Review of CFGs and Parsing - I

Context-free Languages and Grammars

Context-Free Languages

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

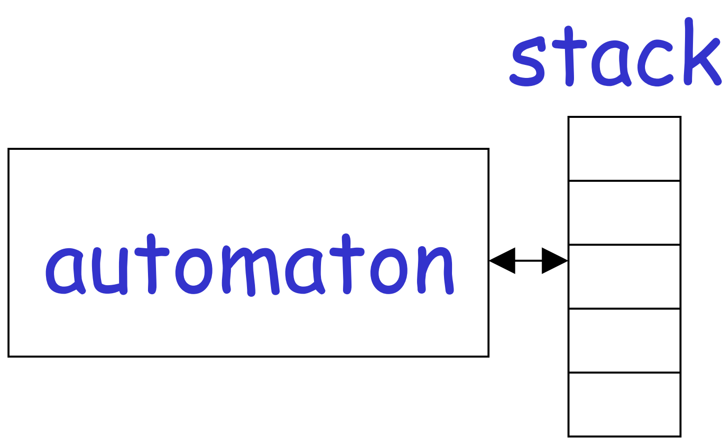
Regular Languages

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

Context-Free Languages

Context-Free Grammars

Pushdown Automata



Context-Free Grammars

Formal Definitions

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

Set of
Variables/
Non-terminal

Set of
terminal
symbols

Start
Symbol

Set of
productions

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

All productions in P are of the form

$$A \rightarrow s$$

Non-terminal

String of
Non-terminal and
terminals

Example of Context-Free Grammar

$$\boxed{S \to aSb \mid \lambda}$$

productions

$$P = \{S \to aSb,\ S \to \lambda\}$$

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

$$V = \{S\}$$

variables

$$T = \{a, b\}$$

terminals

start variable

Language of a Grammar:

For a grammar G with start variable S

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

String of terminals or λ

Example

Sequence of terminals and non-terminals

Grammar: $S \to \overbrace{aSb}$

$S \to \lambda$

Non-terminals

The right side may be λ

Grammar: $S \rightarrow aSb$

$S \rightarrow \lambda$

Derivation of string ab:

$$S \Rightarrow aSb \Rightarrow ab$$

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

Grammar: $S \rightarrow aSb$

$S \rightarrow \lambda$

Derivation of string $aabb$:

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

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

Grammar: $S \rightarrow aSb$

$$S \rightarrow \lambda$$

Language of the grammar:

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

A Convenient Notation

We write: $S \overset{*}{\Rightarrow} aaabbb$

for zero or more derivation steps

Instead of:

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

In general we write: $w_1 \overset{*}{\Rightarrow} w_n$

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

in zero or more derivation steps

Trivially: $w \overset{*}{\Rightarrow} w$

Context-Free Language:

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

Example:

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

is a context-free language

since context-free grammar G :

$$\boxed{S \to aSb \mid \lambda}$$

generates $L(G) = L$

Another Example

Context-free grammar G :

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

Example derivations:

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

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

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

Palindromes of even length

Another Example

Context-free grammar G:

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

Example derivations:

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

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

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

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

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

Describes matched parentheses: $() ((( ))) (( ))$ $\quad a = (, \quad \mathrm{b} =)$

Derivation Order
and
Derivation Trees

Derivation Order

Consider the following example grammar with 5 productions:

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

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

Leftmost derivation order of string *aab*:

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

At each step, we substitute the leftmost variable

$$
\begin{array}{lll}
1.\ S \to AB & 2.\ A \to aaA & 4.\ B \to Bb \\
 & 3.\ A \to \lambda & 5.\ B \to \lambda
\end{array}
$$

Rightmost derivation order of string *aab*:

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

At each step, we substitute the rightmost variable

$$
\begin{array}{lll}
1.\ S \to AB & 2.\ A \to aaA & 4.\ B \to Bb \\
 & 3.\ A \to \lambda & 5.\ B \to \lambda
\end{array}
$$

Leftmost derivation of *aab*:

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

Rightmost derivation of *aab* :

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

Derivation Trees

Consider the same example grammar:

