

Context-Free Grammars

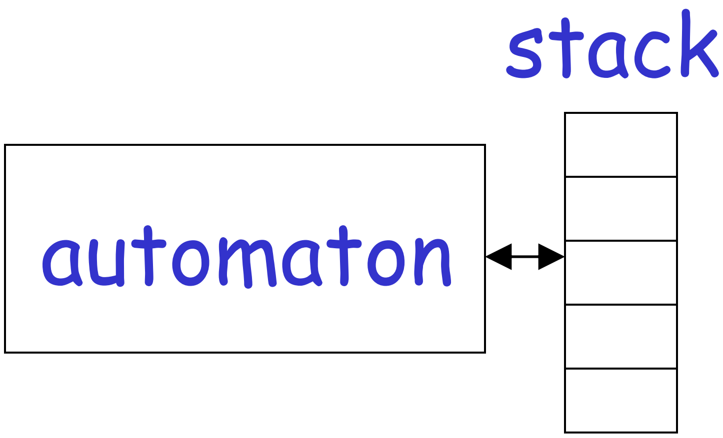
Pushdown Automata

RL

Reg Grammar

Automata

DFA NFA



Context-Free Grammars

Example

ตัวอย่าง

A context-free grammar G:

$$S \rightarrow aSb$$

$$S \rightarrow \lambda$$

ไม่ใช่ reg → left, right linear gramma only

A derivation:

ภาษาของมัน $\{a^n b^n : n \geq 0\}$

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

A context-free grammar G:

$$S \to aSb$$

$$S \to \lambda$$

Another derivation:

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

$$S \to aSb$$

$$S \to \lambda$$

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

ถ้าแทน a ด้วย (, b ด้วย)

การbalance วงเล็บ

Describes parentheses: (((( ))))

ข้องเกิด (()())

Example

ภาษาไม่ธรรมดา

A context-free grammar G:

$$S \to aSa$$

เหมือนกระจก

$$S \to bSb$$

aab baa

$$S \to \lambda$$

A derivation:

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

A context-free grammar G:

$$S \rightarrow aSa$$

$$S \rightarrow bSb$$

$$S \rightarrow \lambda$$

Another derivation:

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

$$S \to aSa$$

$$S \to bSb$$

$$S \to \lambda$$

วนไปวนมา

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

Example

A context-free grammar G:

$$S \to aSb$$

$$S \to SS$$

$$S \to \lambda$$

A derivation:

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

A context-free grammar G:

$$S \to aSb$$

$$S \to SS$$

$$S \to \lambda$$

A derivation:

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

$$S \to aSb$$

$$S \to SS$$

$$S \to \lambda$$

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

ก็ต้องตามจังหวะ prefix

$$\text{and } n_a(v) \geq n_b(v)$$

$$\text{in any prefix } v\}$$

v เป็น prefix ได้

Describes matched parentheses:

แทน a ด้วย (, b ด้วย)

อันนี้แม่งดูยากมาก

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

ภาษาโปรแกรมการเขียนโค้ด

Definition: Context-Free Grammars

เหมือนใน regular grammar

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

Variables

Terminal symbols

Start variable

Productions of the form:

"context-free"

$A \rightarrow x$ $\rightarrow \in \{V, T\}^* \rightarrow$ มันผสมเป็นอะไรก็ได้

Variable

String of variables and terminals

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

นิยาม set ของ strings w

โดยที่ derive กี่ครั้งก็ได้ จะได้ w อยู่ใน T*

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

Definition: Context-Free Languages

A language L is context-free

if and only if

there is a context-free grammar G
with $L = L(G)$

language L ใดๆจะเป็น context-free language ก็ต่อเมื่อเราสามารถสร้าง
context free grammar G ให้มันได้

Derivation Order

ลำดับของการทำ derivation

1. $S \to AB$

2. $A \to aaA$

3. $A \to \lambda$

4. $B \to Bb$

5. $B \to \lambda$

แบ่งเป็น

①

Leftmost derivation:

P P

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

มองจากทาง left เข้ามา derive variable ที่เจอตัวแรก

②

Rightmost derivation:

มองจากทางขวา

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

$$S \to aAB$$ อันนี้แค่ใช้ให้ดู

$$A \to bBb$$

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

Leftmost derivation:

$$S \Rightarrow aAB \Rightarrow abBbB \Rightarrow abAbB \Rightarrow abbBbbB$$
$$\Rightarrow abbbbB \Rightarrow abbbb$$

Rightmost derivation:

$$S \Rightarrow aAB \Rightarrow aA \Rightarrow abBb \Rightarrow abAb$$
$$\Rightarrow abbBbb \Rightarrow abbbb$$

