Regular Grammars

Formal Languages and Abstract Machines

Week 05

Baris E. Suzek, PhD

Outline

- Last week
- Grammars
- Regular Grammars
- Context-free Grammars
    - Definitions

NFA vs. DFA

- Transition functions range is Q vs. 2^Q (powersets of Q)
- λ can be an argument of transition function; transition without consuming a symbol
- $\delta(q_k, a)$ can be empty (not a total function)

δ	a	b
q_0	q_1	
q_1		q_2

Regular Languages

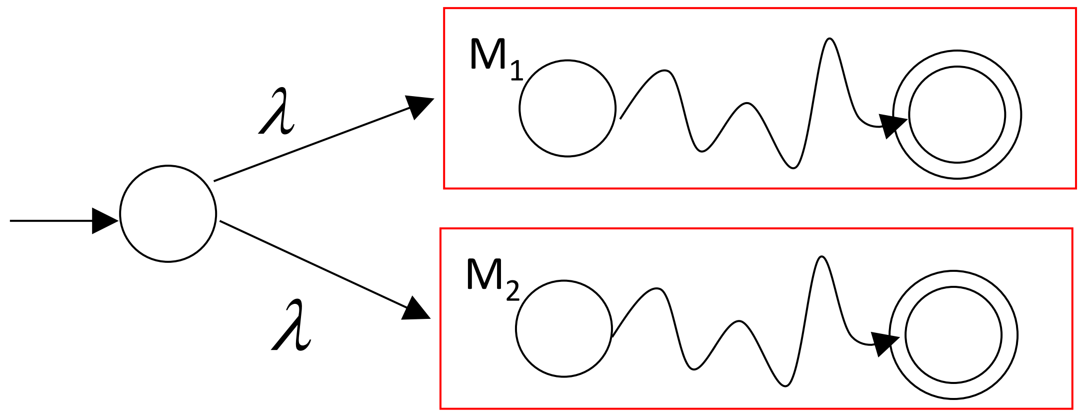
- A language L is regular if there is a DFA M such that $L = L(M)$

- All regular languages form a language family

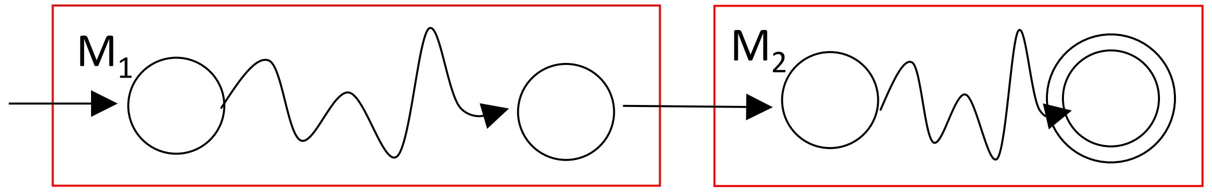
Regular Expressions and Automata

$$L(r_1) \quad L(r_2)$$

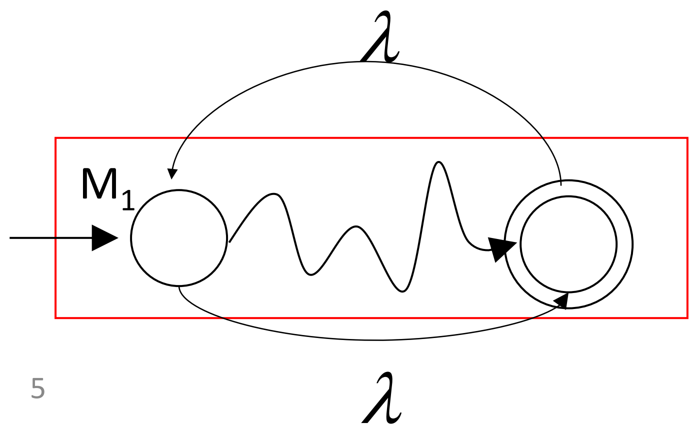
$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$



$$L(r_1 \cdot r_2) = L(r_1)\, L(r_2)$$



$$L(r_1 *) = (L(r_1))*$$



Describing Regular Languages

- DFA or NFA (covered)
- Regular expressions (covered)
- Regular grammars

Outline

- Last week
- Grammars
- Regular Grammars
- Context-free Grammars
  - Definitions

Grammars

- Grammars express languages

- Example: the English language

$$\langle sentence \rangle \rightarrow \langle noun\_phrase \rangle \ \langle verb \rangle$$

$$\langle noun\_phrase \rangle \rightarrow \langle article \rangle \ \langle noun \rangle$$

$$\langle article \rangle \rightarrow a$$
$$\langle article \rangle \rightarrow the$$

$$\langle noun \rangle \rightarrow cat$$
$$\langle noun \rangle \rightarrow dog$$

$$\langle verb \rangle \rightarrow runs$$
$$\langle verb \rangle \rightarrow walks$$

- A derivation of “the dog walks”:

$$\begin{aligned}
\langle sentence \rangle &\Rightarrow \langle noun\_phrase \rangle \ \langle verb \rangle \\
&\Rightarrow \langle article \rangle \ \langle noun \rangle \ \langle verb \rangle \\
&\Rightarrow the \ \ \langle noun \rangle \ \langle verb \rangle \\
&\Rightarrow the \ \ dog \ \ \langle verb \rangle \\
&\Rightarrow the \ \ dog \ \ walks
\end{aligned}$$

- A derivation of "a cat runs":

$$\begin{aligned}\langle sentence\rangle &\Rightarrow \langle noun\_phrase\rangle\ \langle verb\rangle\\ &\Rightarrow \langle article\rangle\ \langle noun\rangle\ \langle verb\rangle\\ &\Rightarrow a\ \ \langle noun\rangle\ \langle verb\rangle\\ &\Rightarrow a\ \ cat\ \ \langle verb\rangle\\ &\Rightarrow a\ \ cat\ \ runs\end{aligned}$$

Language of the Grammar

$\langle article \rangle \to a$

$\langle article \rangle \to the$

$\langle noun \rangle \to cat$ **+**

$\langle noun \rangle \to dog$

$\langle verb \rangle \to runs$

$\langle verb \rangle \to walks$

$\langle sentence \rangle \to \langle noun\_phrase \rangle \; \langle verb \rangle$

$\langle noun\_phrase \rangle \to \langle article \rangle \; \langle noun \rangle$

$\Rightarrow$

L = { "a cat runs",
"a cat walks",
"the cat runs",
"the cat walks",
"a dog runs",
"a dog walks",
"the dog runs",
"the dog walks" }

Notation

$$\overbrace{\begin{gathered}\langle noun \rangle \to cat \\ \langle noun \rangle \to dog\end{gathered}}^{\text{Production Rules}}$$

Variable

Terminal

Another Example

- Grammar:

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

- Derivation of sentence : ab

$$\begin{tikzcd}
S & \Rightarrow & aSb & \Rightarrow & ab \\
S \to aSb \arrow[ur] & & & & S \to \lambda \arrow[ul]
\end{tikzcd}$$

Another Example

- Grammar:
$$S \to aSb$$
$$S \to \lambda$$

- Derivation of sentence : $aabb$

$$\begin{tikzcd}
S & \Rightarrow & aSb & \Rightarrow & aaSbb & \Rightarrow & aabb \\
& & S \to aSb \arrow[ul] \arrow[ur] & & & S \to \lambda \arrow[u] &
\end{tikzcd}$$

Another Example

- Language of the grammar

$$S \to aSb$$

$$S \to \lambda$$

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

- This is not a "regular language"
  - No DFA can accept this
  - We will learn one more method to test regular-ness: "Pumping Lemma"

More Notation

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

$V:$	Set of variables
$T:$	Set of terminal symbols
$S:$	Start variable
$P:$	Set of production rules

Example

$$G \qquad S \rightarrow aSb$$

$$S \rightarrow \lambda$$

$$\begin{tikzcd}
 & G=(V,T,S,P) & \\
V=\{S\} \arrow[ur] & T=\{a,b\} \arrow[u] & \\
 & & P=\{S \rightarrow aSb,\ \ S \rightarrow \lambda\} \arrow[uul]
\end{tikzcd}$$

More Notation

$$\begin{array}{l} S \to aSb \\ S \to \lambda \end{array} \quad \Longrightarrow \quad S \to aSb \mid \lambda$$

$$\begin{array}{l} \langle article \rangle \to a \\ \langle article \rangle \to the \end{array} \quad \Longrightarrow \quad \langle article \rangle \to a \mid the$$

More Notation

- <u>Sentential Form:</u> A sentence that contains both variables and terminals

- Example:

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

Sentential Forms

sentence

More Notation

- In general we write (similar to extended transition function):

$$w_1 \overset{*}{\Rightarrow} w_n$$

- If: $w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow \cdots \Rightarrow w_n$

- Note: $w \overset{*}{\Rightarrow} w$

Example

- We write:

$$S \overset{*}{\Rightarrow} aaabbb$$

- Instead of:

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

Example

Grammar

$$S \rightarrow aSb$$

$$S \rightarrow \lambda$$

Derivations

$$S \overset{*}{\Rightarrow} \lambda$$

$$S \overset{*}{\Rightarrow} ab$$

$$S \overset{*}{\Rightarrow} aabb$$

$$S \overset{*}{\Rightarrow} aaabbb$$

Another Grammar Example

- Grammar G :

$$S \to Ab$$
$$A \to aAb$$
$$A \to \lambda$$

- Derivations:

$$S \Rightarrow Ab \Rightarrow b$$
$$S \Rightarrow Ab \Rightarrow aAbb \Rightarrow abb$$
$$S \Rightarrow Ab \Rightarrow aAbb \Rightarrow aaAbbb \Rightarrow aabbb$$

More Derivations

$S \to Ab$

$A \to aAb$

$A \to \lambda$

$$S \Rightarrow Ab \Rightarrow aAbb \Rightarrow aaAbbb \Rightarrow aaaAbbbb$$

$$\Rightarrow aaaaAbbbbb \Rightarrow aaaabbbbb$$

$$S \overset{*}{\Rightarrow} aaaabbbbb$$

$$S \overset{*}{\Rightarrow} aaaaaabbbbbbb$$

$$S \overset{*}{\Rightarrow} a^n b^n b$$

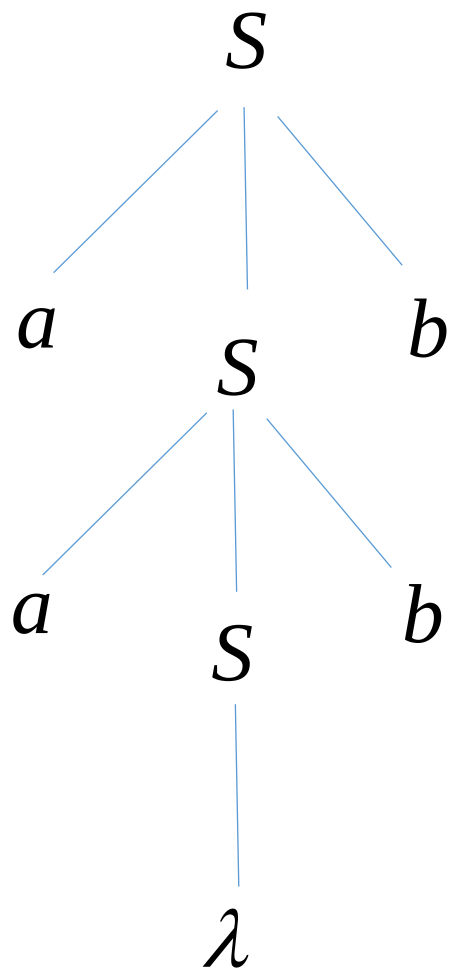
More Notation

- Parse trees: Another representation for derivations where:
  - Each interior node is a variable
  - Each leaf is a variable or terminal or λ
    - If λ then no more child

Example

$S \to aSb$

$S \to \lambda$



$S \overset{*}{\Rightarrow} aabb$

Language of a Grammar

- For a grammar G with start variable S :

$$L(G) = \{\underbrace{w}_{\text{String of terminals}} : \quad S \overset{*}{\Rightarrow} w\}$$

Example

- For grammar G :

$$S \to Ab$$
$$A \to aAb$$
$$A \to \lambda$$

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

Since: $S \overset{*}{\Rightarrow} a^n b^n b$

Outline

- Last week
- Grammars
- Regular Grammars ⬅
- Context-free Grammars
  - Definitions

Linear Grammars

- Grammars with <u>at most one variable</u> at the right side of a production rules

- Examples:

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

$$S \rightarrow Ab$$
$$A \rightarrow aAb$$
$$A \rightarrow \lambda$$

A Non-Linear Grammar

- Grammar G : $S \to SS$

$$S \to \lambda$$

$$S \to aSb$$

$$S \to bSa$$

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

$\nearrow$ Number of a in string w

Another Linear Grammar

- Grammar G :

$$S \to A$$

$$A \to aB \mid \lambda$$

$$B \to Ab$$

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

Right-Linear Grammars

- All productions have form:

$$A \to xB$$

or

$$A \to x$$

string of terminals

- Example:

$$S \to abS$$

$$S \to a$$

Left-Linear Grammars

- All production rules have form:

$$A \to Bx$$

or

$$A \to x$$

string of terminals

- Example:

$$S \to Aab$$

$$A \to Aab \mid B$$

$$B \to a$$

Regular Grammars

- A regular grammar is any right-linear or left-linear grammar

- Examples:

$$G_1$$

$$S \to abS$$

$$S \to a$$

$$G_2$$

$$S \to Aab$$

$$A \to Aab \mid B$$

$$B \to a$$

Observation

- Regular grammars generate regular languages

- Examples:

G_1

$$S \to abS$$

$$S \to a$$

$$L(G_1) = (ab)^* a$$

G_2

$$S \to Aab$$

$$A \to Aab \mid B$$

$$B \to a$$

$$L(G_2) = aab(ab)^*$$

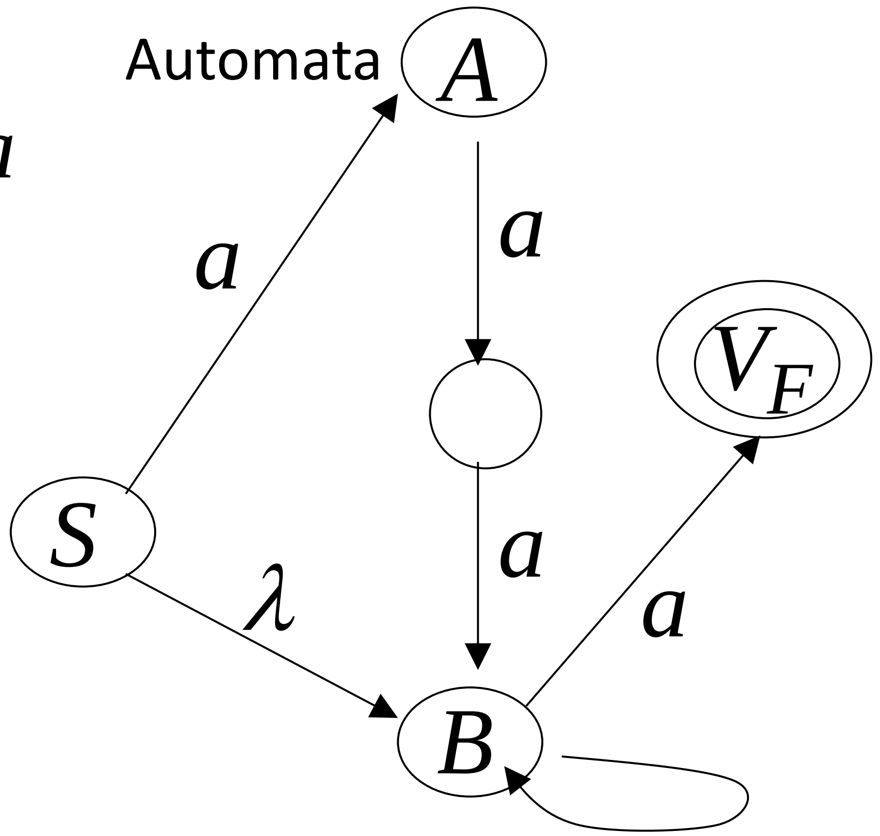
$$\begin{tikzcd}
 & \text{Regular Languages} \arrow[dl, leftrightarrow] \arrow[d, leftrightarrow] \arrow[dr, leftrightarrow] & \\
\text{Regular Grammars} & \text{Regular Expressions} & \text{Finite Automata}
\end{tikzcd}$$

$$S \rightarrow aA \mid B$$

$$A \rightarrow aa\ B$$

$$B \rightarrow bB \mid a$$

$$aaab^{*}a + b^{*}a$$



Grammars in Use
Examples

- Java
  - http://cui.unige.ch/isi/bnf/JAVA/AJAVA.html
- SQL
  - https://ronsavage.github.io/SQL/sql-92.bnf.html#query%20specification

;

Grammars in Use

Lex/flex

```
%%

[0-9]+   { yylval.val = atoi(yytext); return NUM; }
[\+|\-]  { yylval.sym = yytext[0]; return OPA; }
[\*|/]   { yylval.sym = yytext[0]; return OPM; }
"("      { return LP; }
")"      { return RP; }
";"      { return STOP; }
<<EOF>>  { return 0; }
[ \t\n]+ { }
.        { cerr << "Unrecognized token!" << endl; exit(1); }
%%
```

Yacc/bison

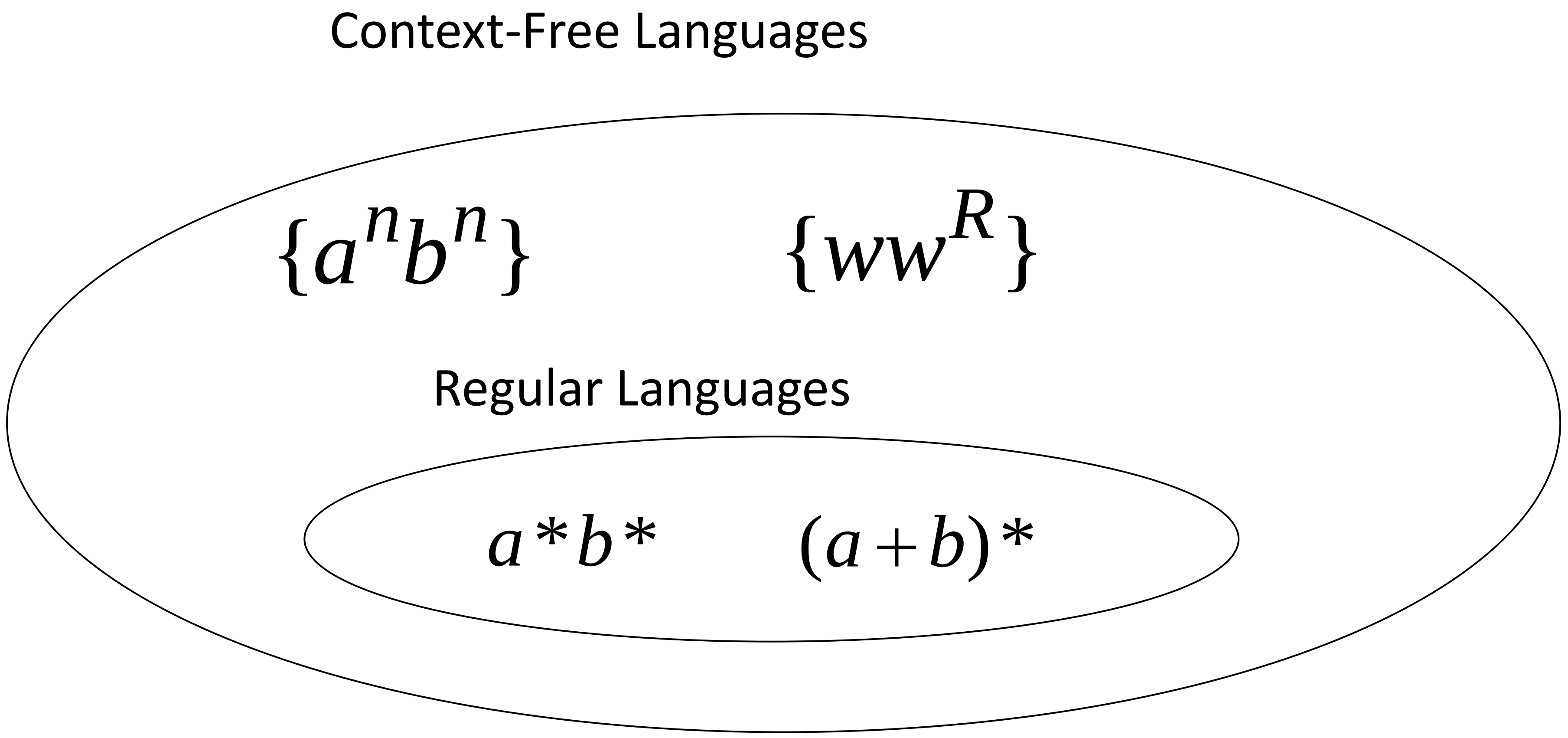
```
%%
run: res run | res    /* forces bison to process many
stmts */
res: exp STOP        { cout << $1 << endl; }

exp: exp OPA term     { $$ = ($2 == '+' ? $1 + $3 : $1 -
$3); }
| term              { $$ = $1; }

term: term OPM factor { $$ = ($2 == '*' ? $1 * $3 : $1
/ $3); }
| sfactor           { $$ = $1; }

sfactor: OPA factor   { $$ = ($1 == '+' ? $2 : -$2); }
| factor            { $$ = $1; }

factor: NUM          { $$ = $1; }
| LP exp RP         { $$ = $2; }
%%
```

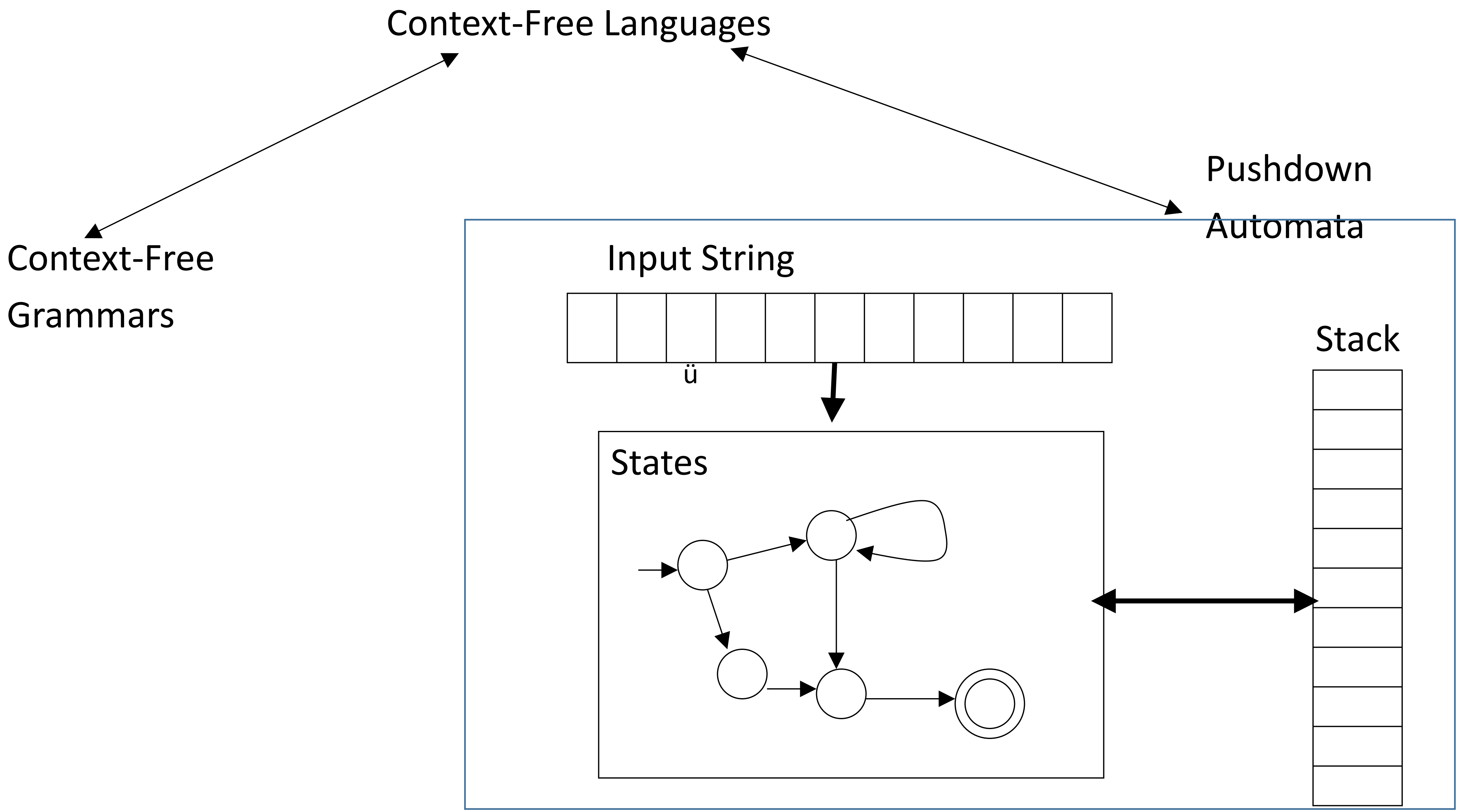
https://www.usna.edu/Users/cs/lmcdowel/courses/si413/F10/labs/L04/calc1/ex1.html

Outline

- Last week
- Grammars
- Regular Grammars
- Context-free Grammars
  - Definitions

Context-Free and Regular Languages




Context-Free Languages
Context-Free Grammars
Pushdown Automata
Input String
ü
Stack
States

Example

A context-free grammar G :

$$S \to aSb$$

$$S \to \lambda$$

Example derivations:

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

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

Describes parentheses in format: (((( ))))

Another Example

- A context-free grammar $:G$

$$S \to aSa$$

$$S \to bSb$$

$$S \to \lambda$$

A derivation:

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

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

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

Another Example

A context-free grammar G :

$$S \rightarrow aSb$$

$$S \rightarrow SS$$

$$S \rightarrow \lambda$$

Example derivations:

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

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

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

Describes open/close parathesis if following format: () ((( ))) (( ))

$$S \to aSb$$

$$S \to SS$$

$$S \to \lambda$$

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

Describes matched parentheses: () ((( ))) (( ))

Definition: Context-Free Grammars

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

$$\begin{tikzcd}
 & G = (V, T, S, P) \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] & & \\
\text{Variables} & \text{Terminal symbols} & \text{Start variable} & \text{Productions of the form:}
\end{tikzcd}$$

Variables

Terminal symbols

Start variable

Productions of the form:

$$A \rightarrow x$$

Variable

String of variables and terminals

Note: There is no constraint on linear-ness

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

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

Definition: Context-Free Languages

- A language L is context-free if and only if there is a context-free grammar G with $L = L(G)$

Derivation Order

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

Leftmost derivation:

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

Rightmost derivation:

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

$$S \to aAB$$

$$A \to bBb$$

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

Leftmost derivation:

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

$$\Rightarrow abbbbB \Rightarrow abbbb$$

Rightmost derivation:

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

$$\Rightarrow abbBbb \Rightarrow abbbb$$

Outline

- Last week
- Grammars
- Regular Grammars
- Context-free Grammars
    - Definitions

Derivation Trees

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

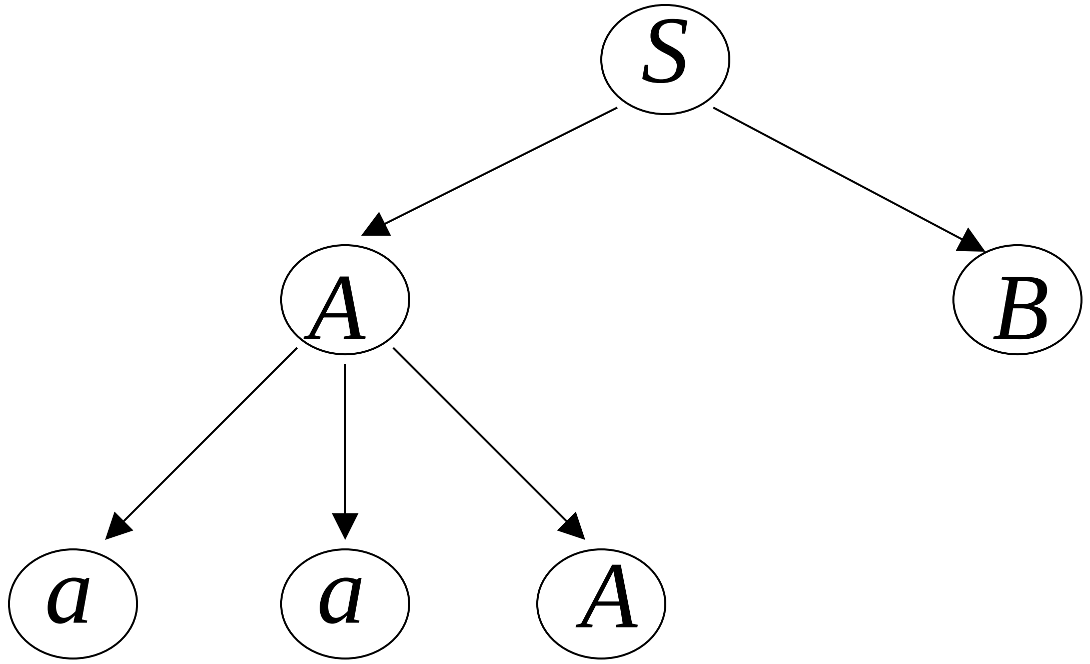
$$S \Rightarrow AB$$

-

$$\begin{tikzcd} & S \arrow[dl] \arrow[dr] & \\ A & & B \end{tikzcd}$$

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

$$S \Rightarrow AB \Rightarrow aaAB$$



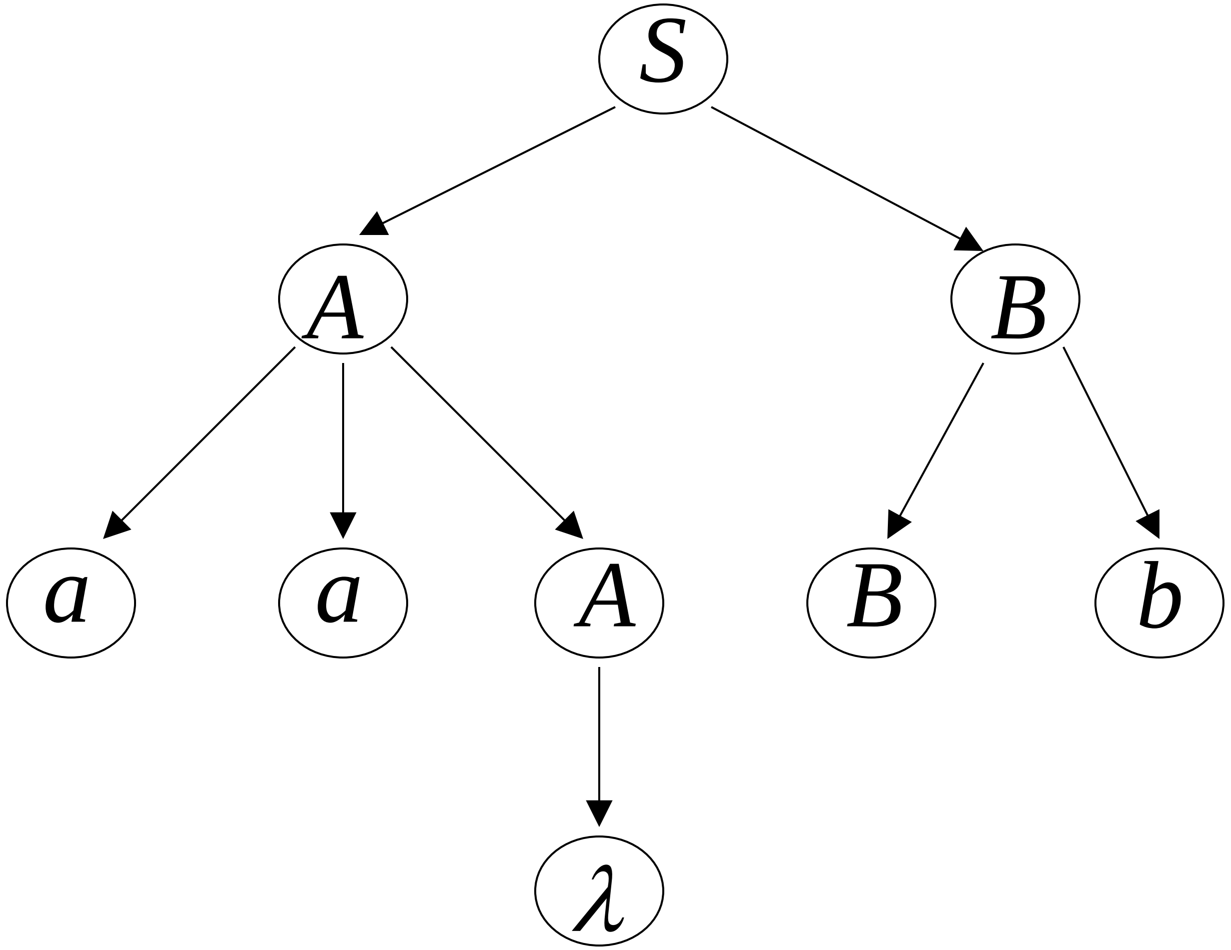
$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$

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

$$\begin{tikzcd}
 & & S \arrow[dl] \arrow[dr] & & \\
 & A \arrow[dl] \arrow[d] \arrow[dr] & & B \arrow[d] \arrow[dr] & \\
a & a & A & B & b
\end{tikzcd}$$

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

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



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

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

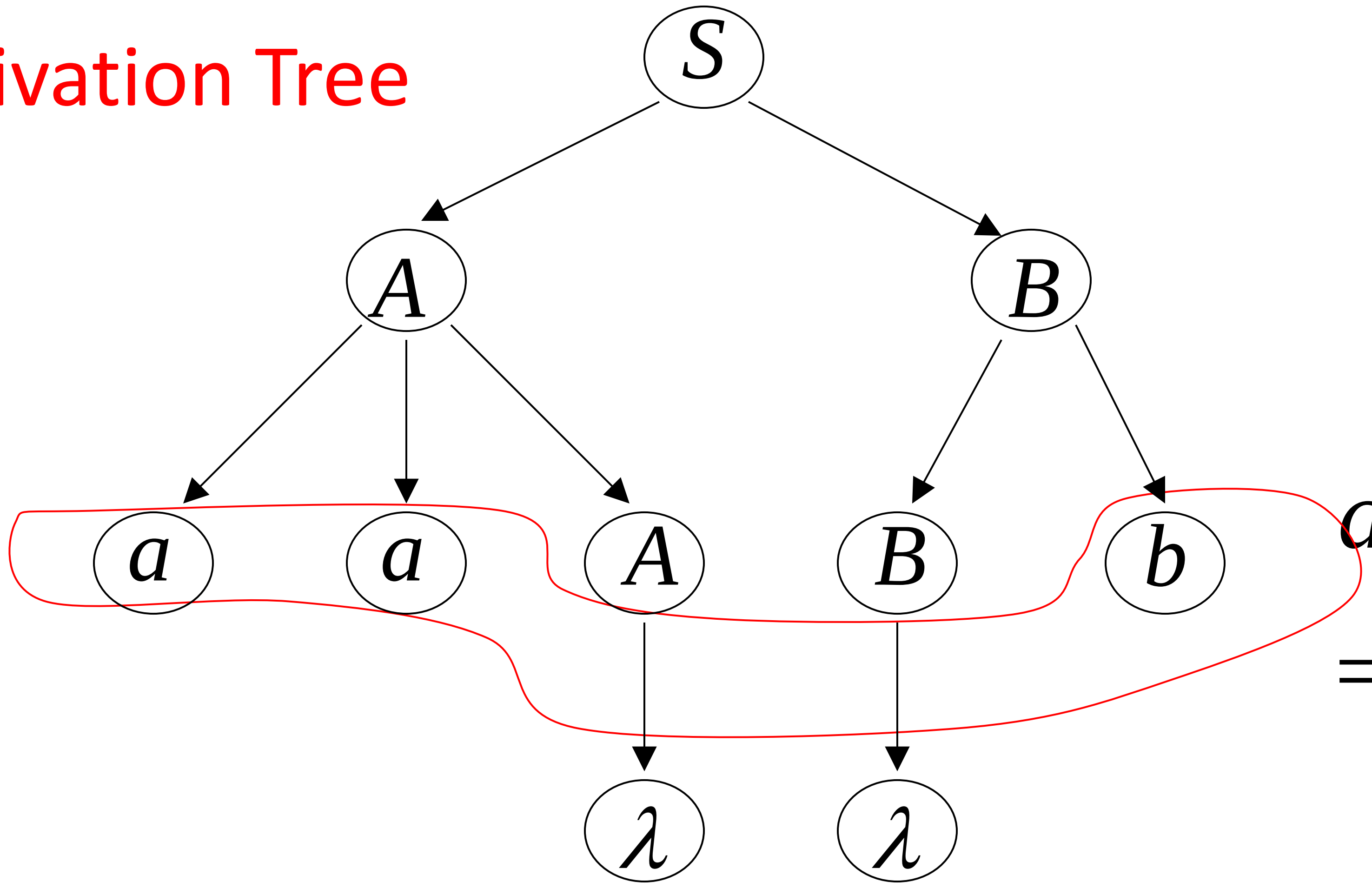
Derivation Tree

$$\begin{tikzcd}
 & & & & S \arrow[dll] \arrow[drrr] & & & & \\
 & & A \arrow[dll] \arrow[d] \arrow[drr] & & & & & B \arrow[dl] \arrow[dr] & \\
a & & a & & A \arrow[d] & & B \arrow[d] & & b \\
 & & & & \lambda & & \lambda & &
\end{tikzcd}$$