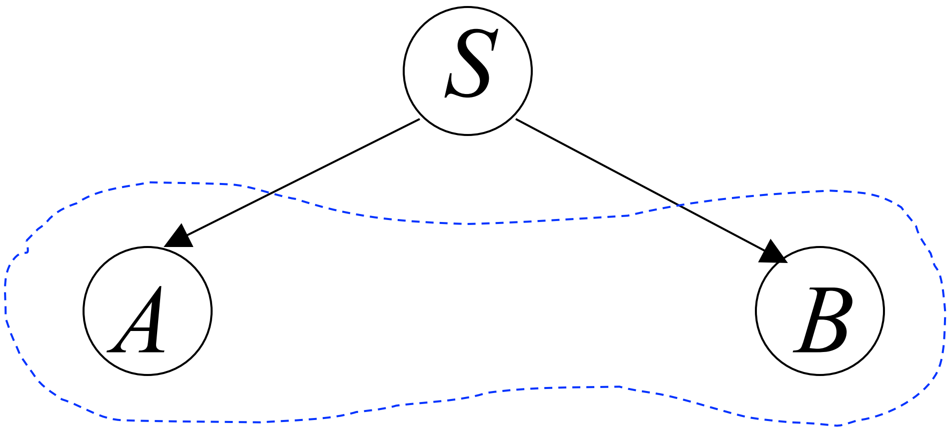
$$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$$

And a derivation of aab:

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

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

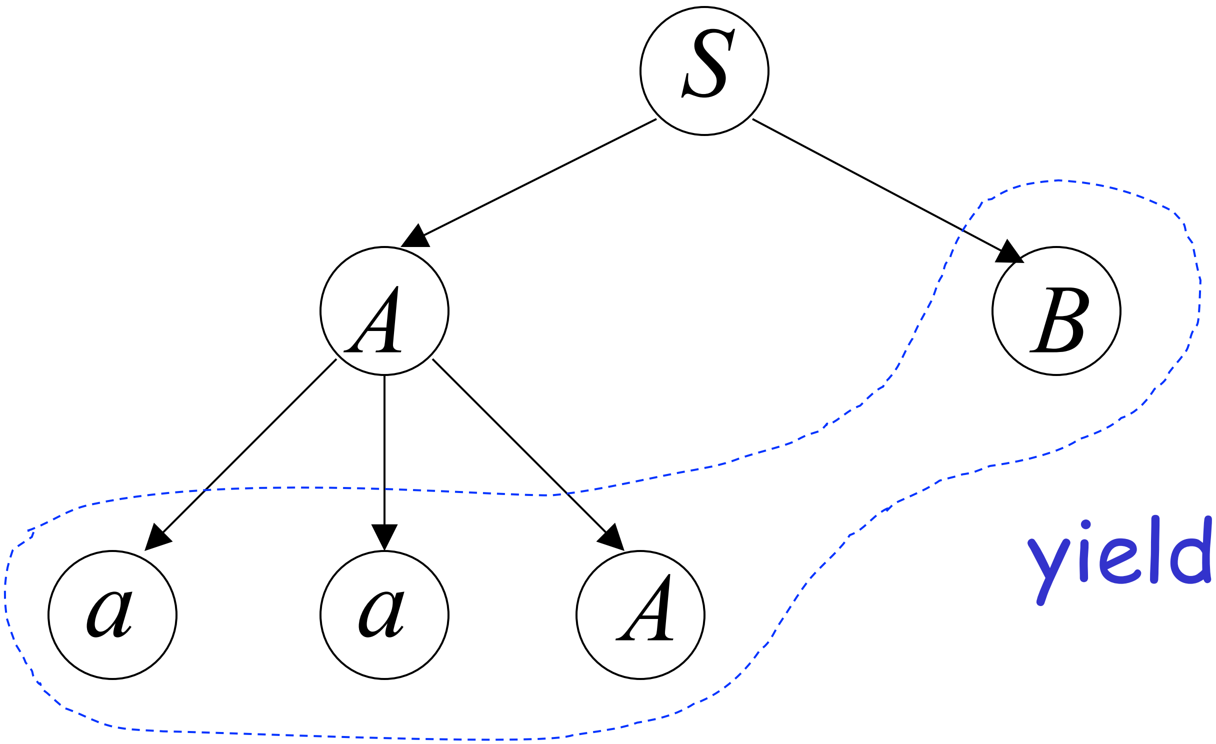
$$S \Rightarrow AB$$



yield AB

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