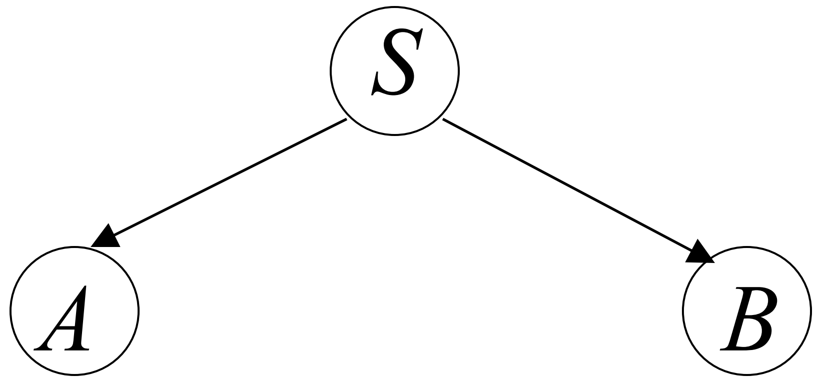
Derivation Trees

Parse Trees

เคยสอนไปแล้ว

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

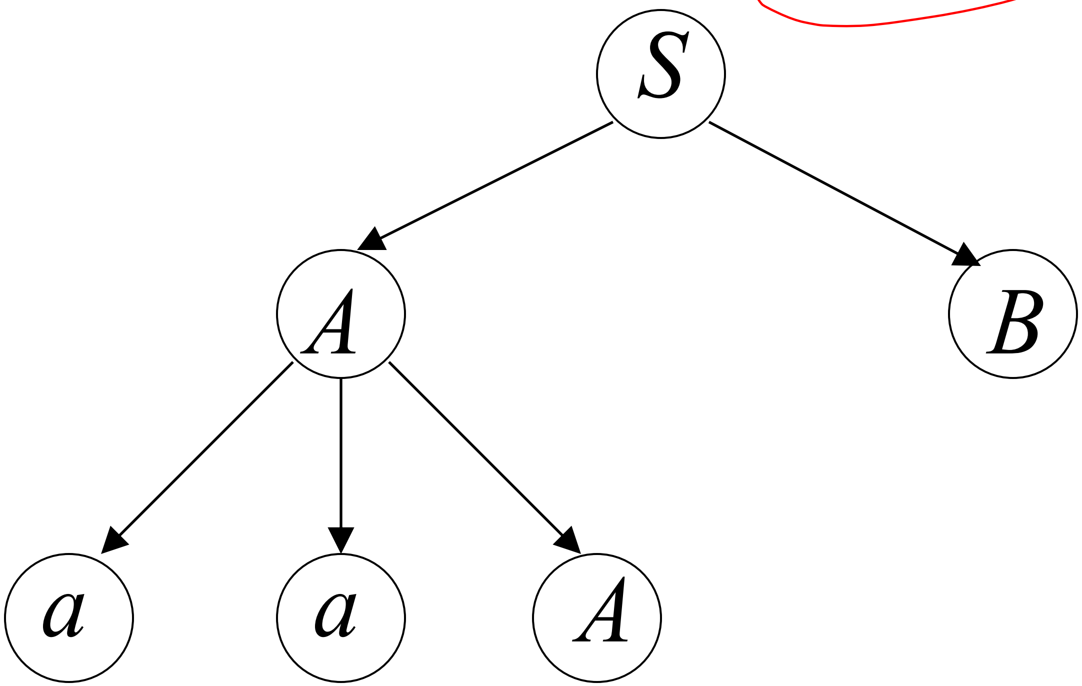
$$S \Rightarrow AB$$



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

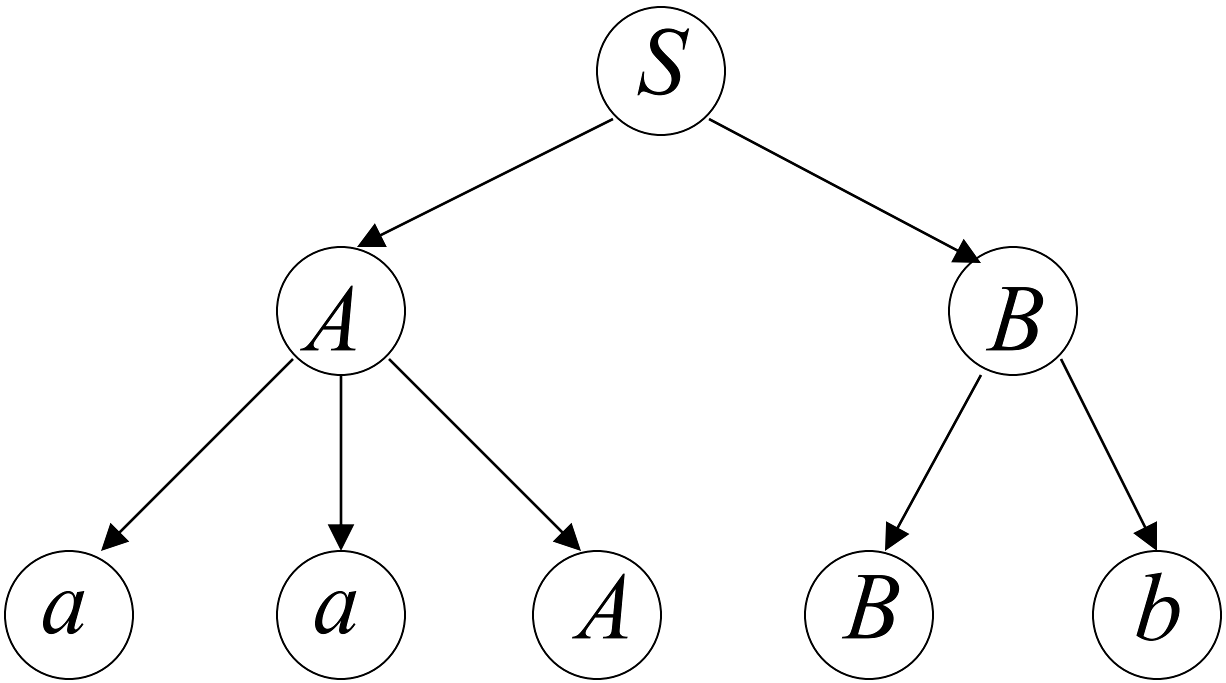
$$S \Rightarrow AB \Rightarrow aaAB$$

sential form → มี var, string ปน



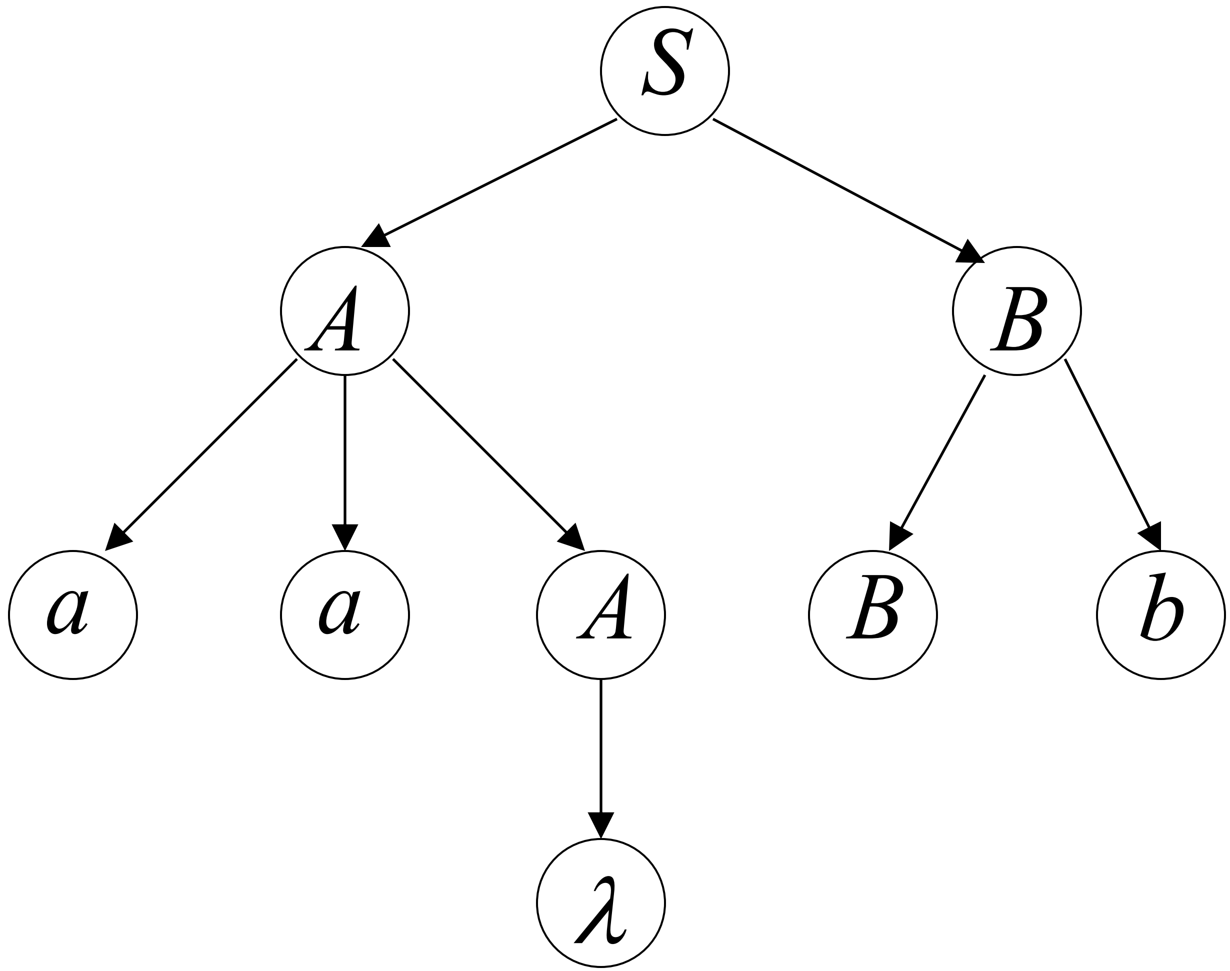
$$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$$

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



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

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

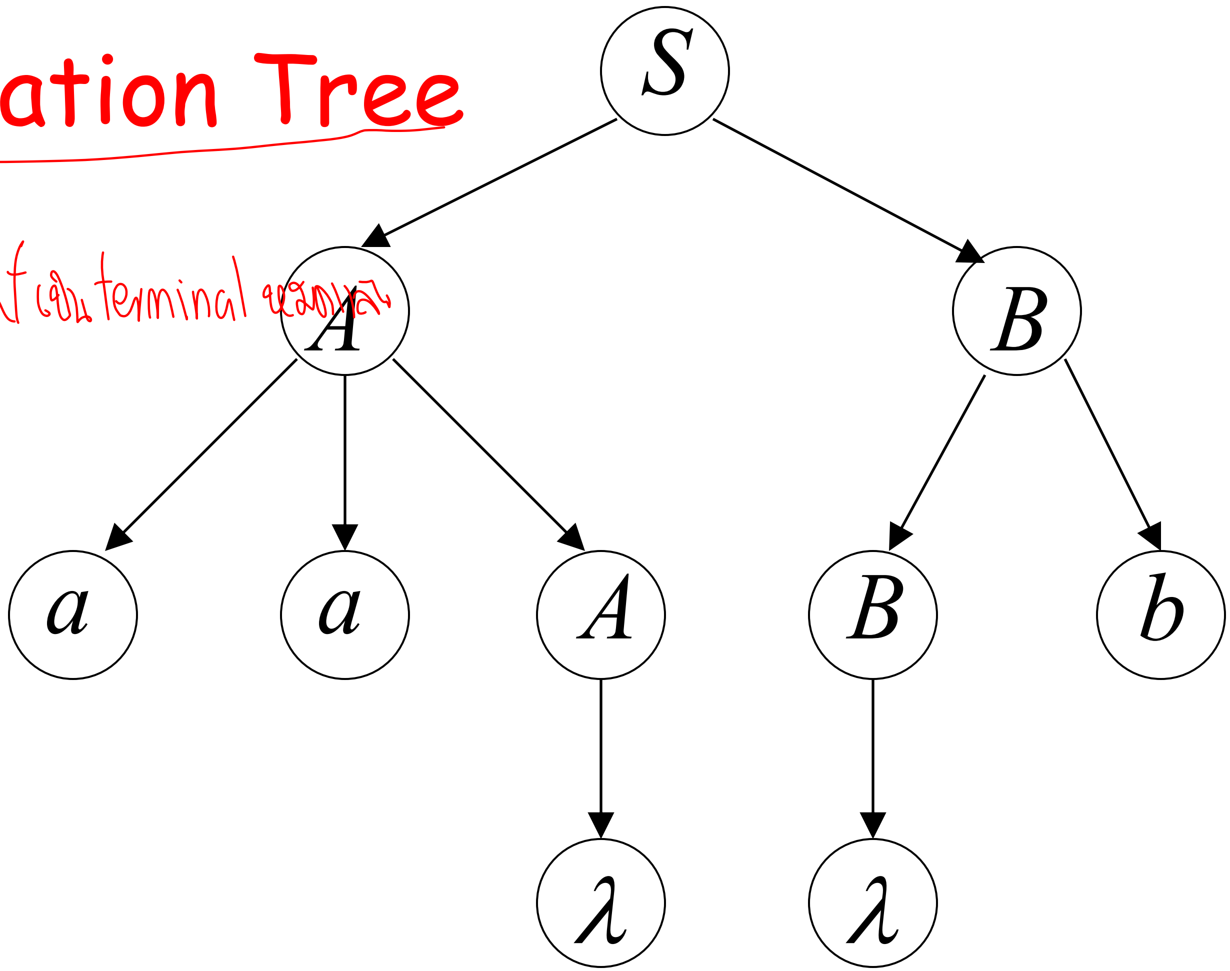


$$S \rightarrow AB \qquad A \rightarrow aaA \mid \lambda \qquad B \rightarrow Bb \mid \lambda$$

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

Derivation Tree

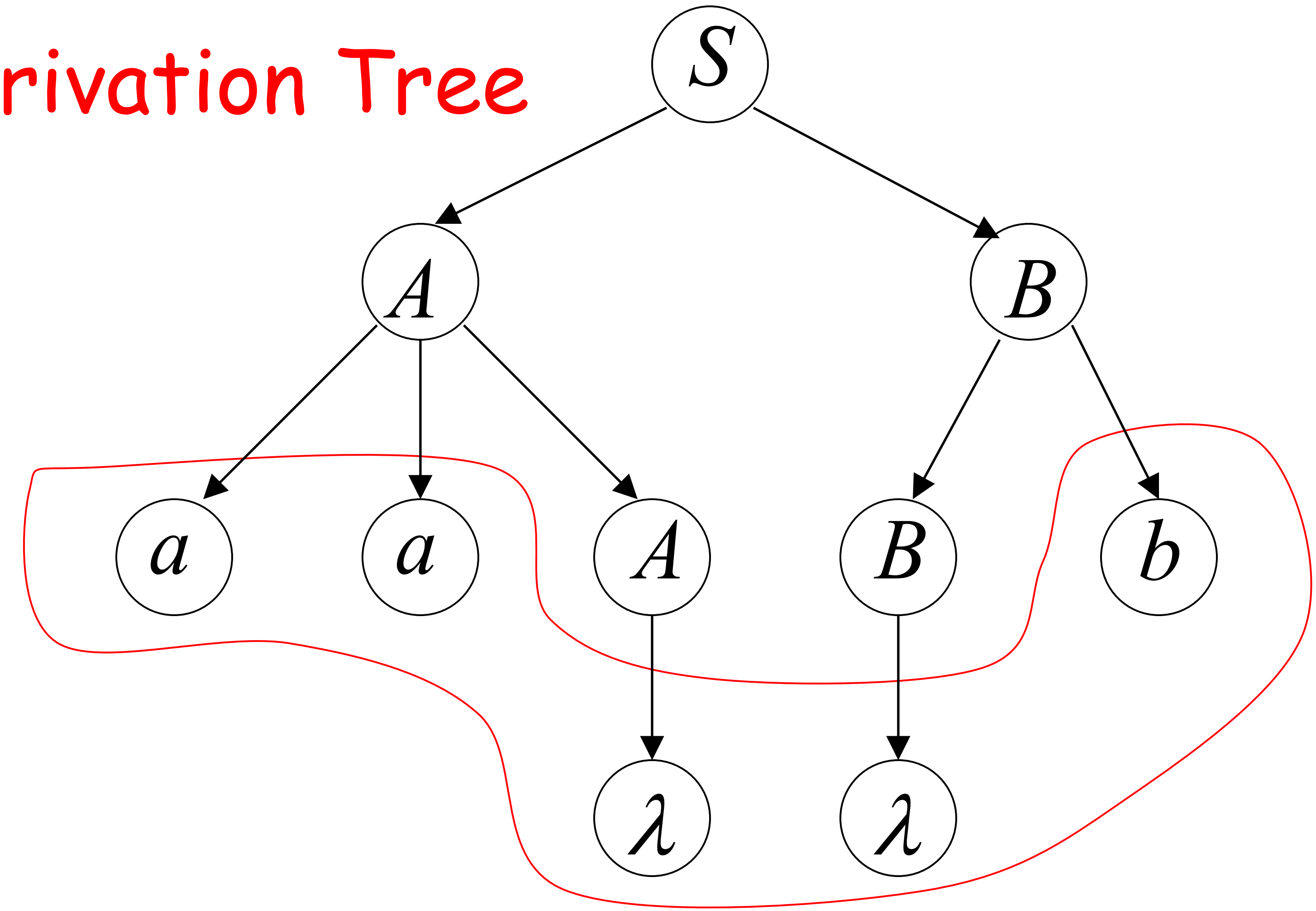
ที่ leaf เป็น terminal หมดแล้ว



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

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

Derivation Tree



yield

$aa\lambda\lambda b$

$= aab$

Partial Derivation Trees

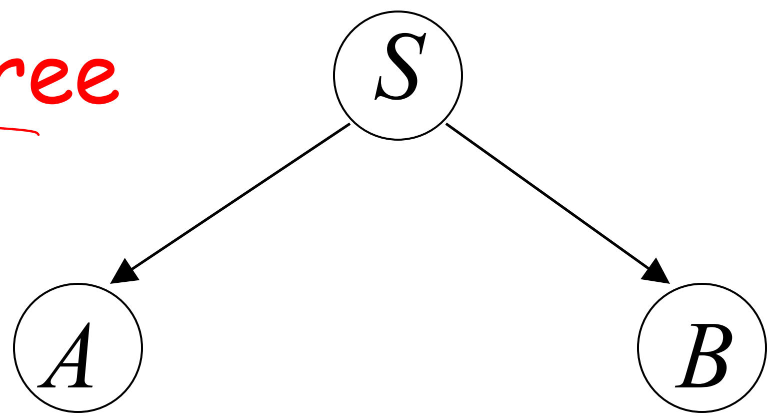
$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$