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

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

Derivation Tree



yield

$aa\lambda\lambda b$

$= aab$

Partial Derivation Trees

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

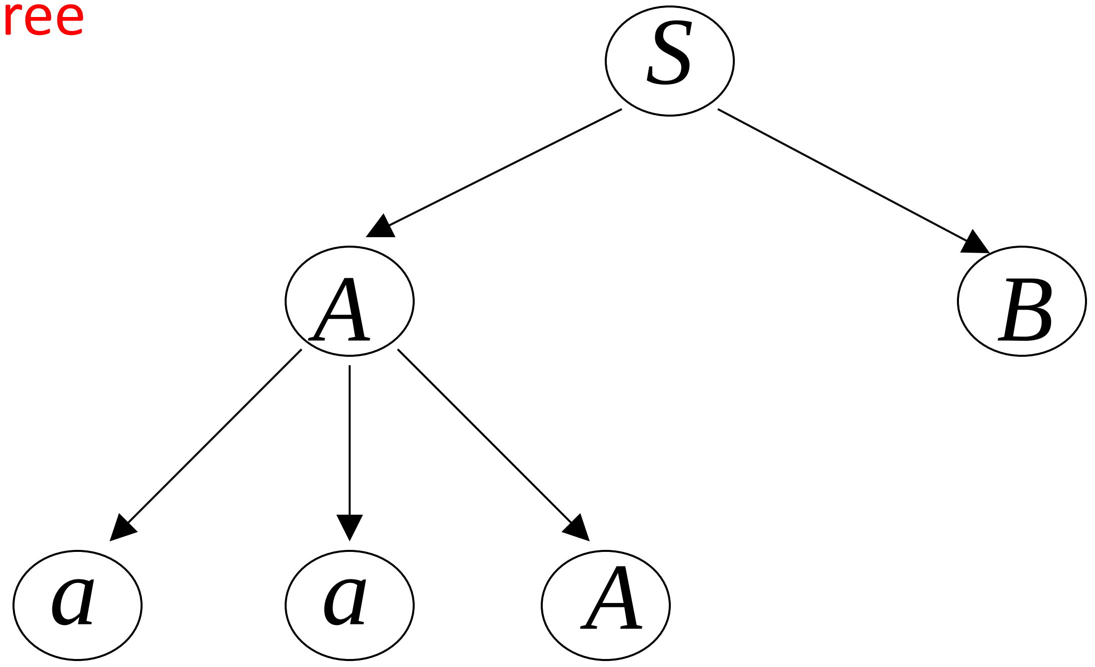
$$S \Rightarrow AB$$

Partial derivation tree

$$\begin{tikzcd}
 & S \arrow[dl] \arrow[dr] & \\
A & & B
\end{tikzcd}$$

$$S \Rightarrow AB \Rightarrow aaAB$$

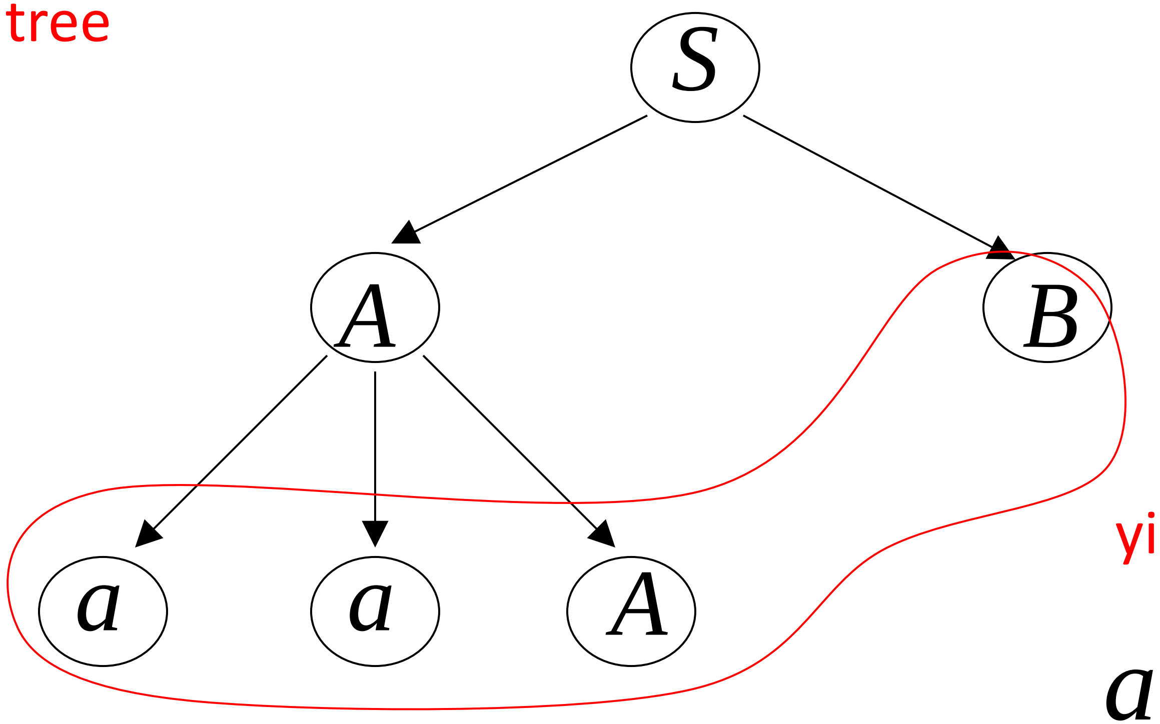
Partial derivation tree



$$S \Rightarrow AB \Rightarrow aaAB$$

sentential form

Partial derivation tree



yield

$aaAB$

Sometimes, derivation order doesn't matter

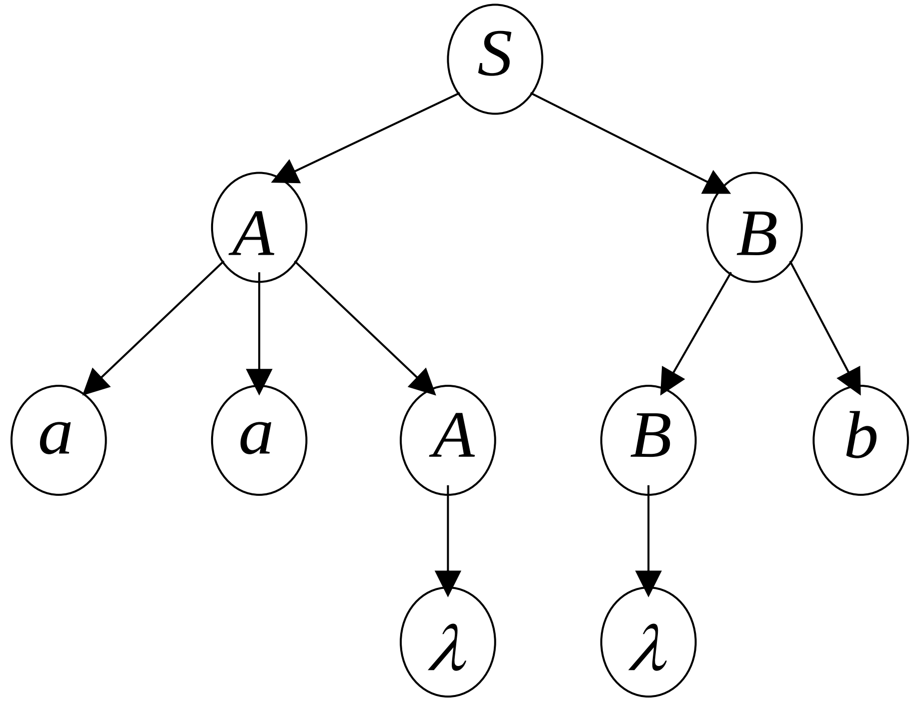
Leftmost:

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

Rightmost:

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

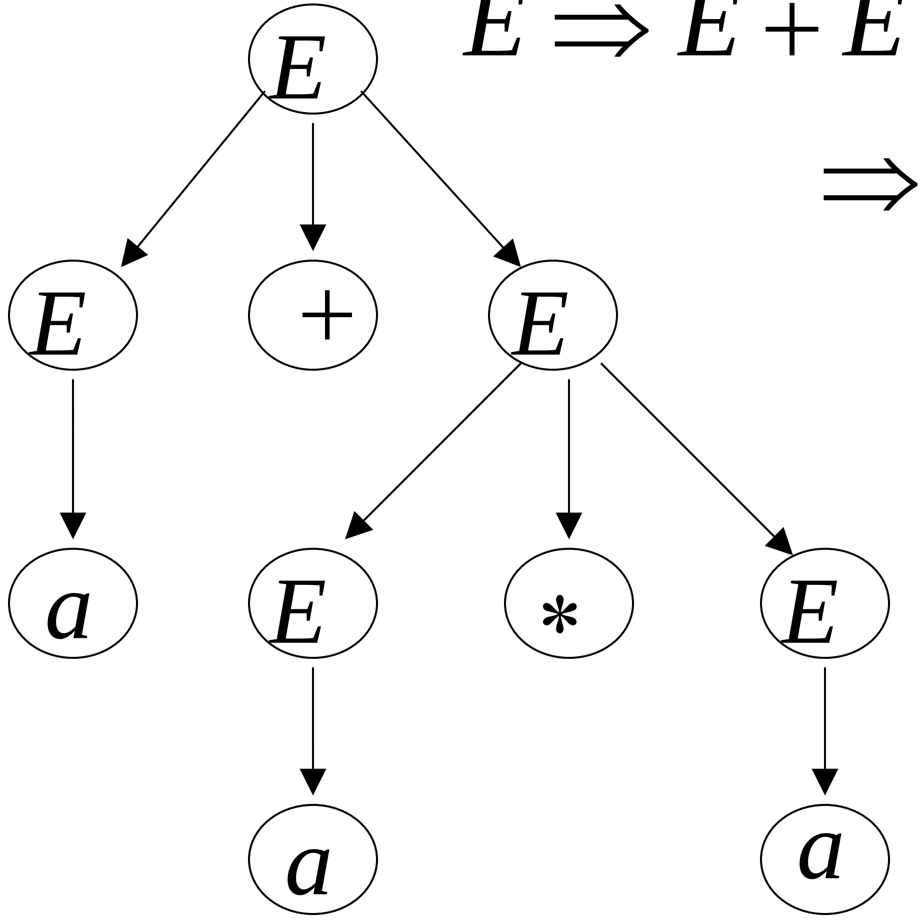
Same derivation tree



Ambiguity

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

$$a + a * a$$



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

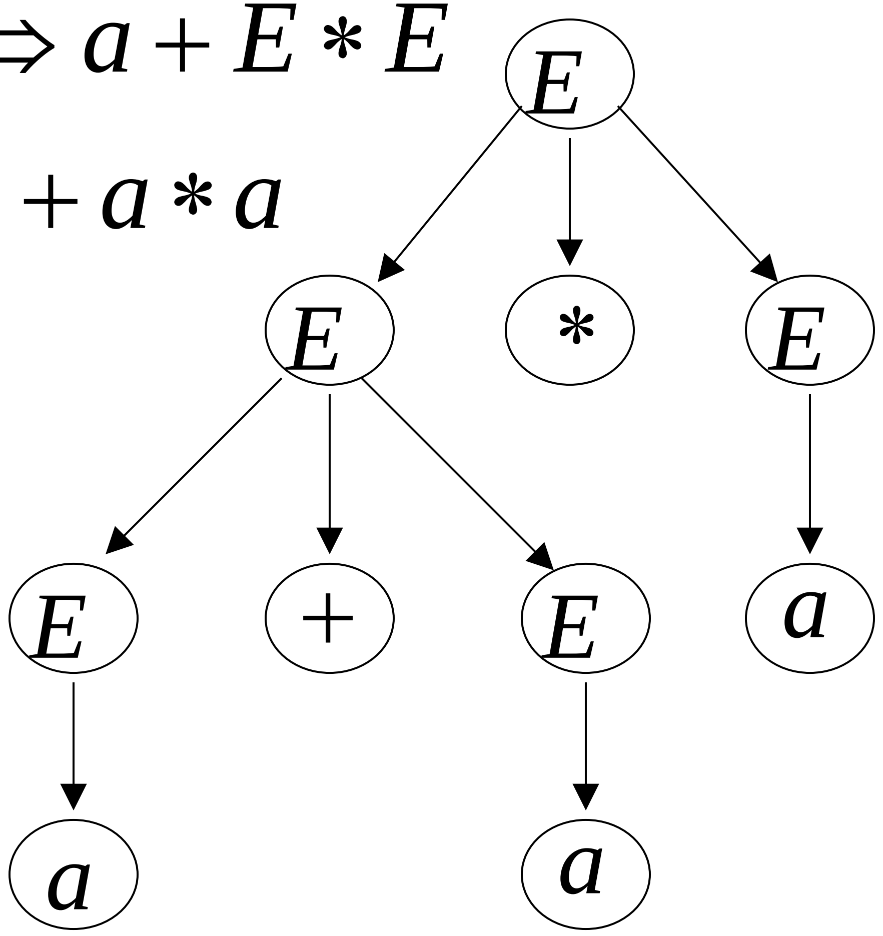
Leftmost derivation

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

$$a + a * a$$

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

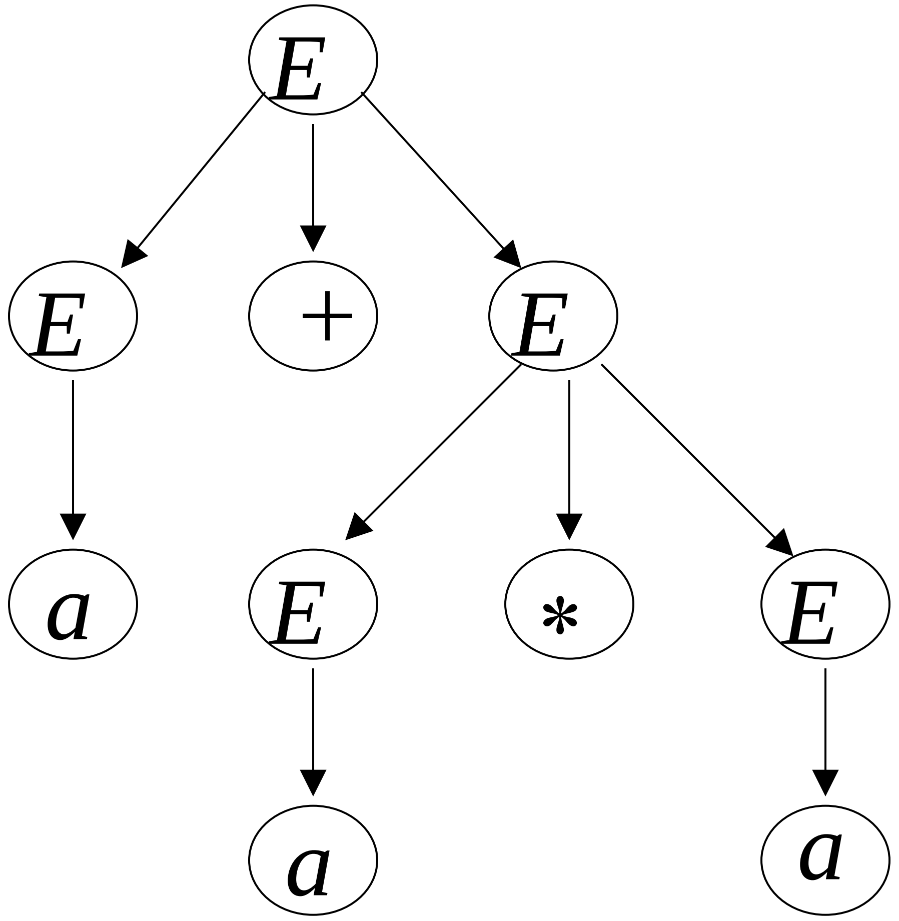
Leftmost derivation

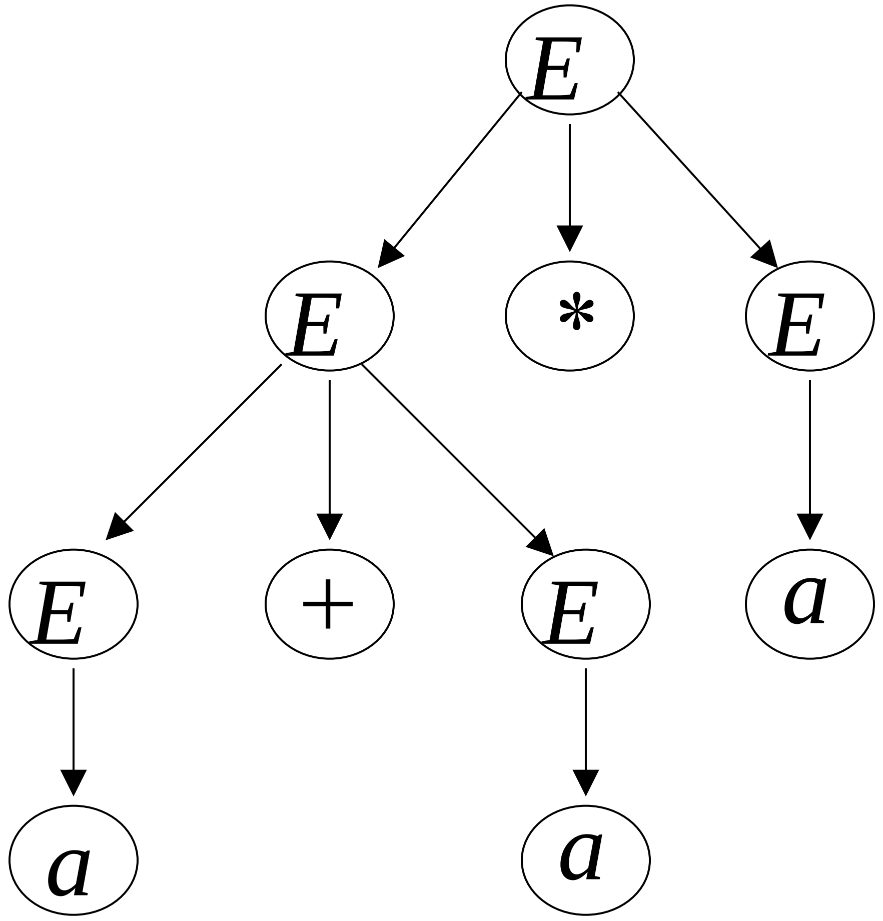


$$E \rightarrow E + E \ \mid E * E \mid (E) \mid a$$

$$a + a * a$$

Two derivation trees

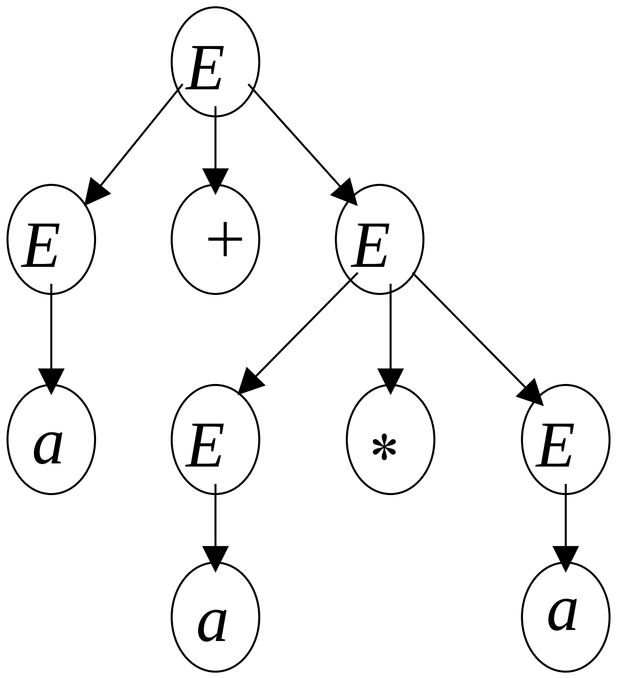


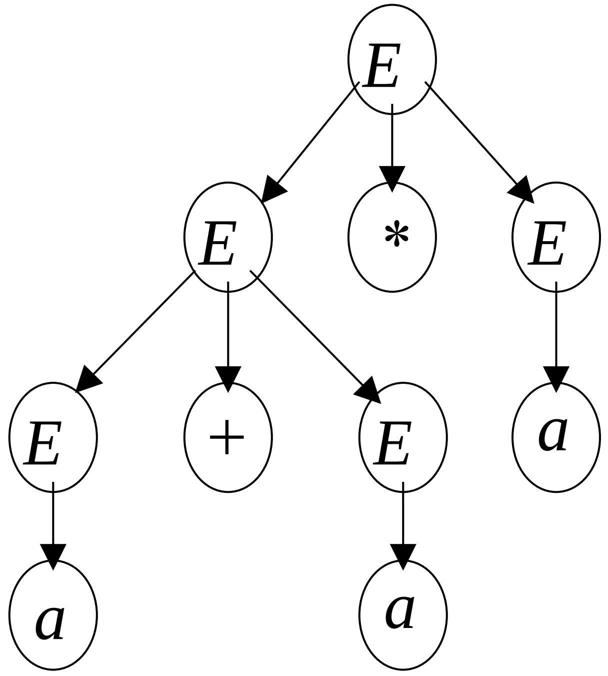


The grammar $E \to E + E \quad | \; E * E \; | \; (E) \; | \; a$

is ambiguous:

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

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

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

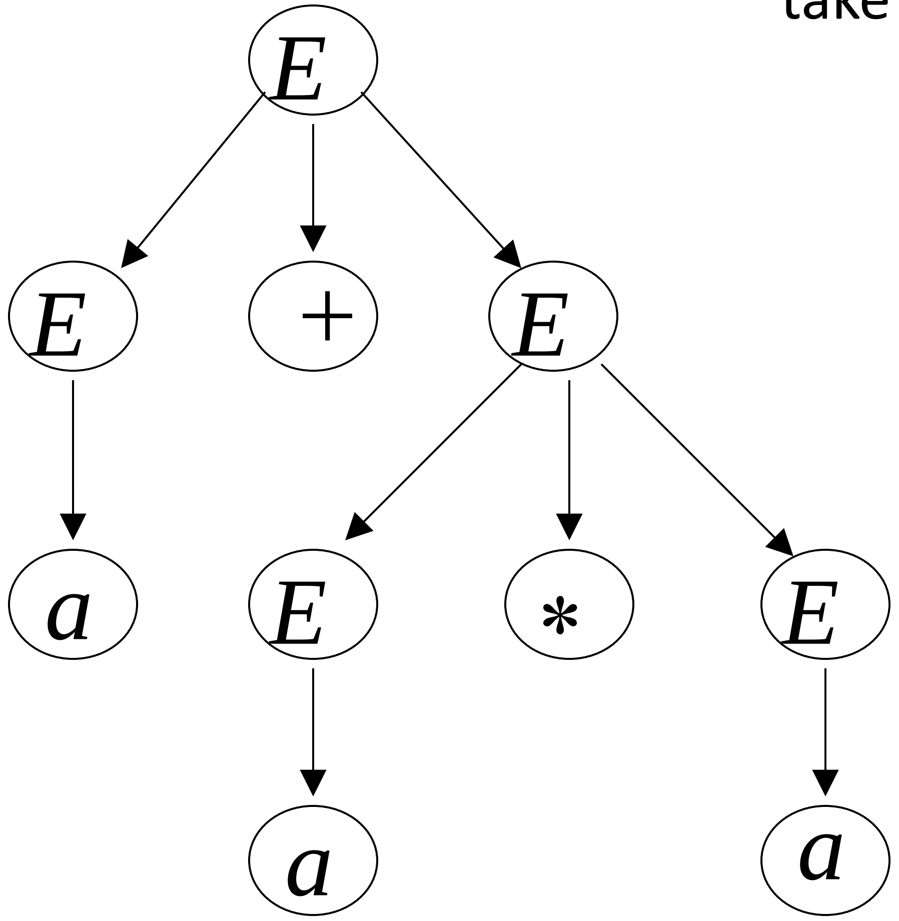
Definition:

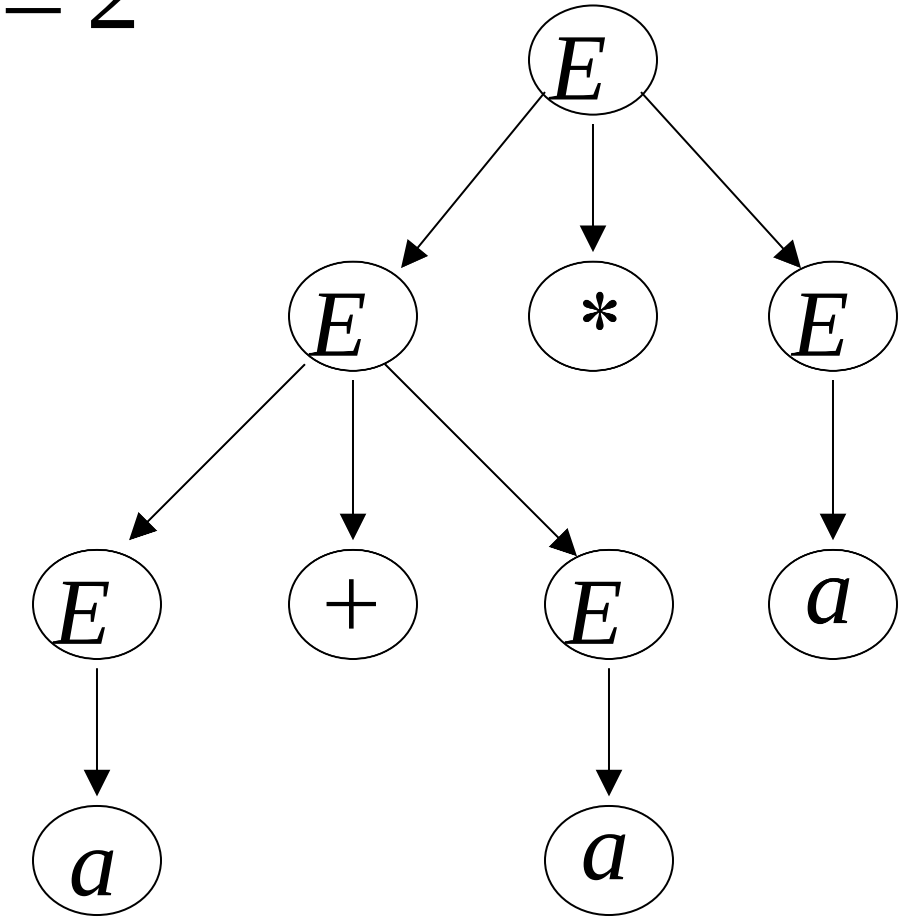
- A context-free grammar G is **ambiguous** if some string $w \in L(G)$ has two or more derivation trees (OR derivations)

Why do we care about ambiguity?