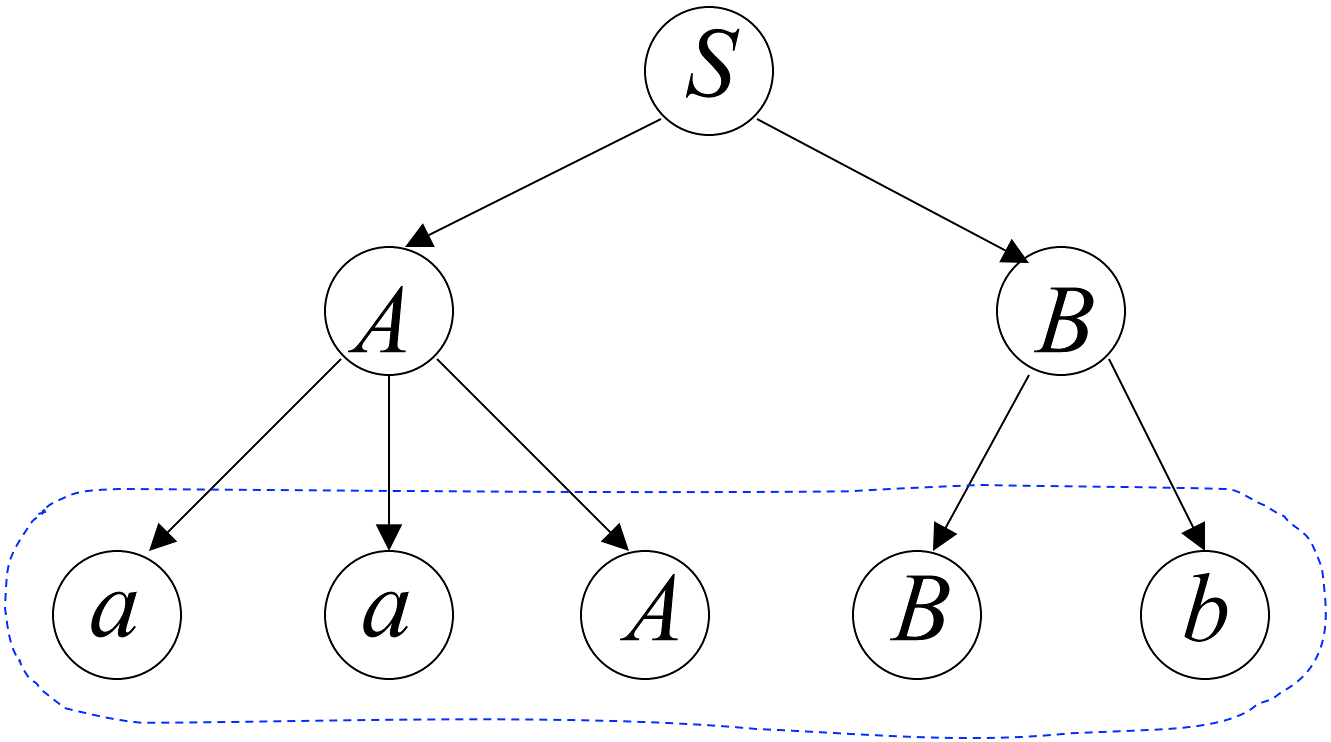
$$S \Rightarrow AB \Rightarrow aaAB$$



yield $aaAB$

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

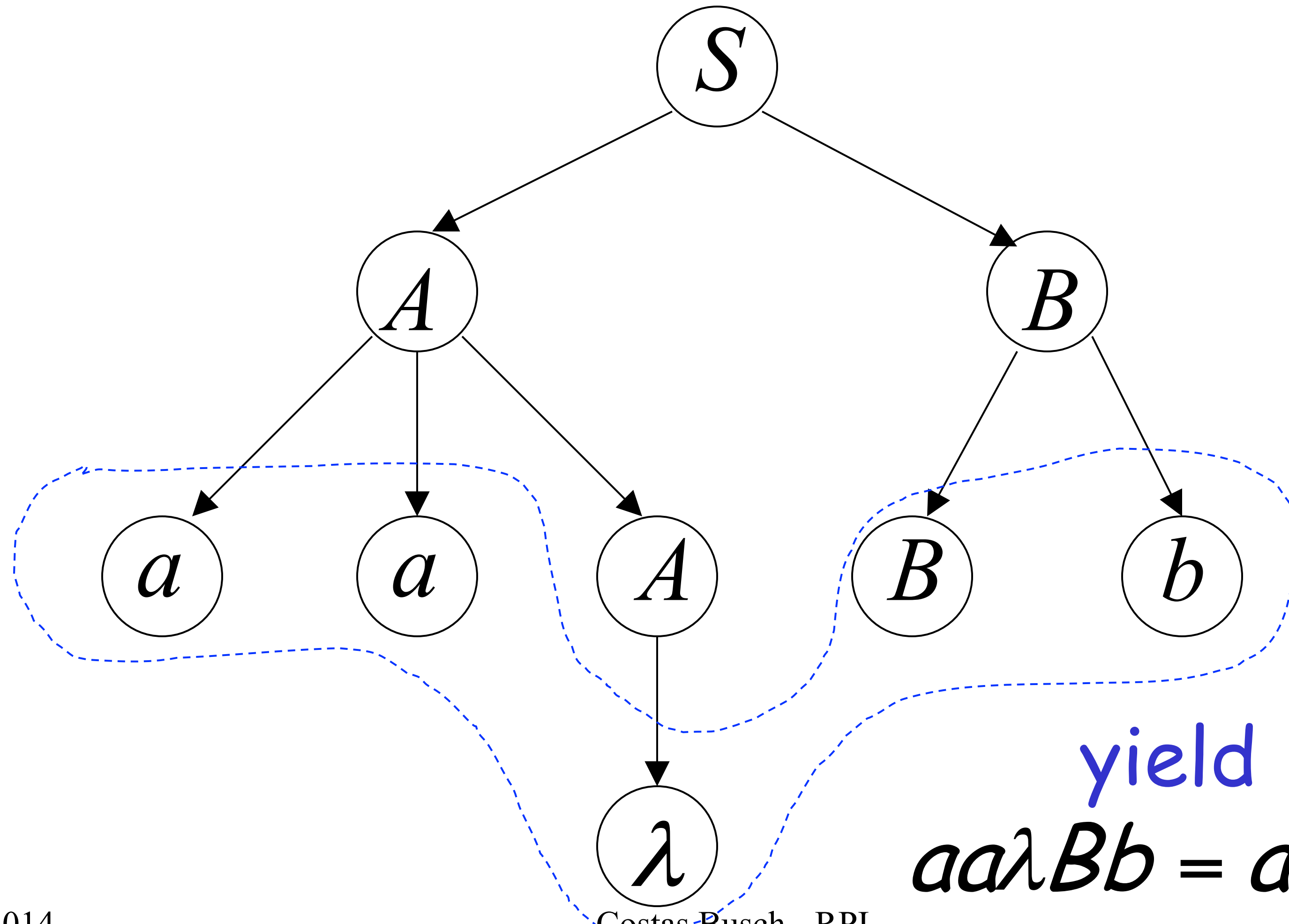
$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb$$