$S \Rightarrow AB$

Partial derivation tree

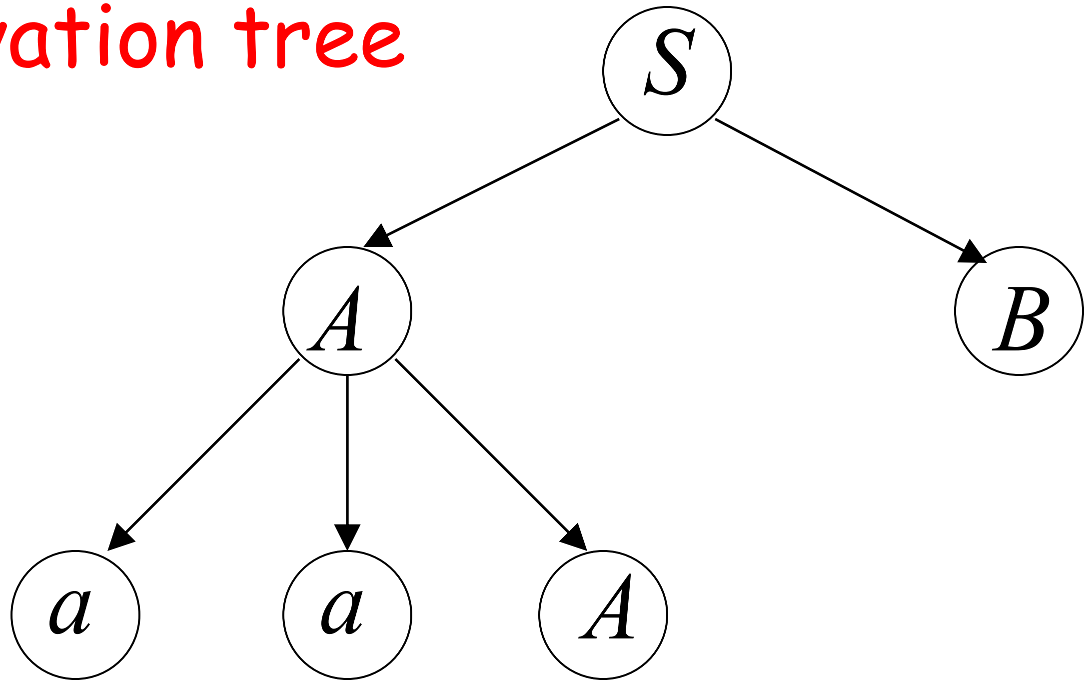
ยังทำไม่เสร็จ

มี varicble ที่ leaf อยู่



$$S \Rightarrow AB \Rightarrow aaAB$$

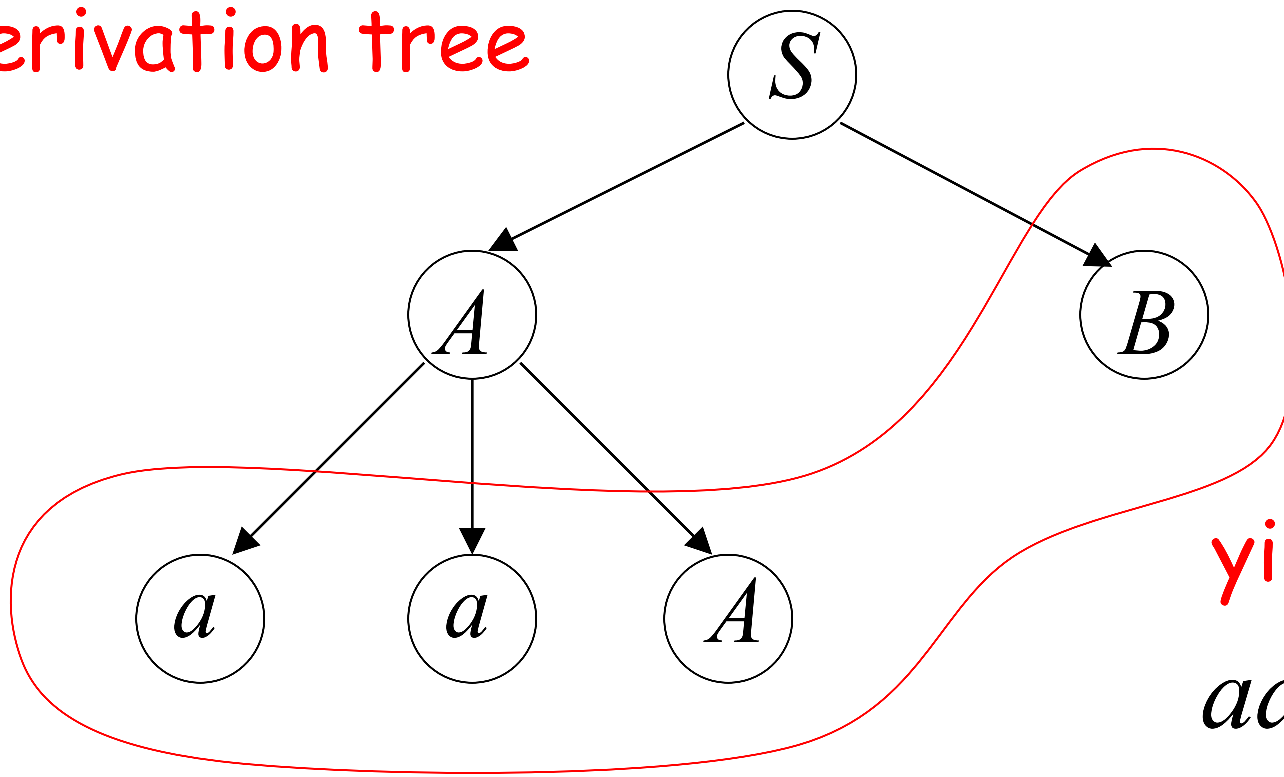
Partial derivation tree



$$S \Rightarrow AB \Rightarrow aaAB$$

sentential form

Partial derivation tree



yield

$aaAB$

Sometimes, derivation order doesn't matter

เราสามารถสร้าง string เดียวกันได้จากทั้ง 2 แบบ และสามารถแทนด้วย derivation tree ได้

Leftmost:

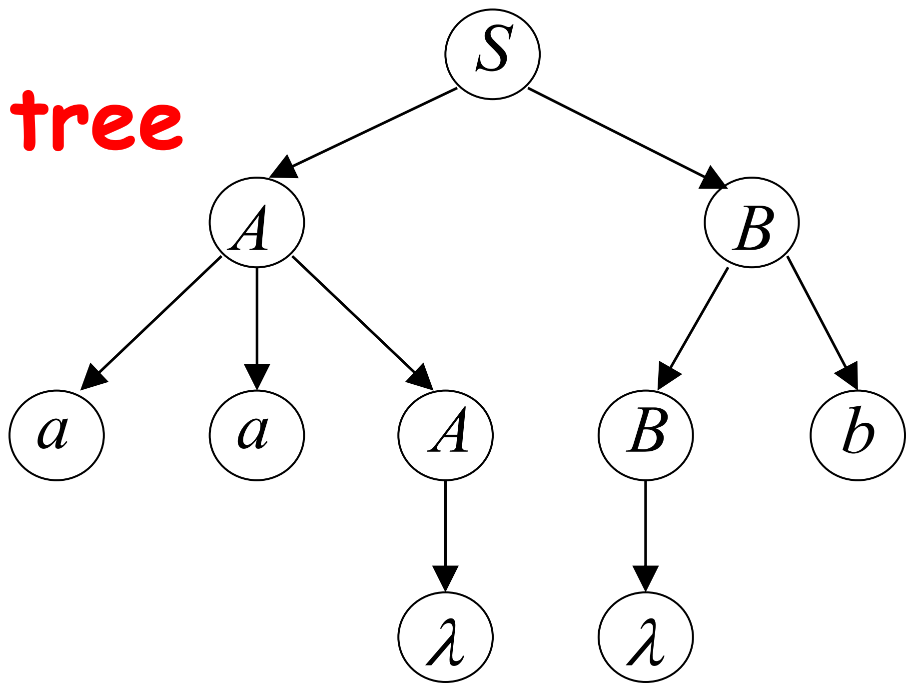
$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aaBb \Rightarrow aab$$

อยากเขียนแบบไหนก็เรื่องของมึง

Rightmost:

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

Same derivation tree



ambigous กำกวม

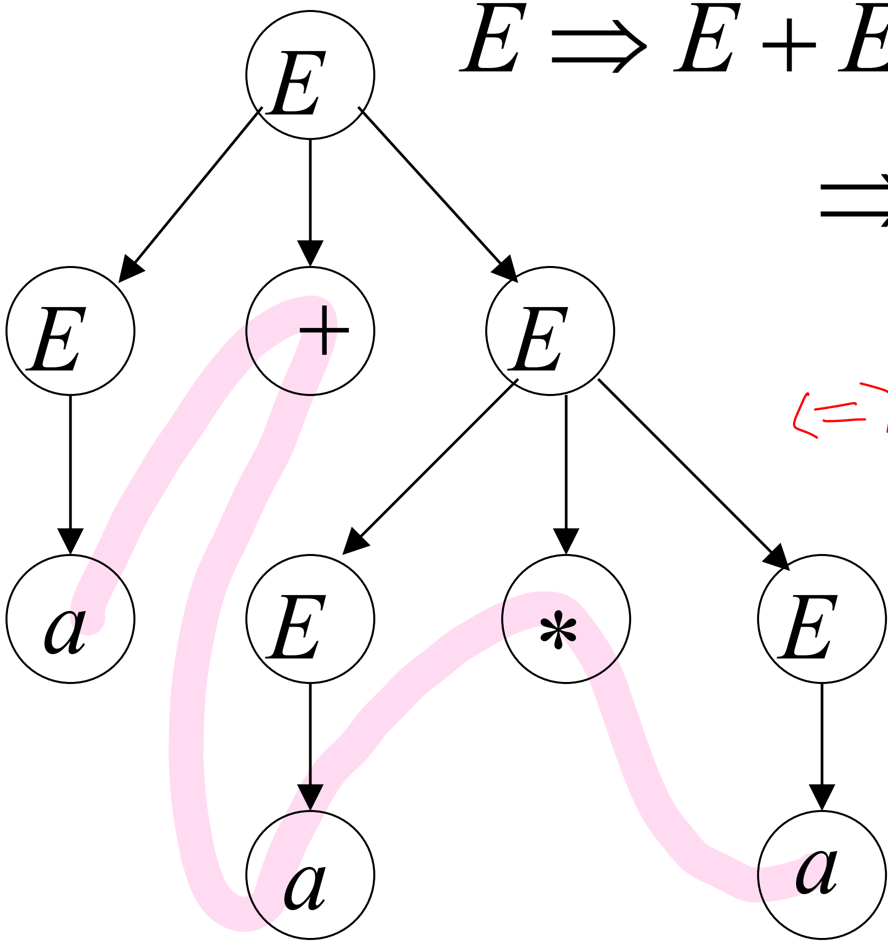
Ambiguity

2 + นัย → กำกวม

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

ใช้ครบหมด 4 production rule

$$a + a * a$$



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

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

$\Leftarrow$ **leftmost derivation**

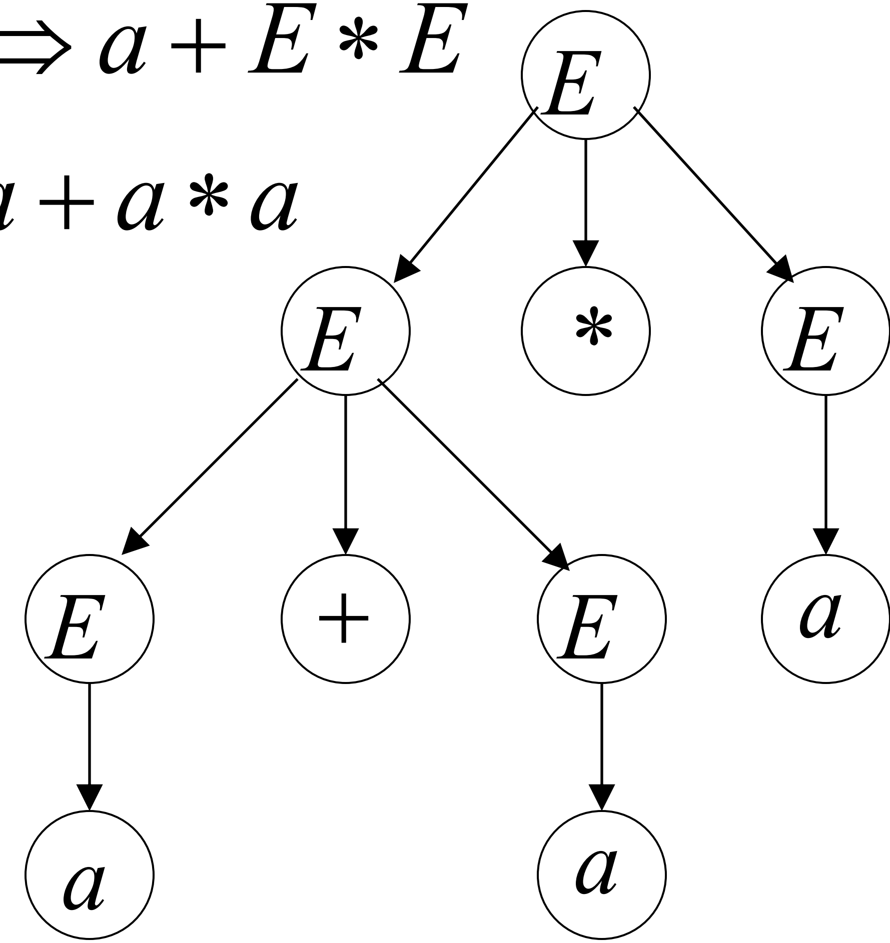
$$E \to E + E \quad | \; E * E \; | \; (E) \; | \; a$$

$$a + a * a$$

derive * ออกมาก่อน

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

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



leftmost derivation

ได้ tree ที่ต่างกัน แต่ string เดียวกัน

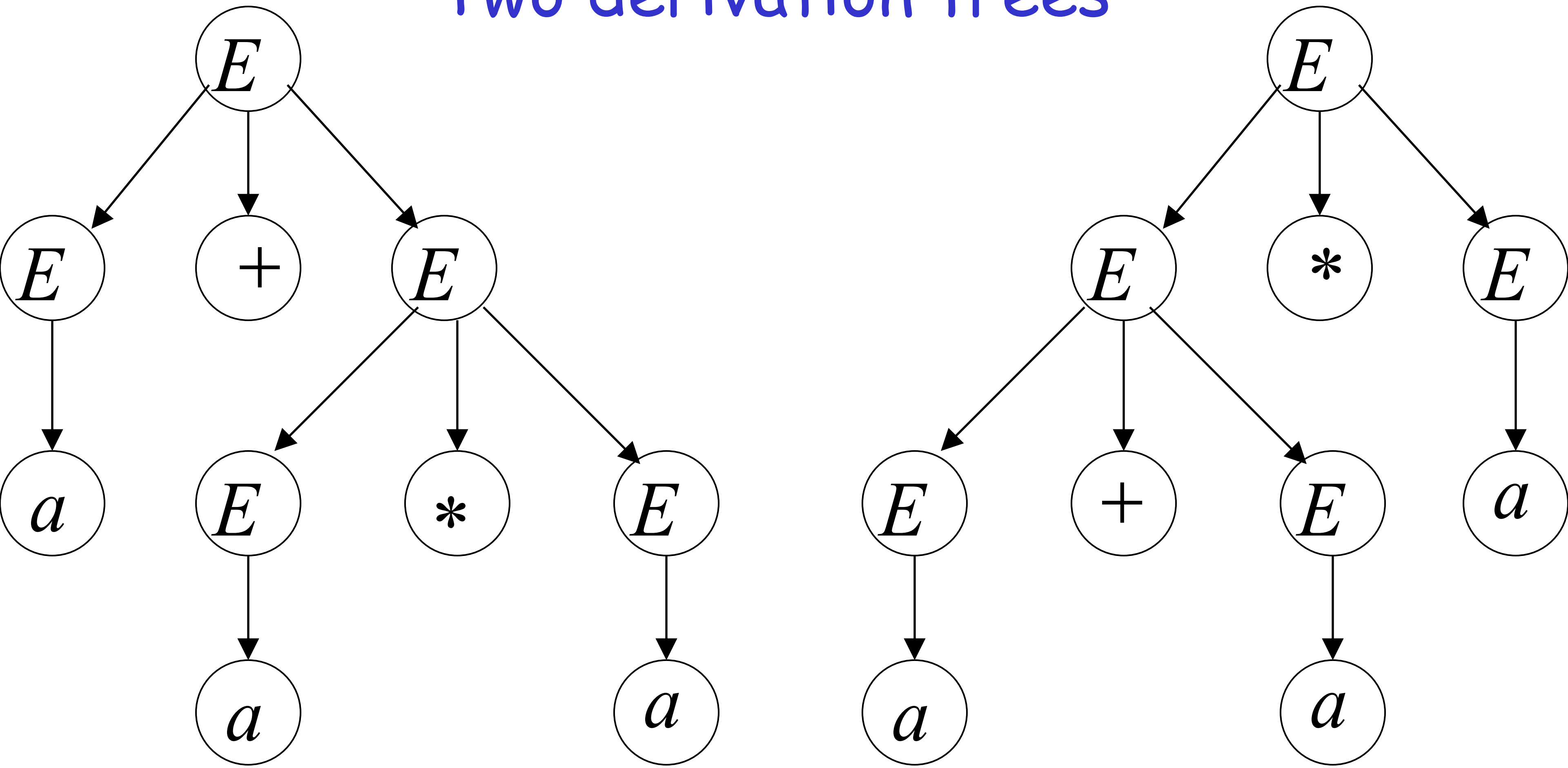
↓

แสดงว่า grammar นี้กำกวม

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

$$a + a * a$$

Two derivation trees



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

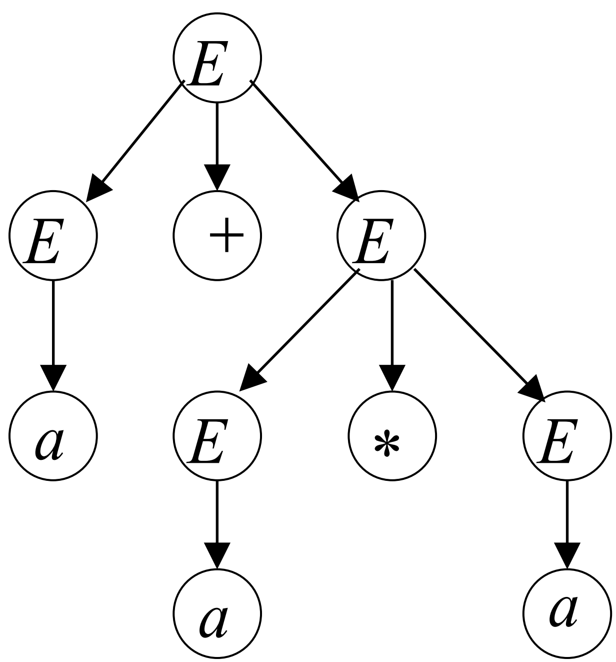
is **ambiguous:**

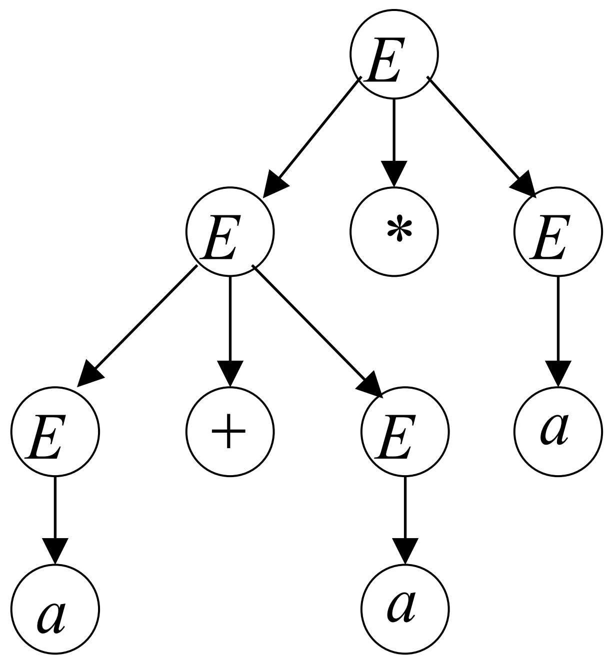
ถ้าจะพิสูจน์ว่า grammar นี้กำกวม → กำหนด string ขึ้นมา 1 อัน

แล้วหา derivation tree ที่ต่างกันมา ≥ 2 ต้น

✓ กำกวม

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





The grammar $E \to E + E \;|\; E * E \;|\; (E) \;|\; a$

is **ambiguous:**

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

ถ้าเขียนแผน derivation ได้

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

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

Definition:

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

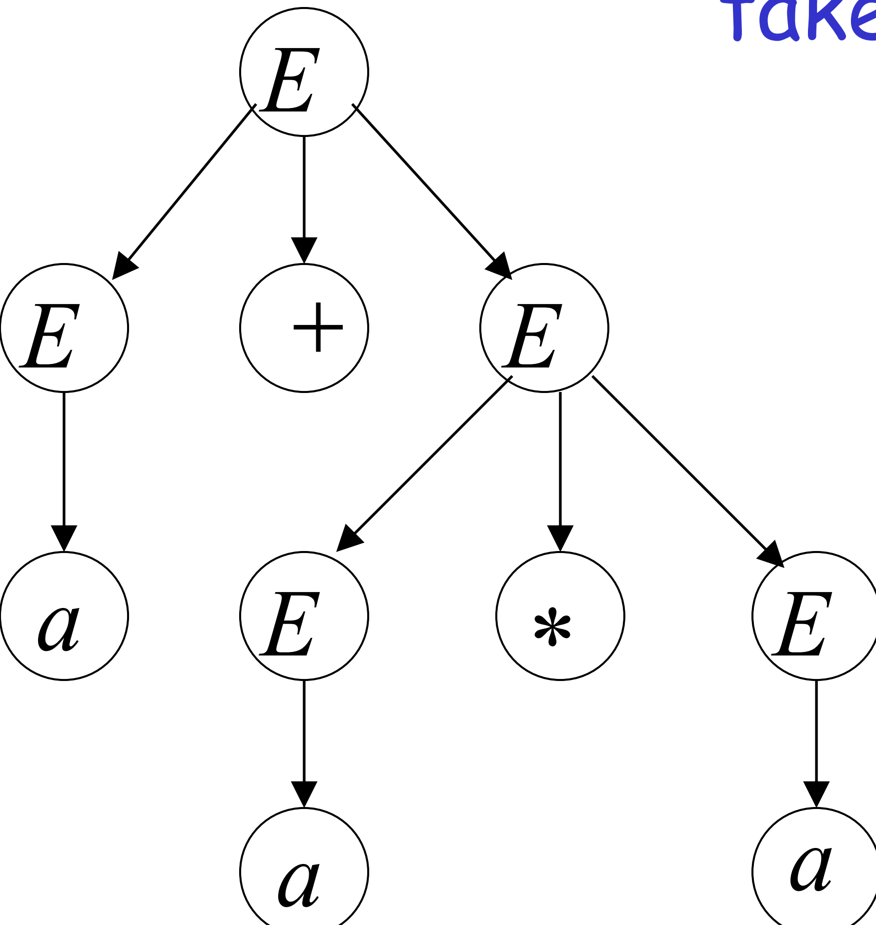
(or rightmost)