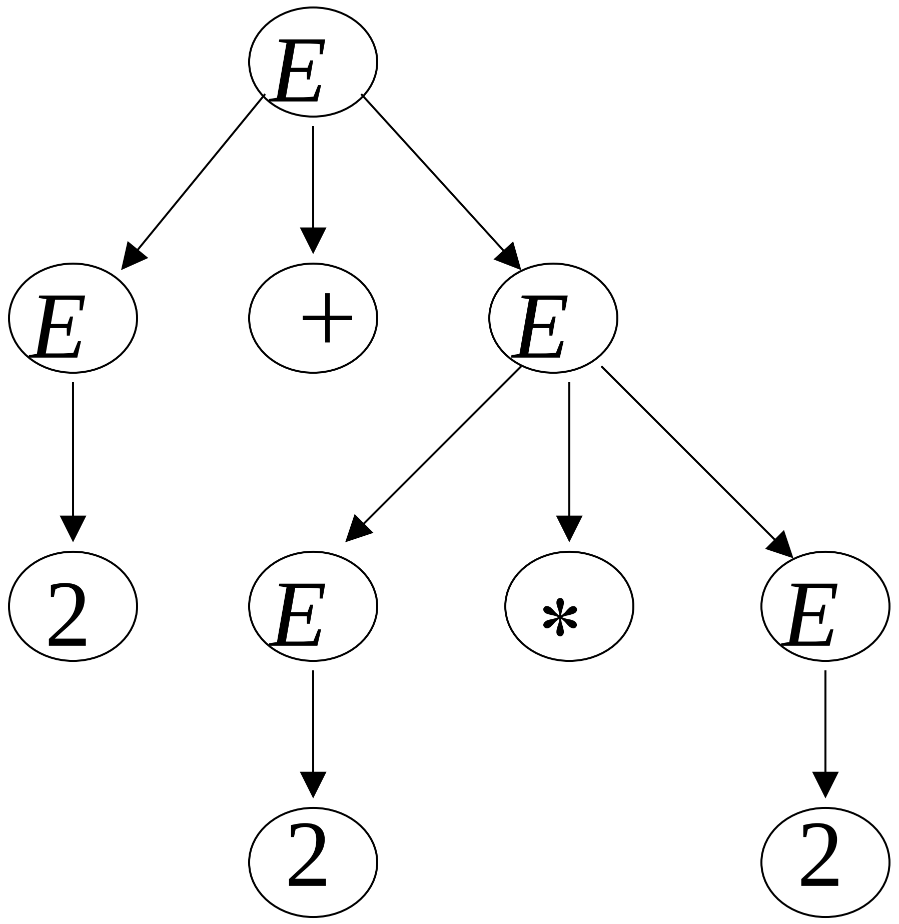
$$a + a * a$$

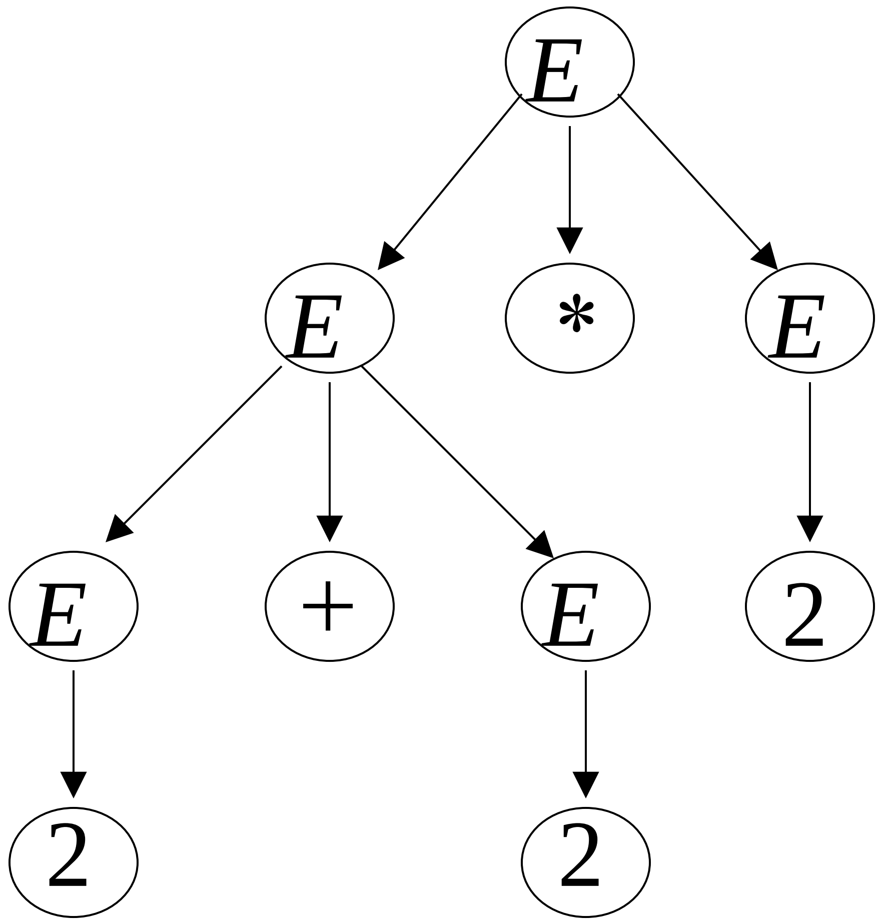
take $a = 2$





$$2 + 2 * 2$$

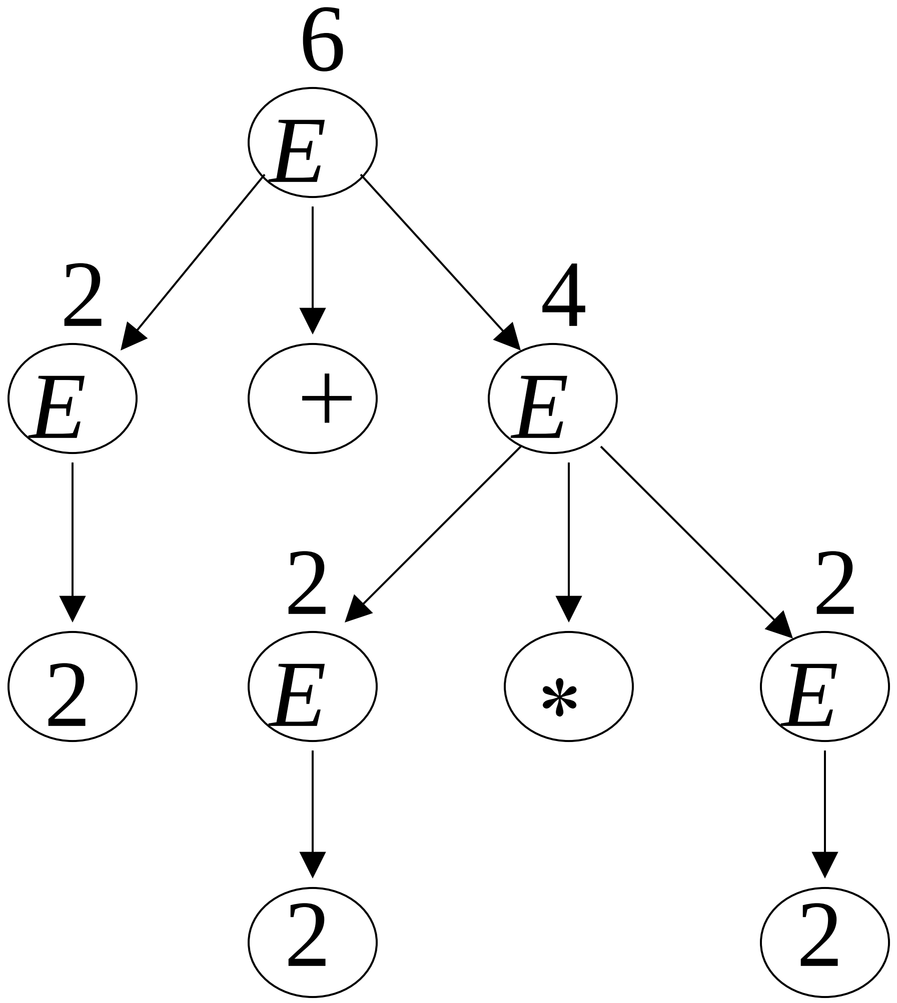




$$2 + 2 \ast 2 = 6 \qquad\qquad 2 + 2 \ast 2 = 8$$

$$\begin{tikzcd}
 & \overset{6}{E} \arrow[dl] \arrow[d] \arrow[dr] & & \\
\overset{2}{E} \arrow[d] & + & \overset{4}{E} \arrow[dl] \arrow[d] \arrow[dr] & \\
2 & \overset{2}{E} \arrow[d] & \ast & \overset{2}{E} \arrow[d] \\
 & 2 & & 2
\end{tikzcd}
\qquad
\begin{tikzcd}
 & & \overset{8}{E} \arrow[dl] \arrow[d] \arrow[dr] & \\
 & \overset{4}{E} \arrow[dl] \arrow[d] \arrow[dr] & \ast & \overset{2}{E} \arrow[d] \\
\overset{2}{E} \arrow[d] & + & \overset{2}{E} \arrow[d] & 2 \\
2 & & 2 &
\end{tikzcd}$$

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



- Ambiguity is **bad** for programming languages
  - What if you have a program to do calculations for orbiting satellites?

Right derivation....

Wrong derivation....

- We want to remove ambiguity

We fix the ambiguous grammar:

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

New non-ambiguous grammar:

$$E \to E + T$$

$$E \to T$$

$$T \to T * F$$

$$T \to F$$

$$F \to (E)$$

$$F \to a$$

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

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

$$a+a*a$$

$$E \rightarrow E+T$$

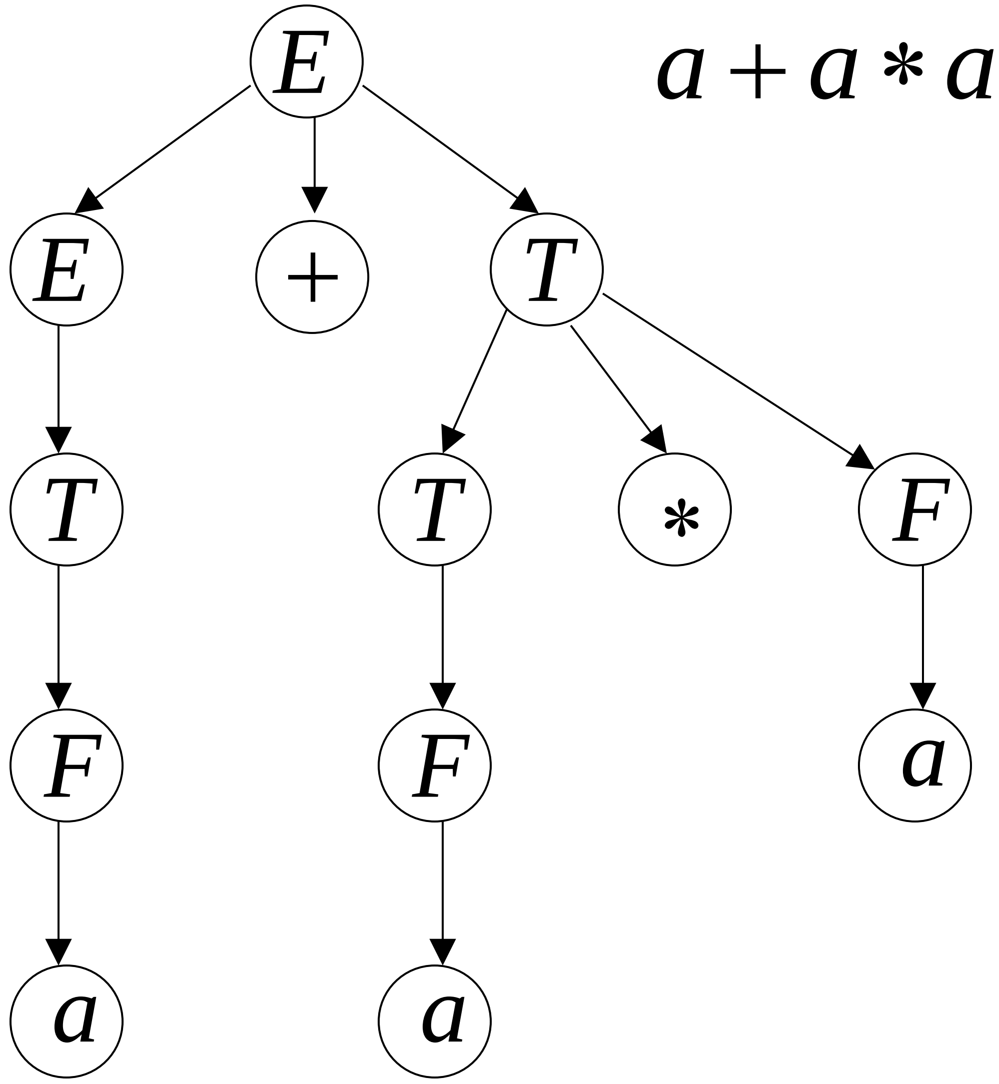
$$E \rightarrow T$$

$$T \rightarrow T*F$$

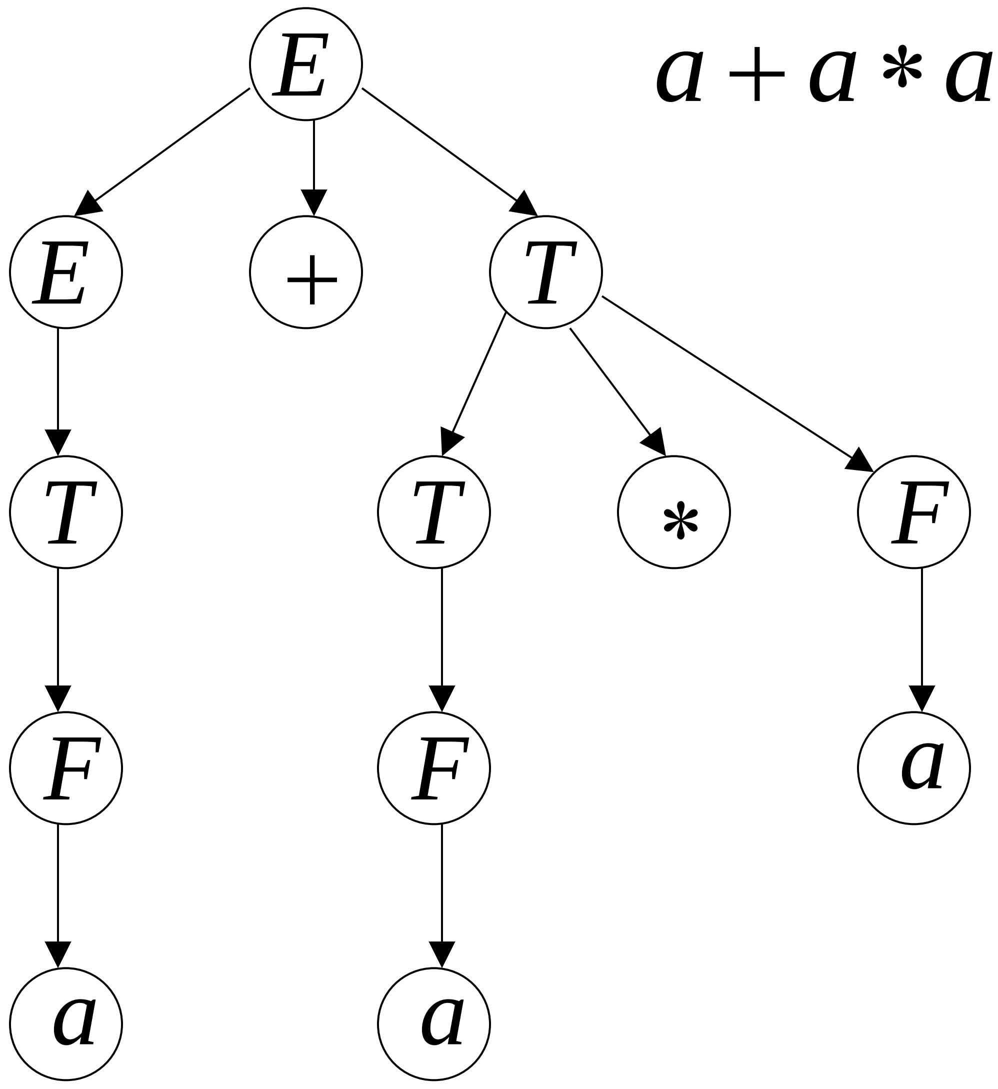
$$T \rightarrow F$$

$$F \rightarrow (E)$$

$$F \rightarrow a$$



Unique derivation tree



$$a + a * a$$

The grammar G:

$$E \to E + T$$

$$E \to T$$

$$T \to T * F$$

$$T \to F$$

$$F \to (E)$$

$$F \to a$$

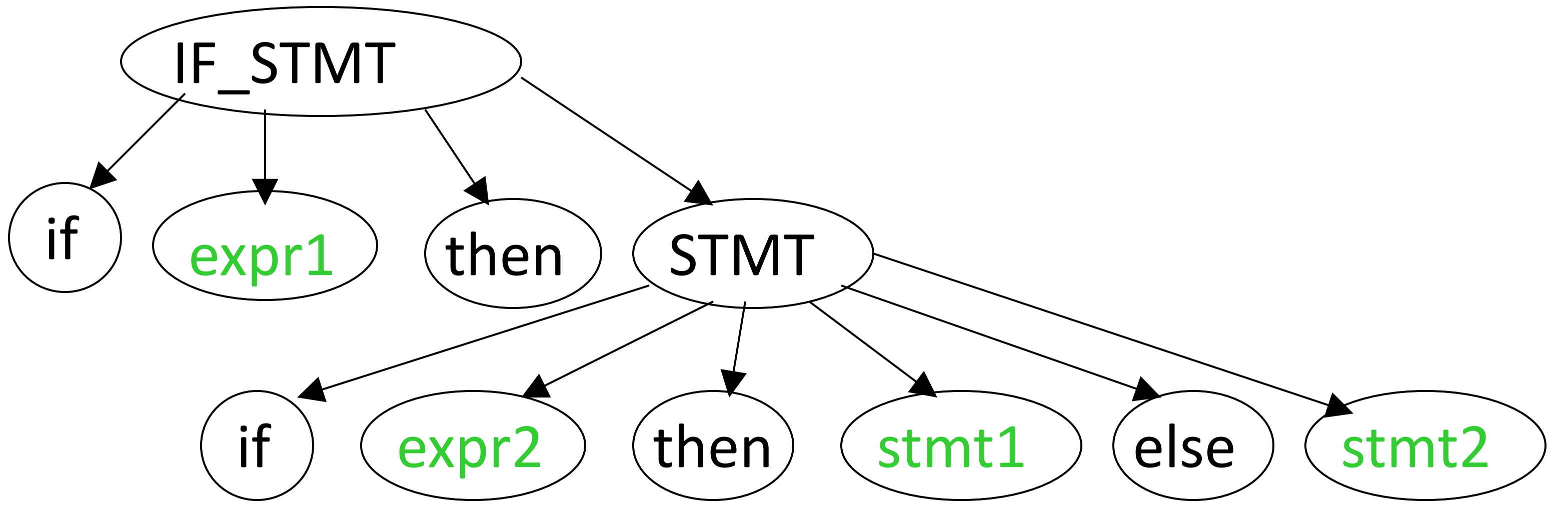
is non-ambiguous:

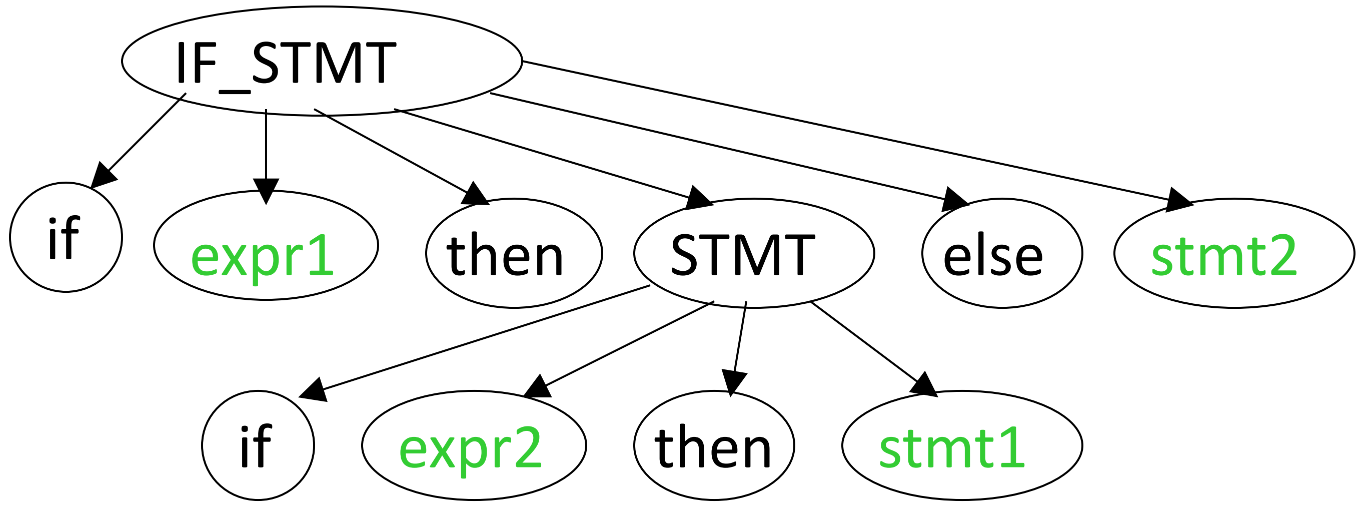
Every string $w \in L(G)$ has a unique derivation tree

Another Ambiguous Grammar

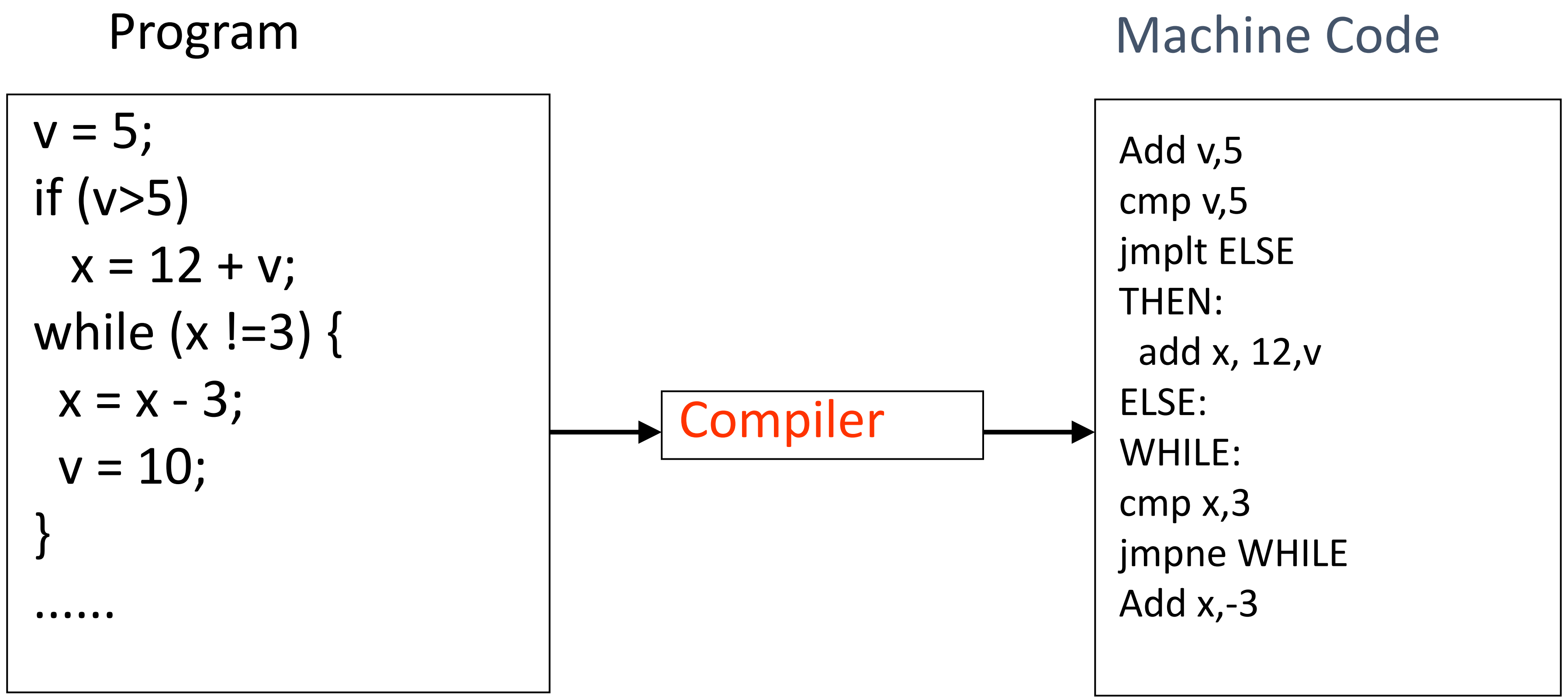
$$\begin{aligned} \text{IF\_STMT} \;\rightarrow\;& \text{if EXPR then STMT} \\ \mid\;& \text{if EXPR then STMT else STMT} \end{aligned}$$

If expr1 then if expr2 then stmt1 else stmt2





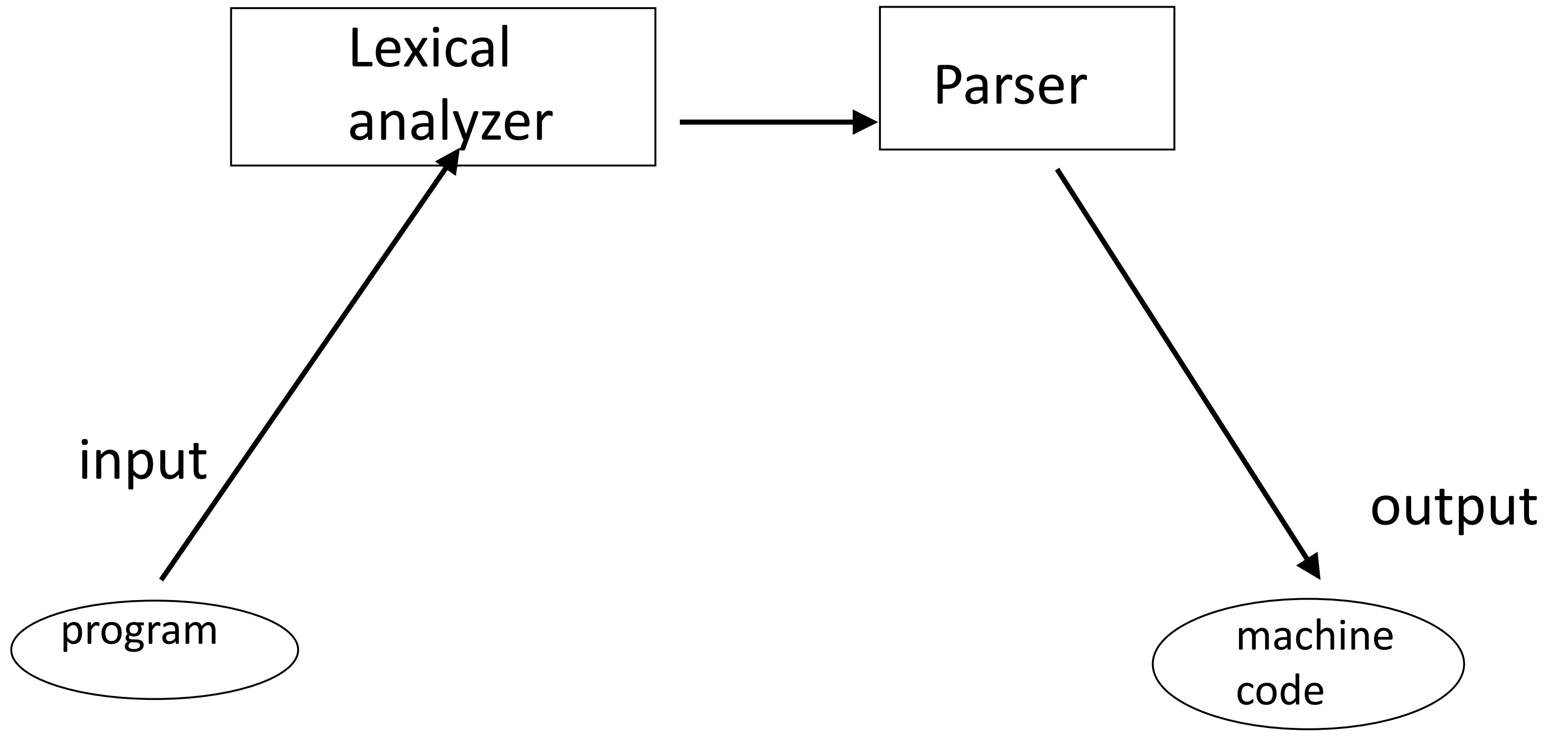
Compilers/Parsers



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

```
Add v,5
cmp v,5
jmplt ELSE
THEN:
  add x, 12,v
ELSE:
WHILE:
cmp x,3
jmpne WHILE
Add x,-3
```