yield *aaABb*

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

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



yield

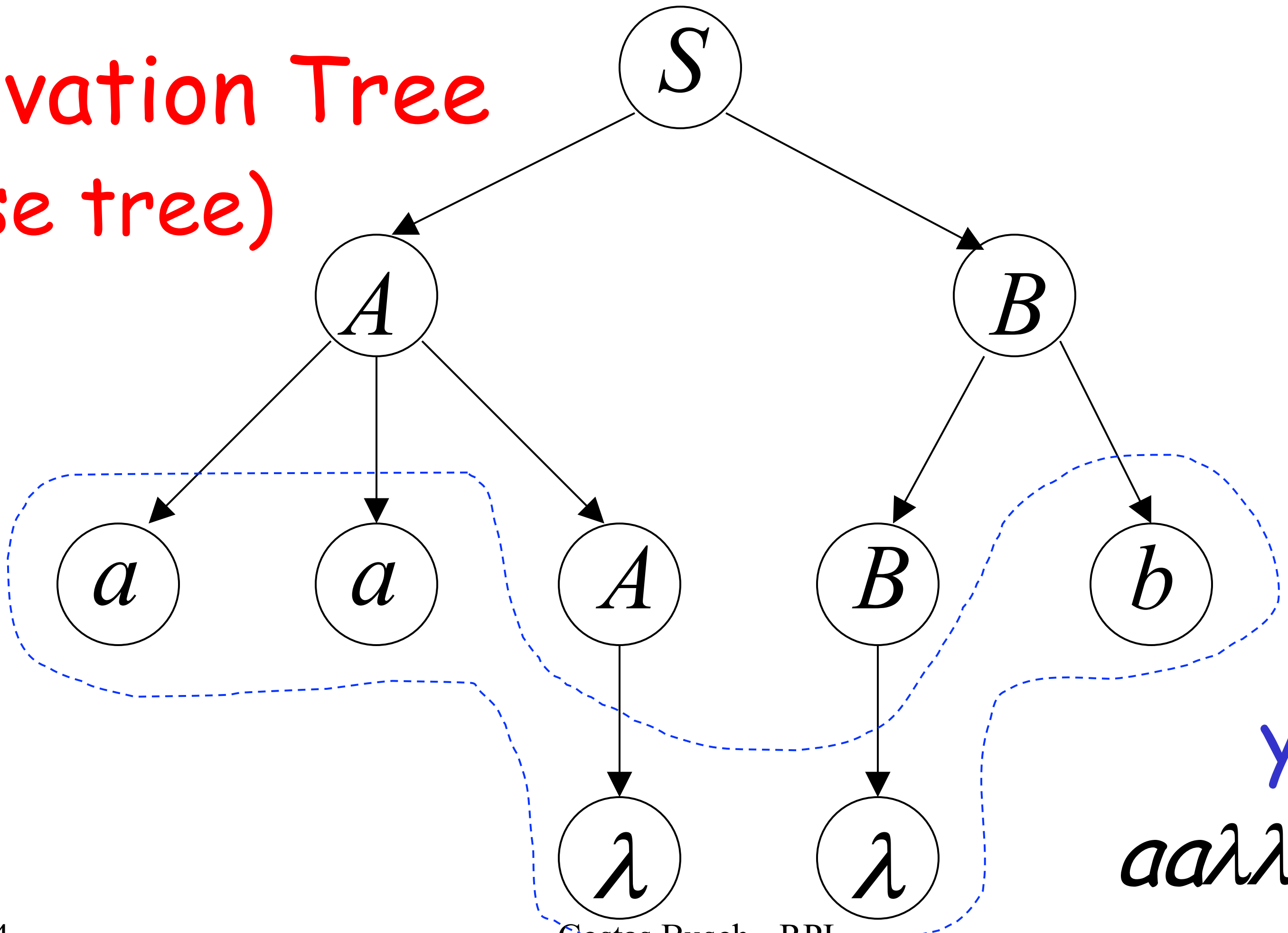
$aa\lambda Bb = aaBb$

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

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

Derivation Tree

(parse tree)



yield

$aa\lambda\lambda b = aab$

Sometimes, derivation order doesn't matter

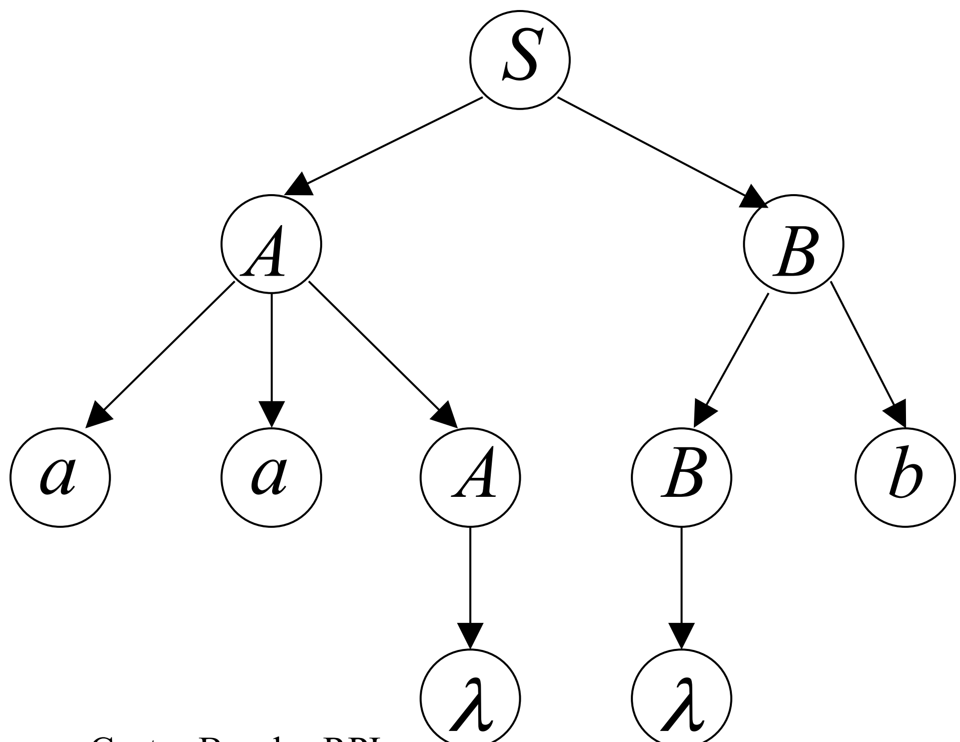
Leftmost derivation:

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

Rightmost derivation:

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

Give same derivation tree



Ambiguity

Grammar for mathematical expressions

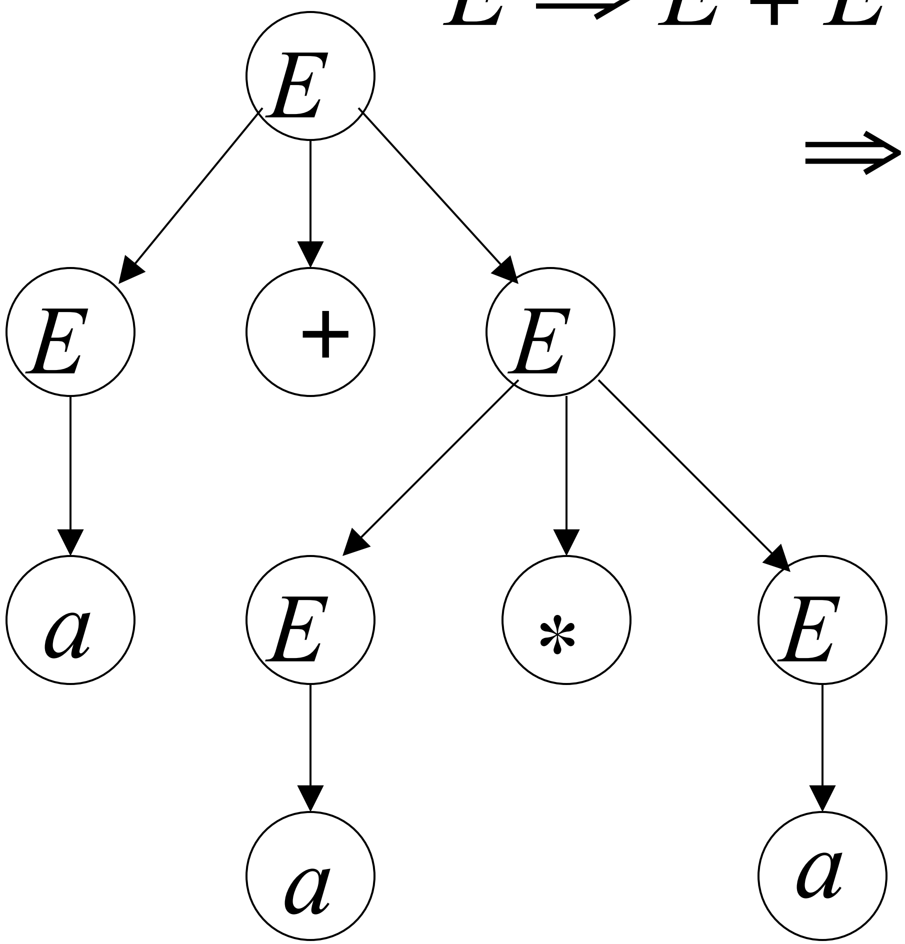
$$\boxed{E \rightarrow E + E \quad | \; E * E \; | \; (E) \; | \; a}$$

Example strings:

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

Denotes any number

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

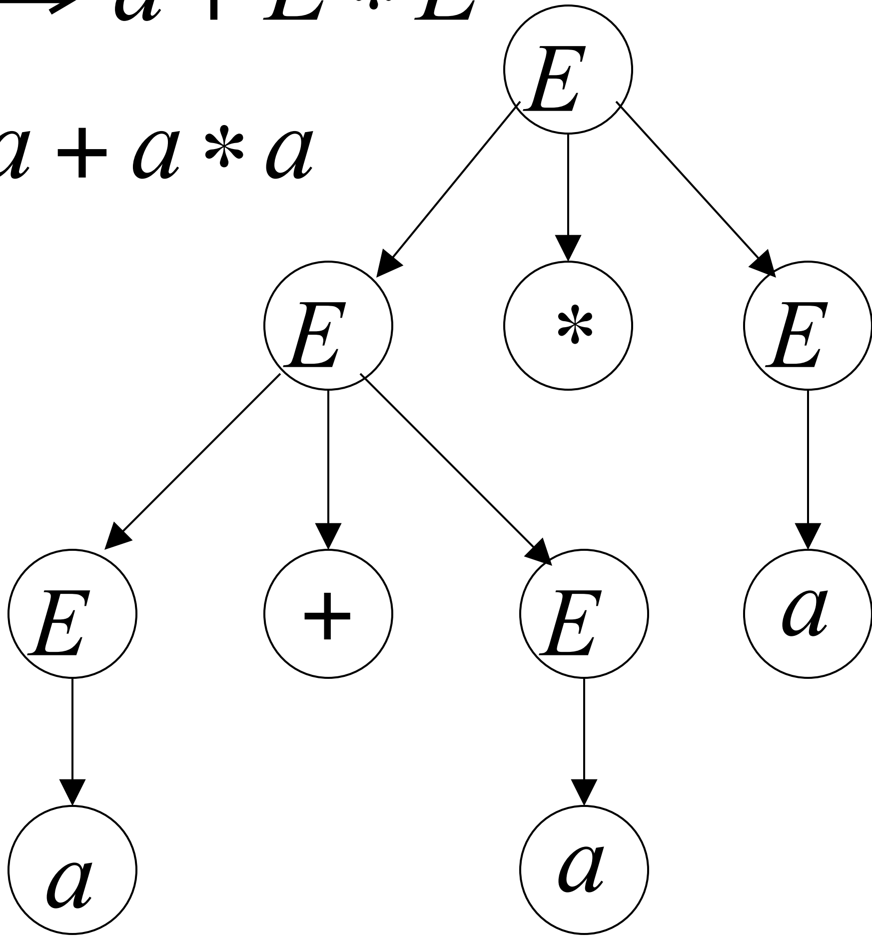


$$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E$$
$$\Rightarrow a + a * E \Rightarrow a + a * a$$