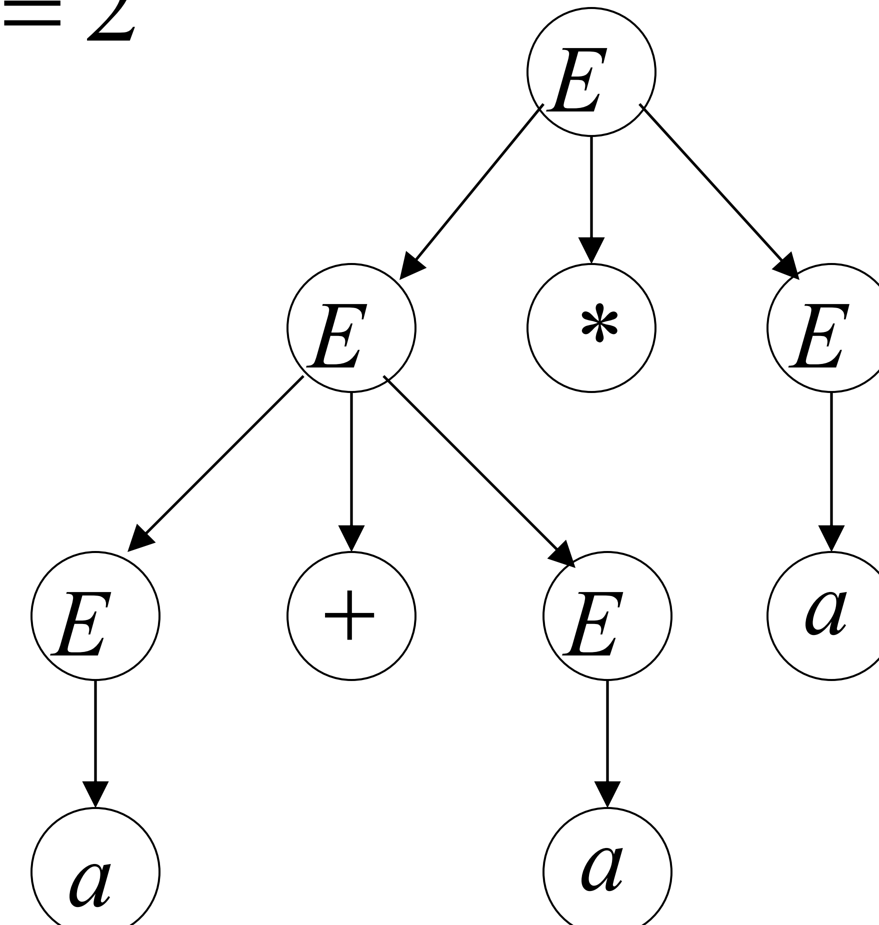
Why do we care about ambiguity?