Compiler

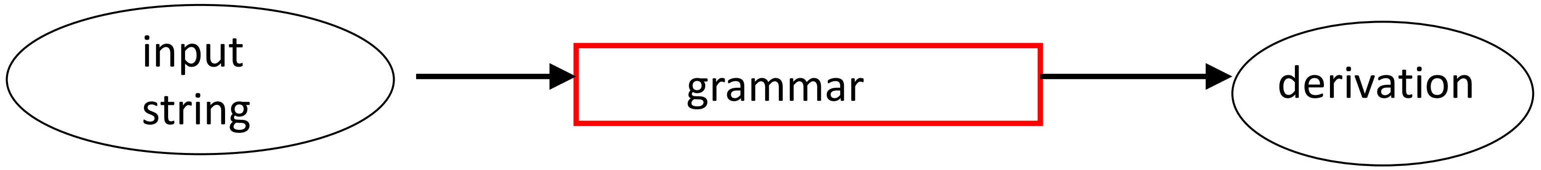

Lexical analyzer
Parser
input
output
program
machine code

Parser

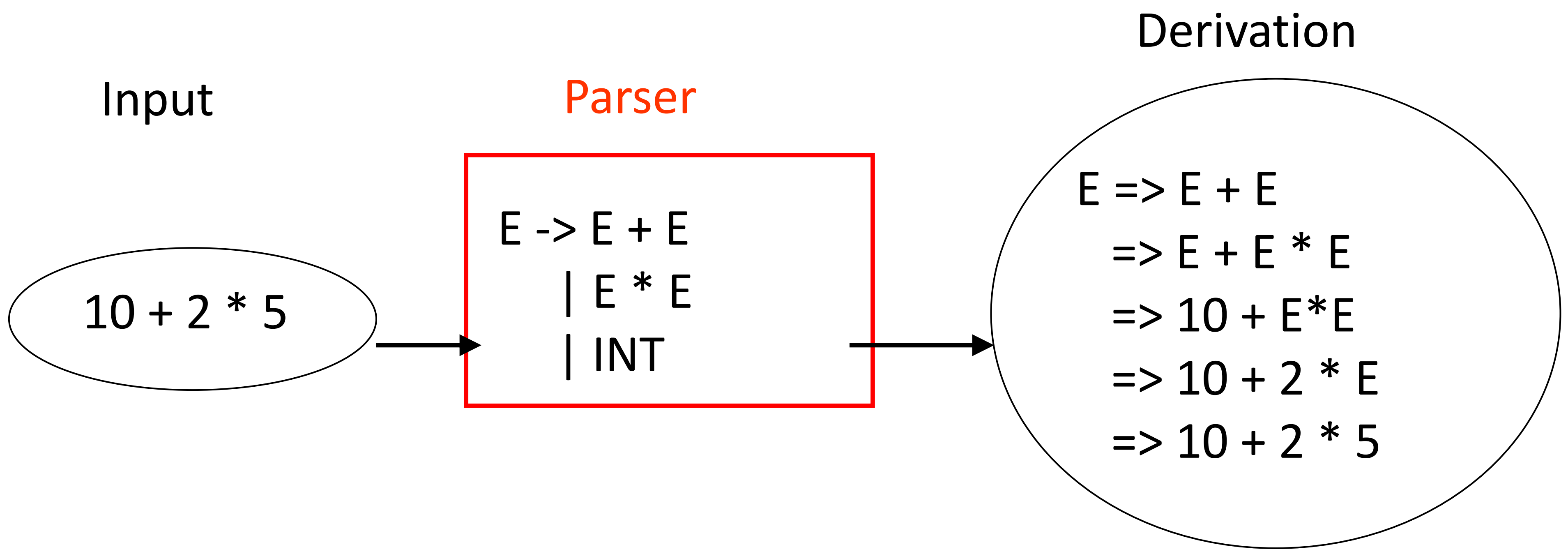
A parser knows the grammar of the programming language

```
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
```

Parser



The parser finds the derivation of a particular input



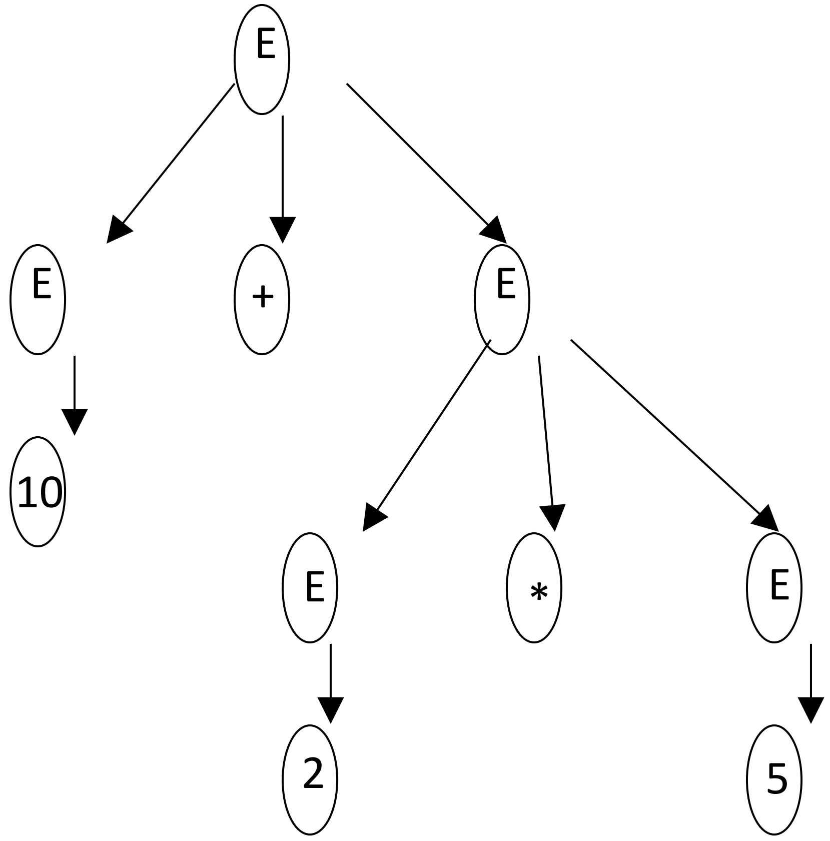
```
E -> E + E
   | E * E
   | INT
```

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

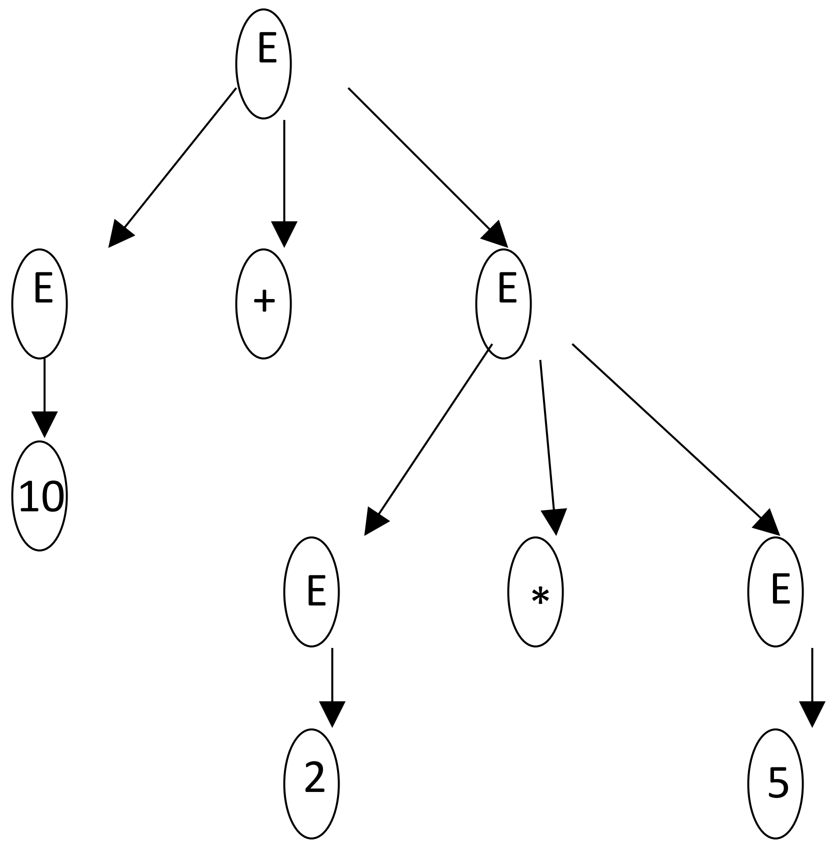
Derivation

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

Derivation tree



Derivation tree

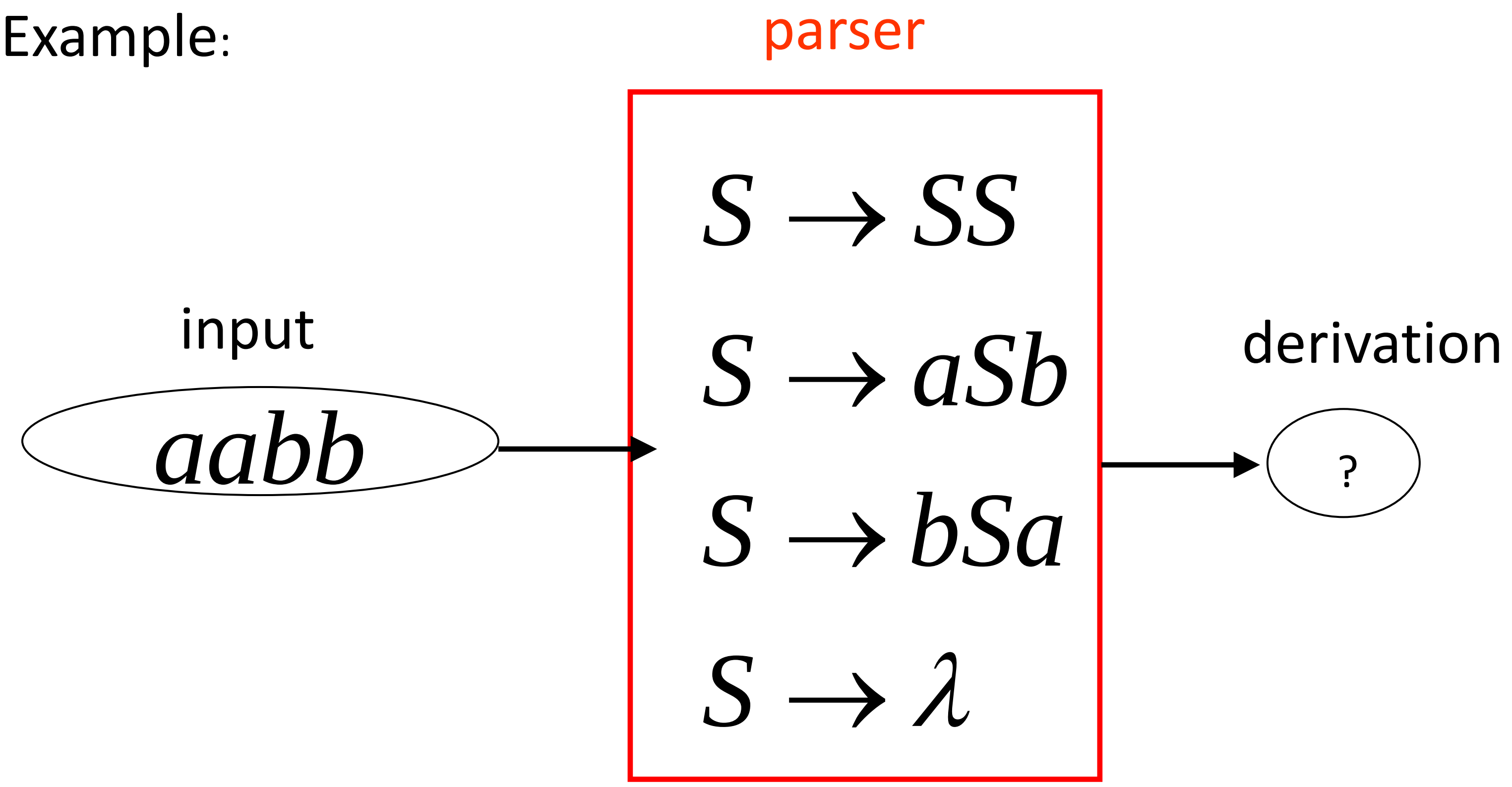


Machine code

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

Parser

Example:



Exhaustive Search

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

Find derivation of $aabb$

Phase 1:

	$S \Rightarrow SS$	$S \Rightarrow SS$
	$S \Rightarrow aSb$	$S \Rightarrow aSb$
	$S \Rightarrow bSa$	$\cancel{S \Rightarrow bSa}$
	$S \Rightarrow \lambda$	$\cancel{S \Rightarrow \lambda}$

All possible derivations of length 1

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

$aabb$

Phase 1

$S \Rightarrow SS$

$S \Rightarrow SS \Rightarrow SSS$

$S \Rightarrow SS \Rightarrow aSbS$

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

$S \Rightarrow SS \Rightarrow S$

$S \Rightarrow aSb$

$S \Rightarrow aSb \Rightarrow aSSb$

$S \Rightarrow aSb \Rightarrow aaSbb$

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

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

Phase 2

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

$$aabb$$

$$S \Rightarrow SS \Rightarrow SSS$$

$$S \Rightarrow SS \Rightarrow aSbS$$

$$S \Rightarrow SS \Rightarrow S$$

$$S \Rightarrow aSb \Rightarrow aSSb$$

Phase 3

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

Final result of exhaustive search (top-down parsing)

Input

$aabb$

Parser

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

Derivation

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

Time complexity of exhaustive search

Suppose there are no productions of the form

$$A \rightarrow \lambda$$

$$A \rightarrow B$$

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

For grammar with k rules

Time for phase 1: k possible derivations

Time for phase 2: k^2 possible derivations

Time for phase $|w|$: $k^{|w|}$ possible derivations

Total time needed for string w:

$$\underbrace{k}_{\text{phase } 1} + \underbrace{k^2}_{\text{phase } 2} + \cdots + \underbrace{k^{|w|}}_{\text{phase } |w|}$$

Pretty bad!!!

There exist faster algorithms for specialized grammars

S-grammar:

$$A \to \underset{\text{symbol}}{a}\,\underset{\text{string of variables}}{x}$$

Pair (A, a) appears once

S-grammar example:

$$S \to aS$$

$$S \to bSS$$

$$S \to c$$

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 : $|w|$

For general context-free grammars:

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

(we will show it in the next class)