A leftmost derivation for $a + a * a$

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

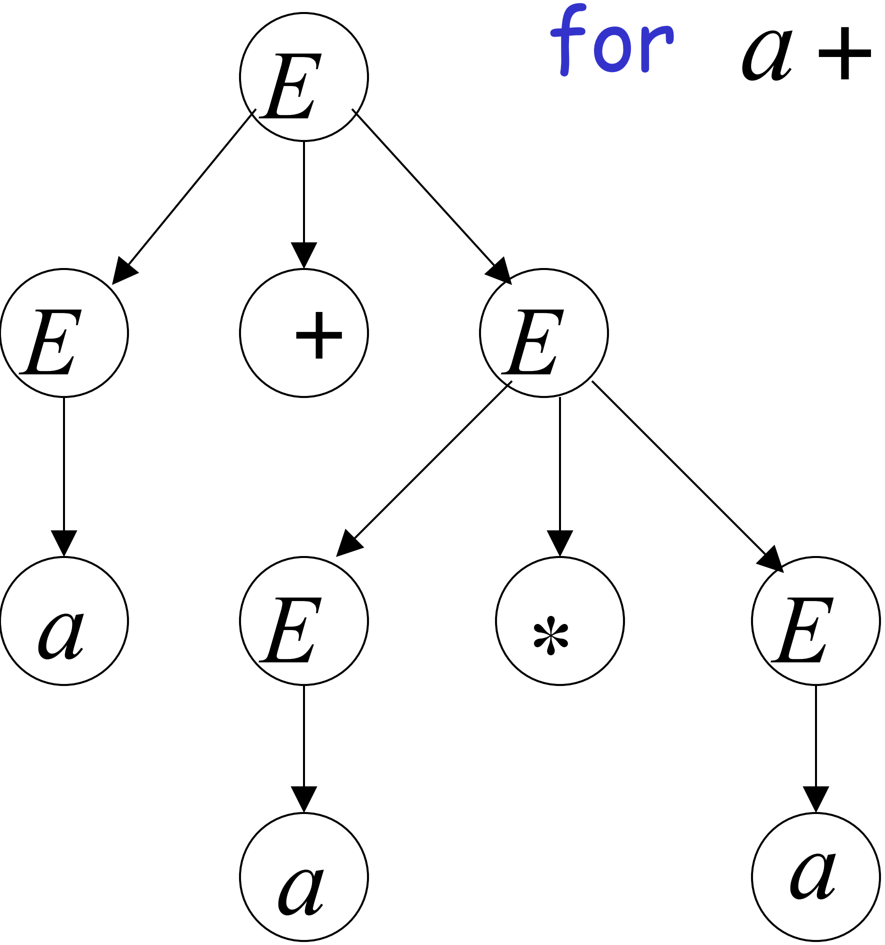
$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E$$
$$\Rightarrow a + a * E \Rightarrow a + a * a$$