ถ้ามันกำกวมแล้วทำไม

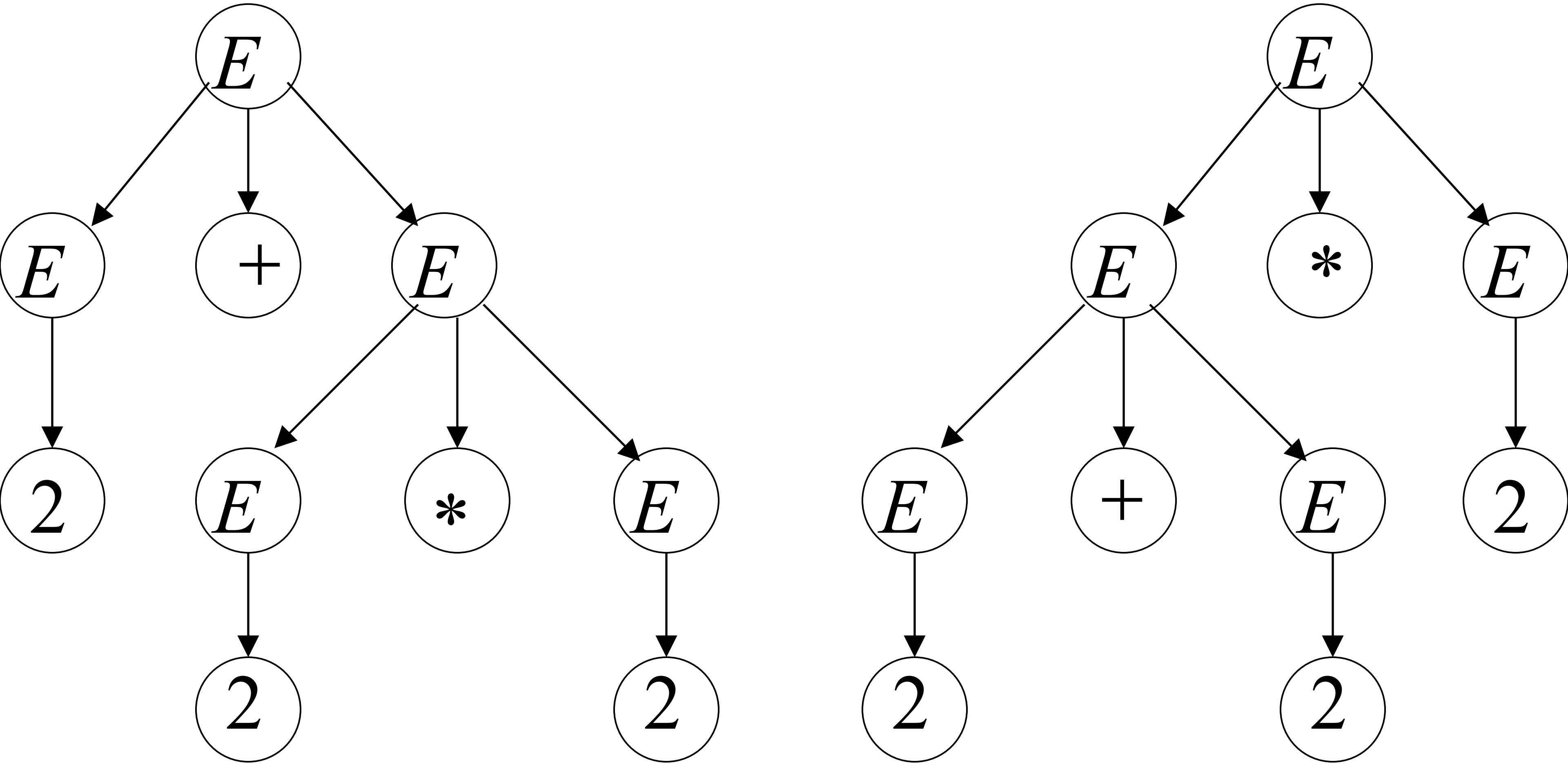
$$a + a * a$$

take $a = 2$

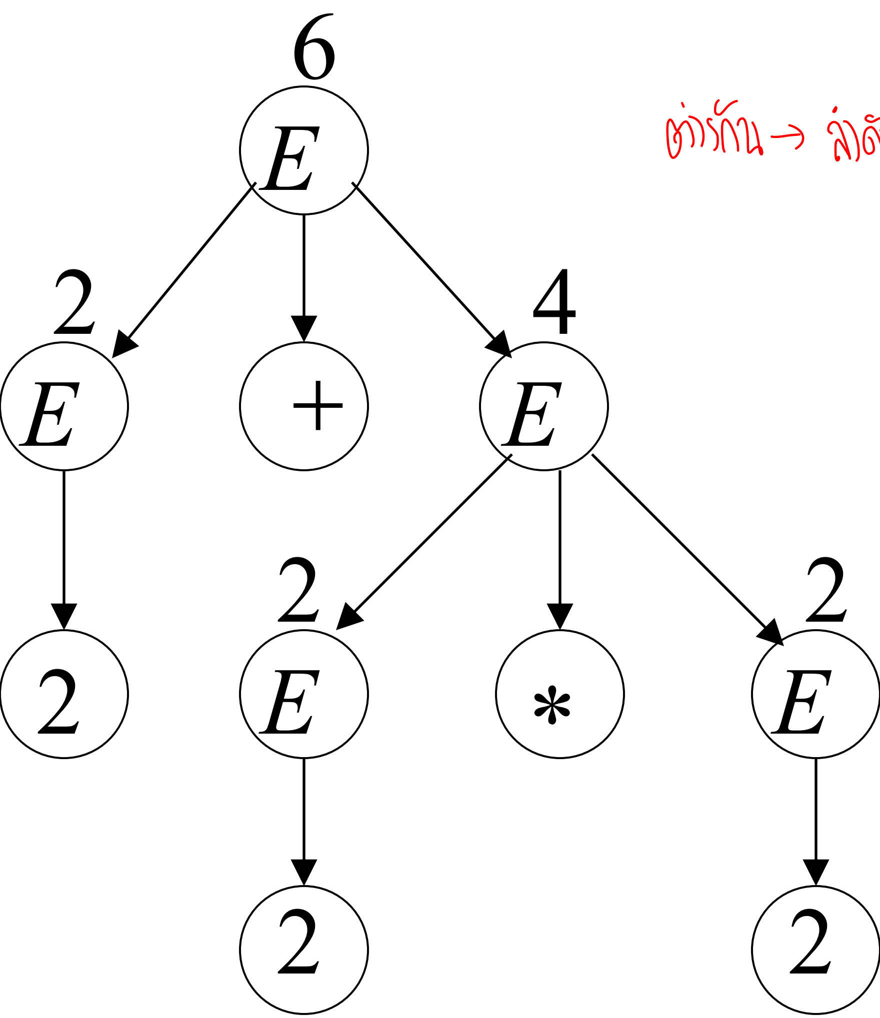




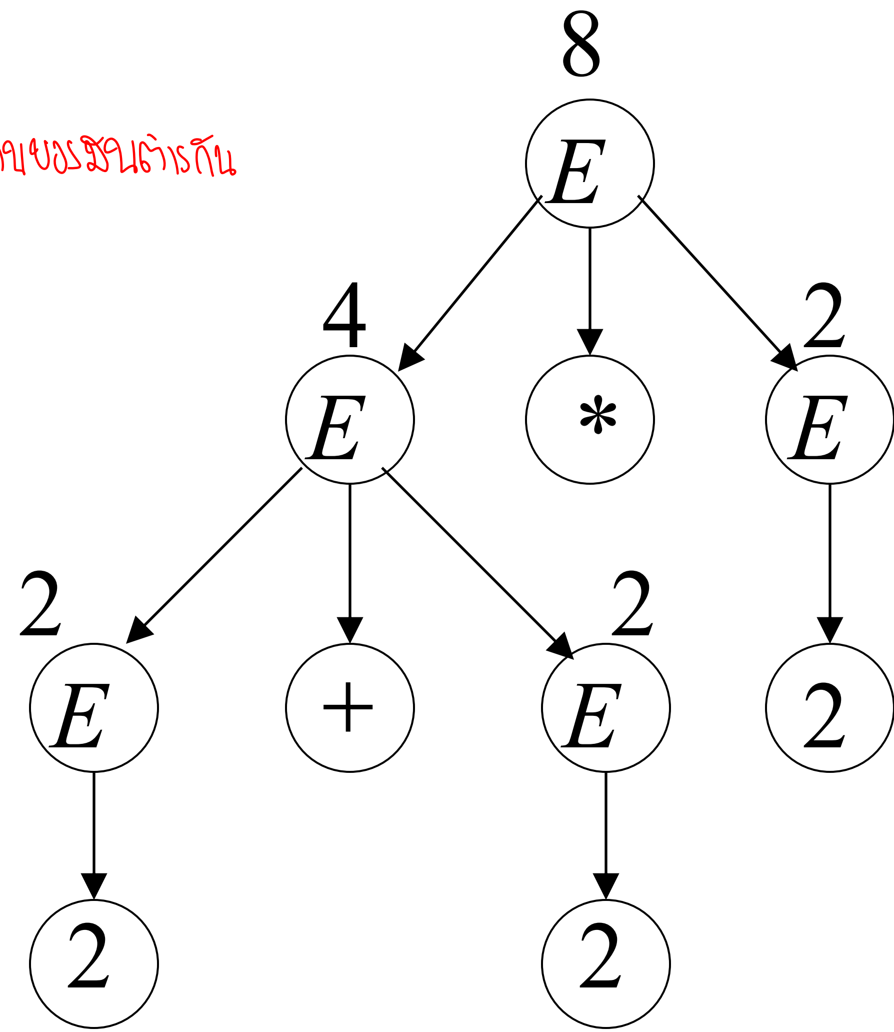
$$2 + 2 * 2$$



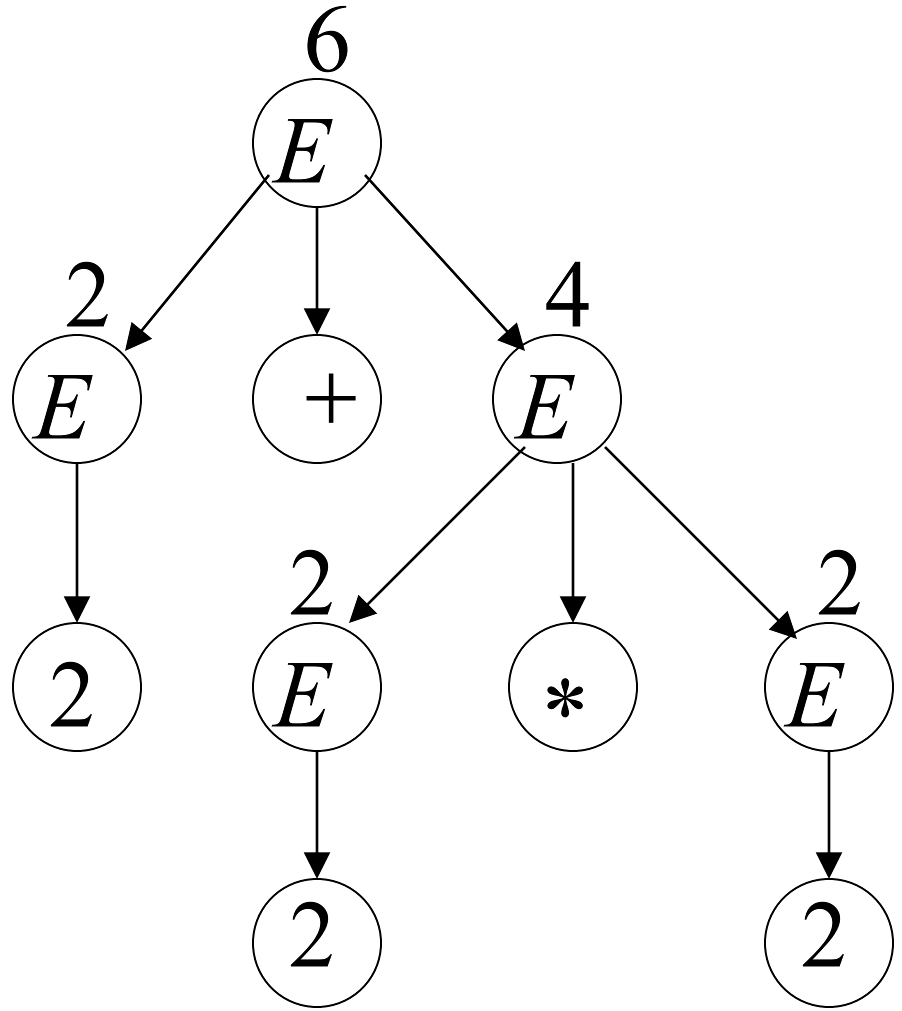
$$2 + 2 * 2 = 6 \qquad\qquad 2 + 2 * 2 = 8$$



ต่างกัน → ลำดับการทำงานของมันต่างกัน



Correct result: $2+2*2=6$



- Ambiguity is **bad** for programming languages

- We want to remove ambiguity → ต้องกำจัดทิ้ง

We fix the ambiguous grammar:

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

แกรมม่าที่มีทั้ง left , right recursion มีโอกาสเป็น ambiguous เยอะมาก

แปลงให้เป็นรูปแบบไม่กำกวม

New non-ambiguous grammar:

$$E \to E + T$$

$$E \to T$$

$$T \to T * F$$

$$T \to F$$

$$F \to (E)$$

$$F \to a$$

จารย์ไม่สอนวิธีแปลง

ambiguous → non-ambiguous

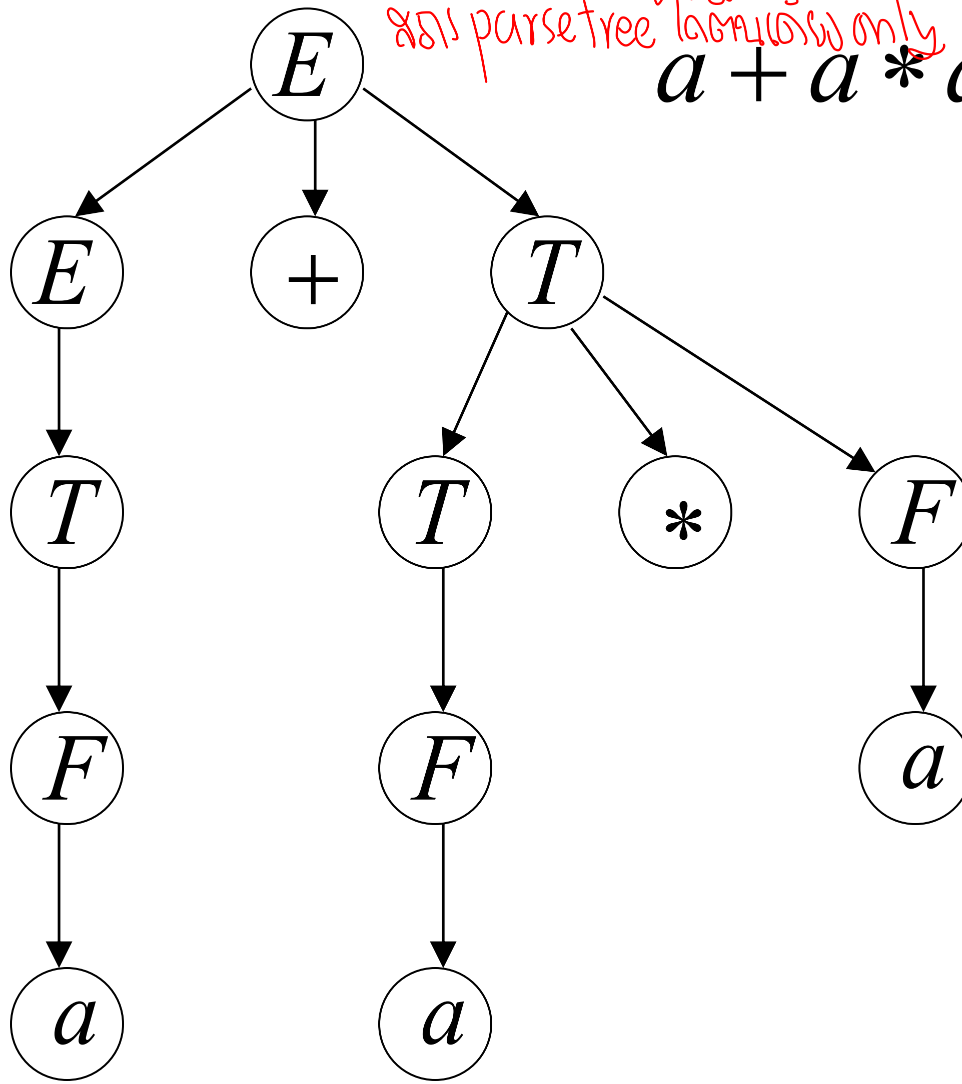
จารย์แค่ทำให้ดู

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

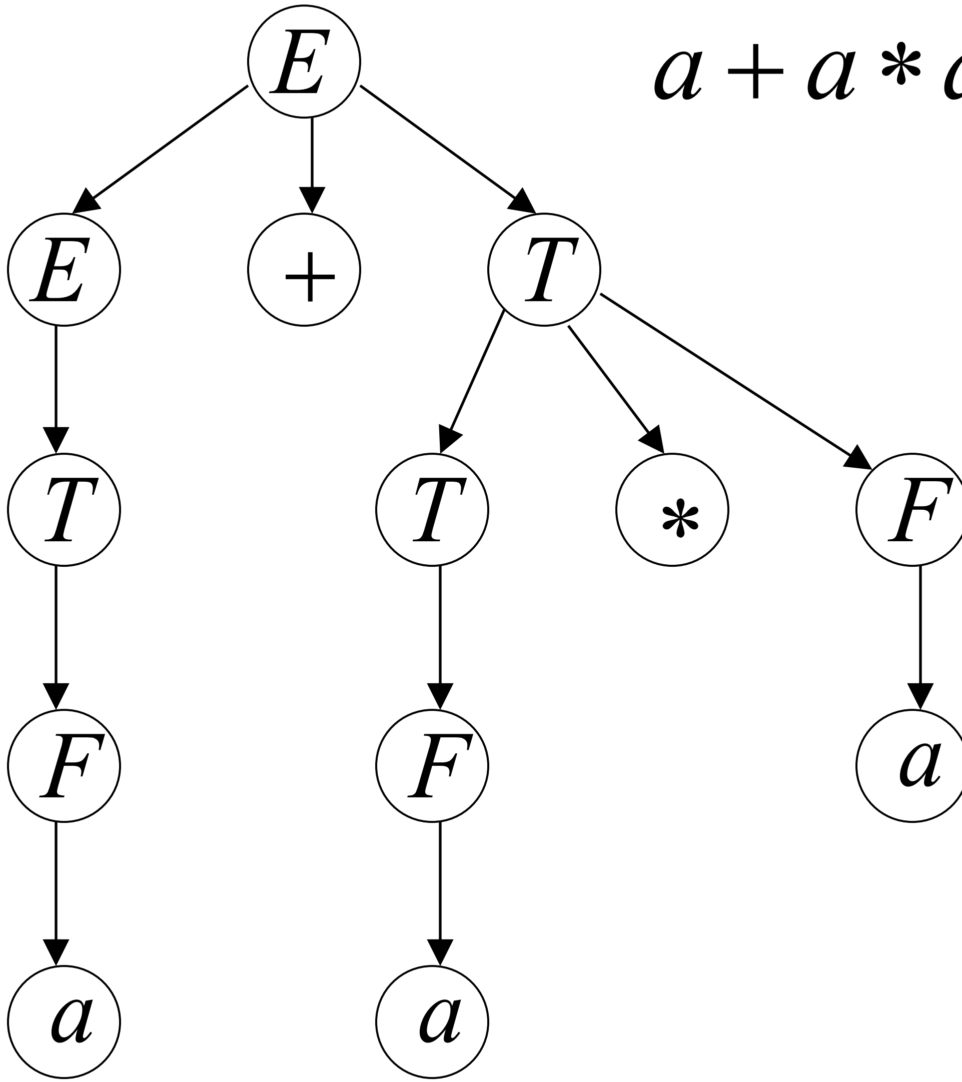
สร้าง parsetree ได้ต้นเดียว only

$$a+a*a$$

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



Unique derivation tree



$$a + a * a$$

The grammar G:

$$E \to E + T$$

$$E \to T$$

$$T \to T * F$$

$$T \to F$$

$$F \to (E)$$

$$F \to a$$

is non-ambiguous:

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

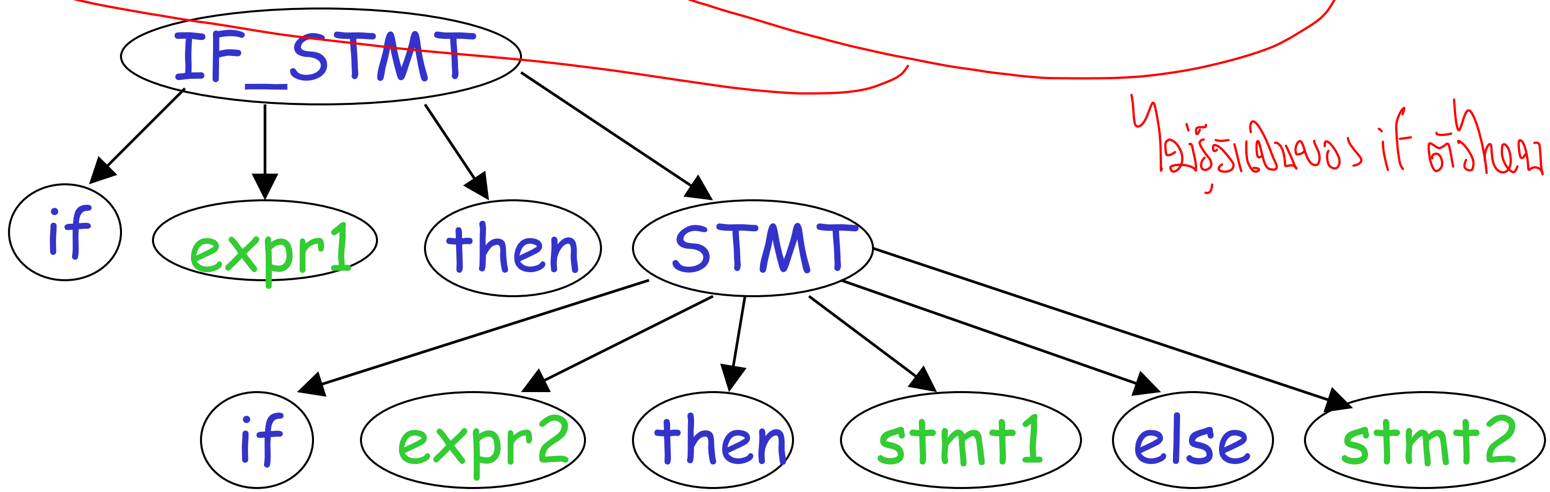
Another Ambiguous Grammar

Grammar ที่เป็น ambiguous

IF_STMT → if EXPR then STMT

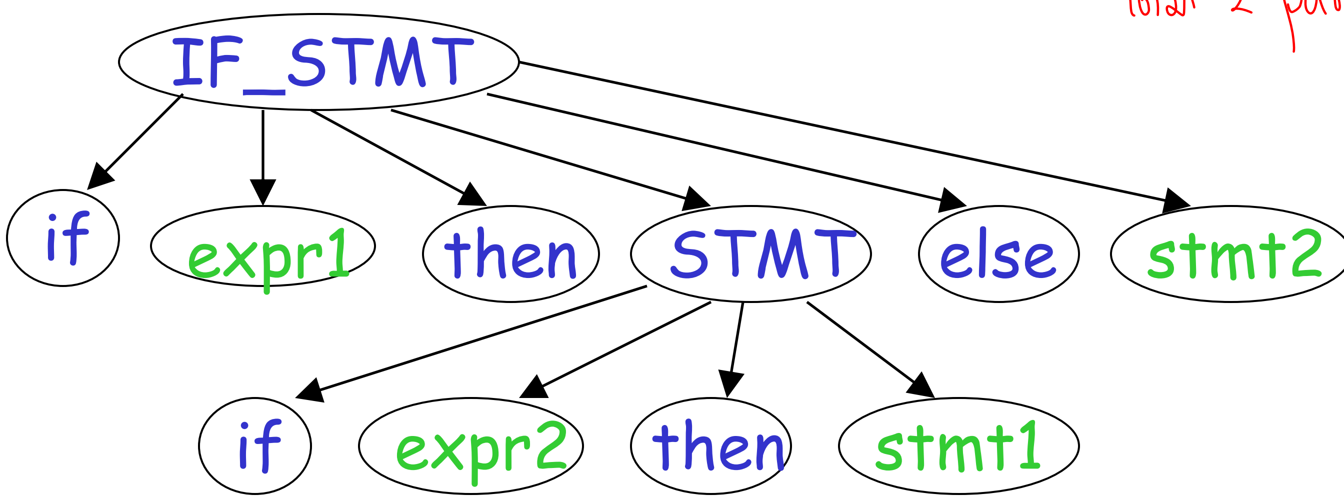
| if EXPR then STMT else STMT

If expr1 then if expr2 then stmt1 else stmt2



ไม่รู้ว่าเป็นของ if ตัวไหน

ได้มา 2 parse tree



Inherent Ambiguity

ได้ความกำกวมตั้งแต่กำเนิด

Some context free languages have only ambiguous grammars

ambiguous

$a^n b^n c^n$ อยู่ในทั้ง 2 → ambiguous

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

จะมี tree ของแกรมมาร์ซ้อน ก็ได้

ของแกรมมาก็ได้

$$S \rightarrow S_1 \mid S_2$$

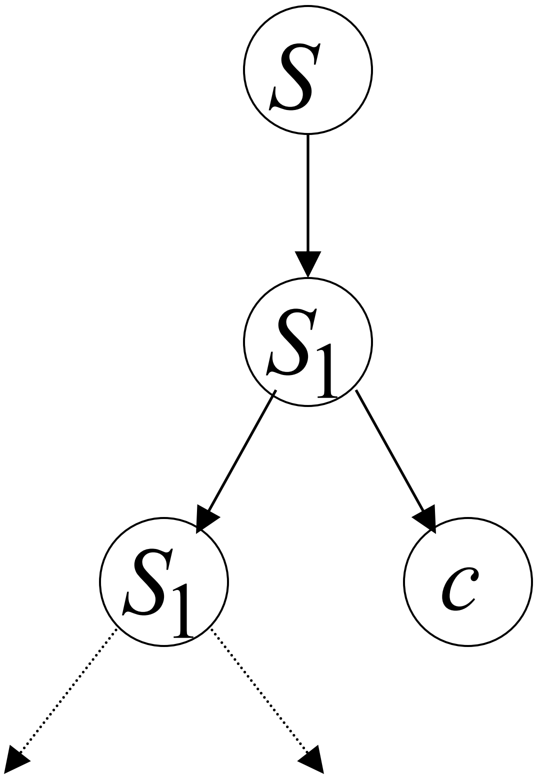
$$S_1 \rightarrow S_1 c \mid A$$
$$A \rightarrow aAb \mid \lambda$$

$$S_2 \rightarrow aS_2 \mid B$$
$$B \rightarrow bBc \mid \lambda$$

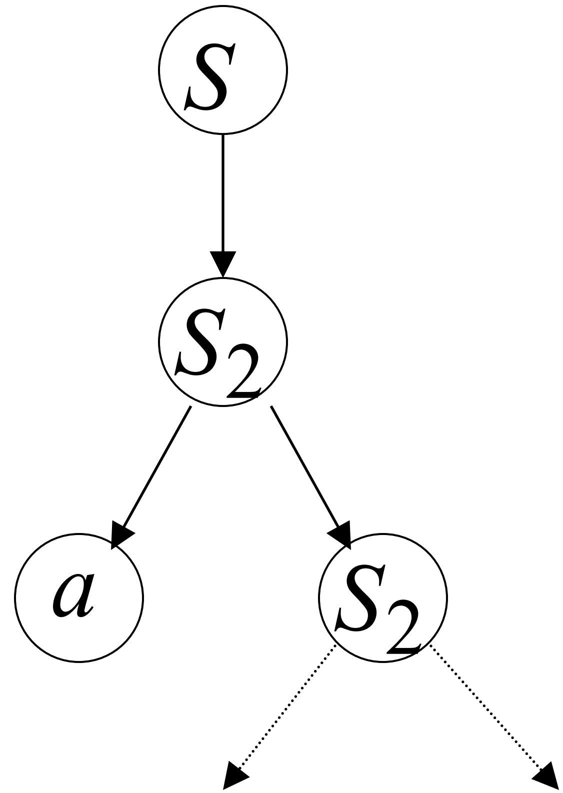
The string $a^n b^n c^n$

has two derivation trees

สามารถทำได้ ๒ วิธี



ลำดับการเติมสตริงต่างกัน



Compilers

เป็นของ ขั้นสูง

Program

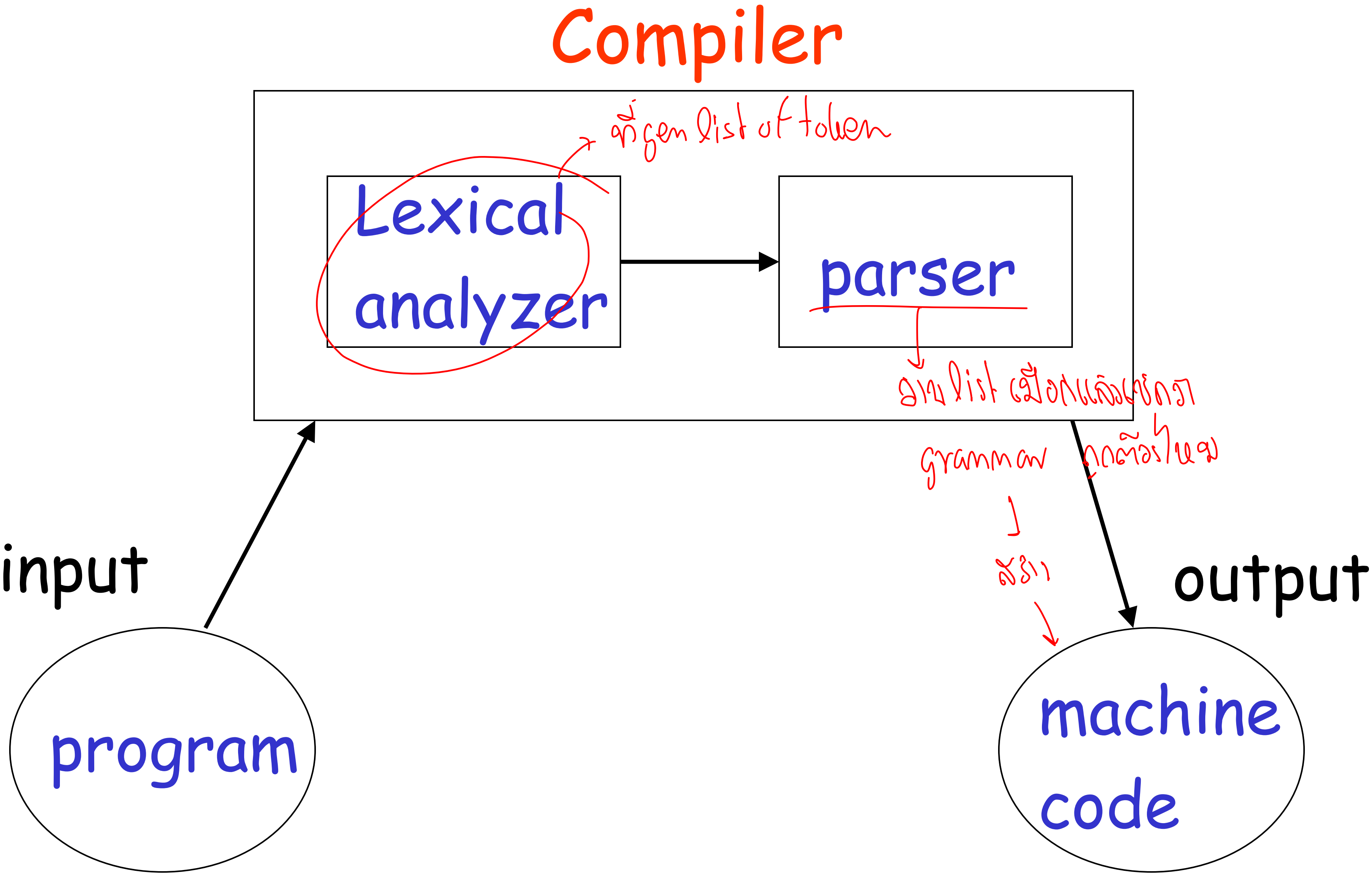
```
v = 5;
if (v>5)
   x = 12 + v;
while (x !=3) {
  x = x - 3;
  v = 10;
}
......
```

→ **Compiler** →

แปลงโปรแกรมต้นฉบับ → machine code

Machine Code

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


Compiler
ที่gen list of token
Lexical analyzer
parser
อ่าน list เพื่อแล้วเช็คว่า grammar ถูกต้องไหม
สร้าง
input
output
program
machine code

A parser knows the grammar of the programming language

Parser ต้องอยู่ใน grammar ใน 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

input

10 + 2 * 5

Parser

E -> E + E
 | E * E
 | INT

ดู grammar ลง
เพื่อหาทางเดินไป

derivation

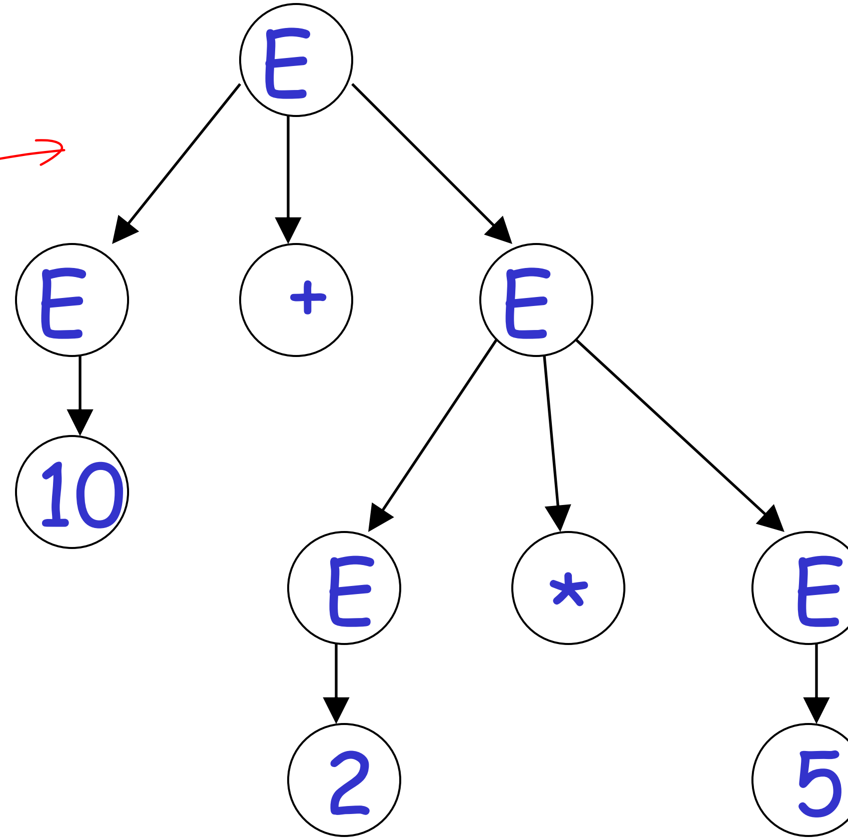
E => E + E
=> E + E * E
=> 10 + E*E
=> 10 + 2 * E
=> 10 + 2 * 5

derivation

$$
\begin{aligned}
E &\Rightarrow E + E \\
&\Rightarrow E + E * E \\
&\Rightarrow 10 + E * E \\
&\Rightarrow 10 + 2 * E \\
&\Rightarrow 10 + 2 * 5
\end{aligned}
$$

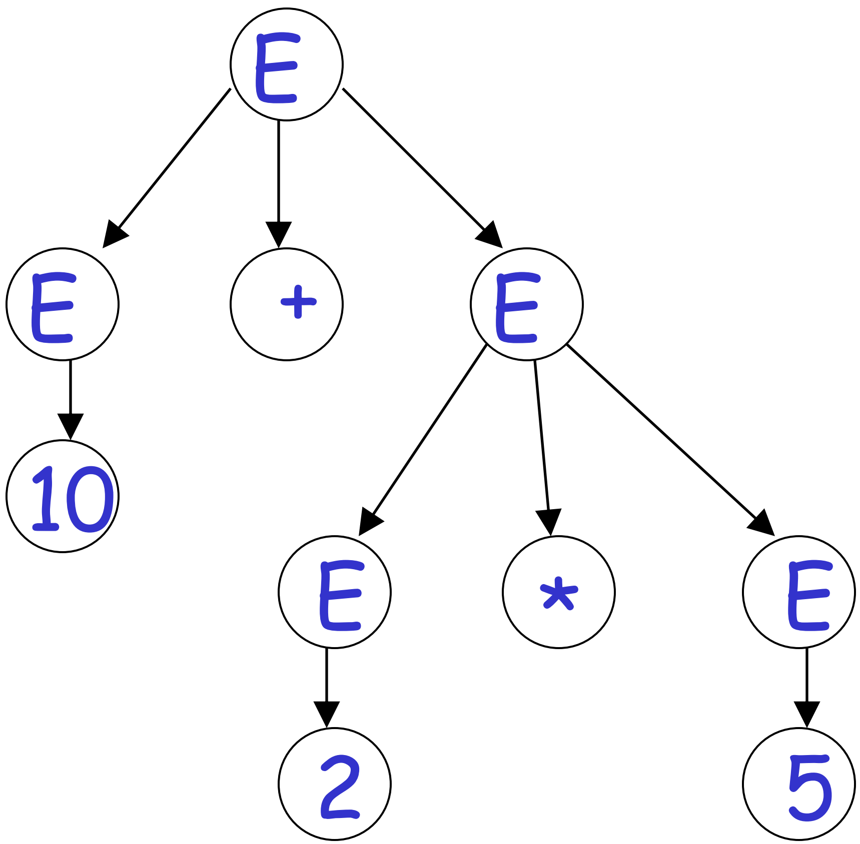
สร้าง

derivation tree



derivation tree

ซึ่งจะเดินตามเส้นของ tree เพื่อสร้าง machine code



machine code

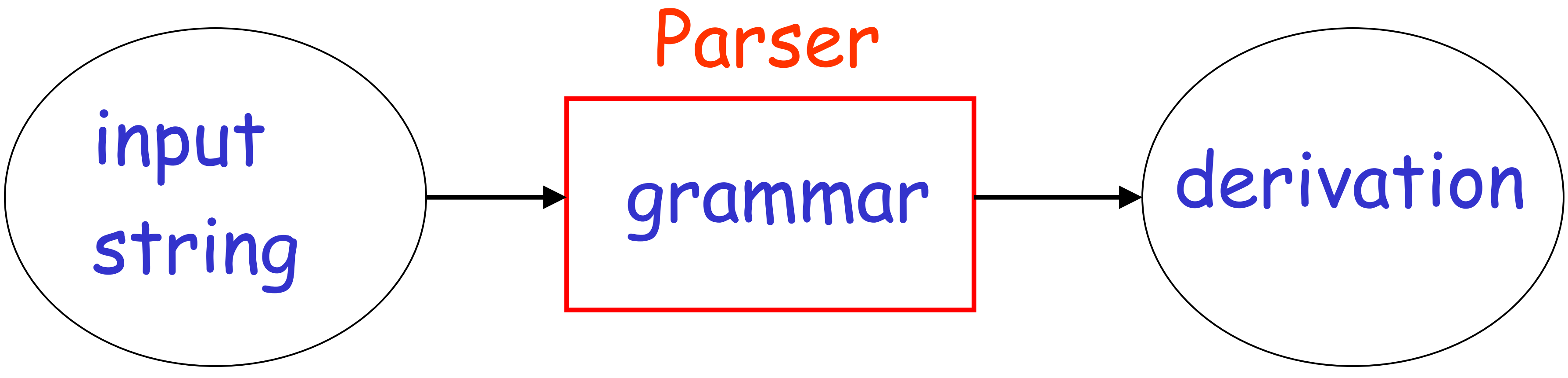
mult a, 2, 5
add b, 10, a

สิ่งที่เราไม่สนใจคือ จะเปลี่ยนไปเป็น machine code อย่างไร

แต่สนใจคือ จะรู้ได้อย่างไรว่าต้อง derivation แบบไหน

กระบวนการ

Parsing


Parser
input string
grammar
derivation

Example:

Parser

$$S \to SS$$

$$S \to aSb$$

$$S \to bSa$$

$$S \to \lambda$$

input

$aabb$

derivation

?

Exhaustive Search

ฮินรู้ได้ไรว่าแน่ไหม

brute force

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

Phase 1:

$$S \Rightarrow SS$$

$$S \Rightarrow aSb$$

$$S \Rightarrow bSa$$

$$S \Rightarrow \lambda$$

Find derivation of

$$aabb$$

All possible derivations of length 1

$$\boxed{\begin{aligned} S &\Rightarrow SS \\ S &\Rightarrow aSb \end{aligned}}$$

ได้ 2 กรณี

$$aabb$$

$$\cancel{S \Rightarrow bSa}$$

$$\cancel{S \Rightarrow \lambda}$$

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

$$S \Rightarrow SS \Rightarrow SSS$$

$$S \Rightarrow SS \Rightarrow aSbS \qquad\qquad aabb$$

$$\cancel{S \Rightarrow SS \Rightarrow bSaS}$$

Phase 1

$$S \Rightarrow SS \qquad\qquad S \Rightarrow SS \Rightarrow S$$

แตกเพิ่มไปเรื่อยๆ

$$S \Rightarrow aSb$$

$$S \Rightarrow aSb \Rightarrow aSSb$$

$$S \Rightarrow aSb \Rightarrow aaSbb$$

$$\cancel{S \Rightarrow aSb \Rightarrow abSab}$$

$$\cancel{S \Rightarrow aSb \Rightarrow ab}$$

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

Phase 2

$$S \Rightarrow SS \Rightarrow SSS$$

$$S \Rightarrow SS \Rightarrow aSbS$$

$$aabb$$

$$S \Rightarrow SS \Rightarrow S$$

$$S \Rightarrow aSb \Rightarrow aSSb$$

$$S \Rightarrow aSb \Rightarrow aaSbb$$

Phase 3

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

Final result of exhaustive search (top-down parsing)

Parser

แตกไปเรื่อยๆ

$$S \to SS$$

อย่าหาค่า

$$S \to aSb$$

$$S \to bSa$$

$$S \to \lambda$$

input

$aabb$

derivation

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

Time complexity of exhaustive search

Suppose there are no productions of the form

ไม่มีพวกแตกแล้วได้ λ หรือ 1 terminal

$$A \rightarrow \lambda$$

$$A \rightarrow B$$

Number of phases for string w : $2|w|$

จำนวน phase = 2 เท่าของความยาว string

For grammar with k rules

Time for phase 1: k

k possible derivations

Time for phase 2: k^2

b^n

k^2 possible derivations

ได้ $d = 2|w|$

Time for phase $2|w|$: $k^{2|w|}$

$k^{2|w|}$ possible derivations

เวลาที่รันหมดทั้งเอาไว้งาน

Total time needed for string w:

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

phase 1

phase 2

phase 2|w|

Extremely bad!!!

There exist faster algorithms
for specialized grammars

ไม่ใช่ทุกภาษาเขียนแบบนี้ได้

ไม่จริง แต่เร็วกว่าแบบ exhaustiv search

S-grammar:

ถ้าไม่ตรงไม่ไป

$$A \to ax$$

symbol

string of variables

Pair (A, a) appears once

S-grammar example:

$$S \to aS$$

$$S \to bSS$$

$$S \to c$$

Each string has a unique derivation

ไม่ได้แตกออกมาทางเลือก

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

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: $|w|$

For general context-free grammars:

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

(we will show it in the next class)