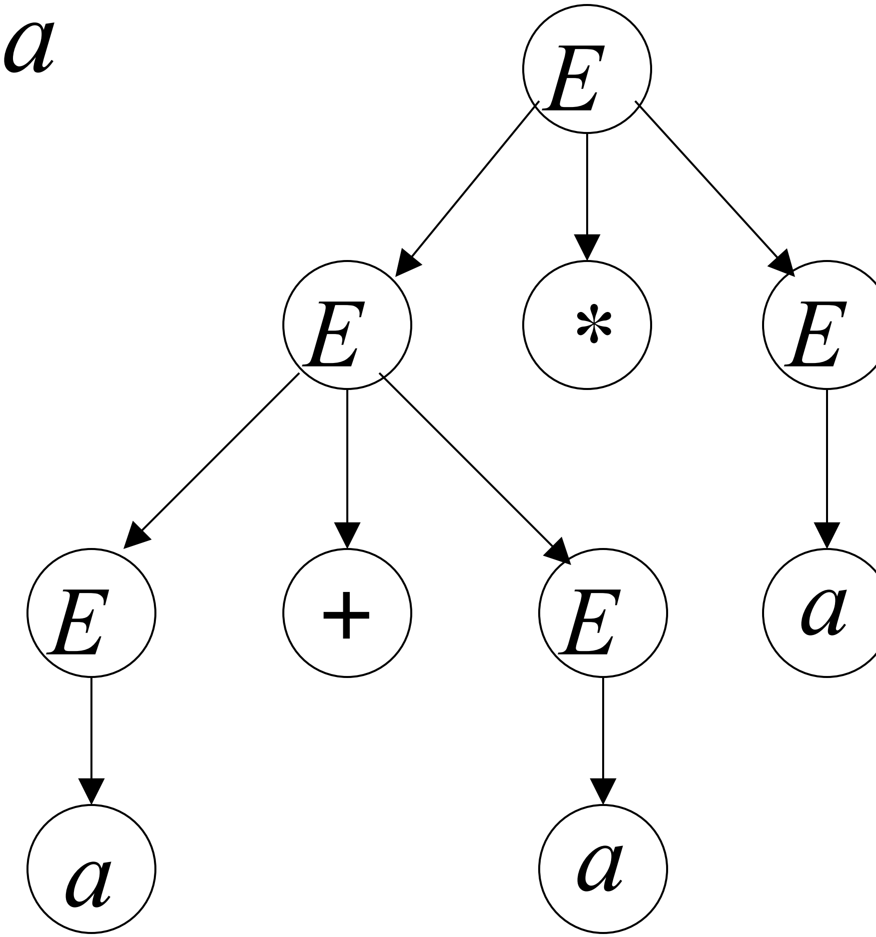


Another leftmost derivation for $a + a * a$

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

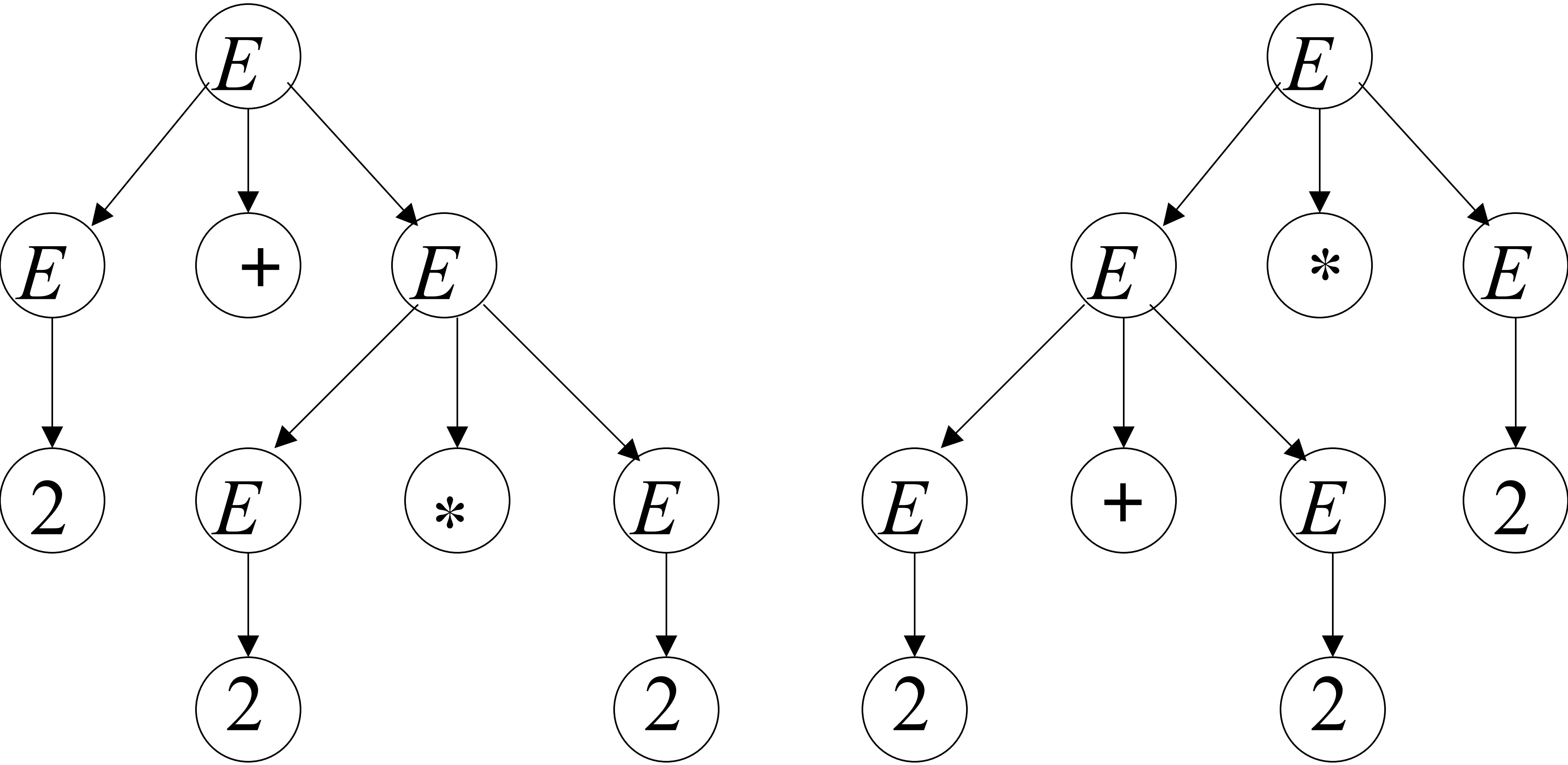
Two derivation trees for $a + a * a$





take $a = 2$

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



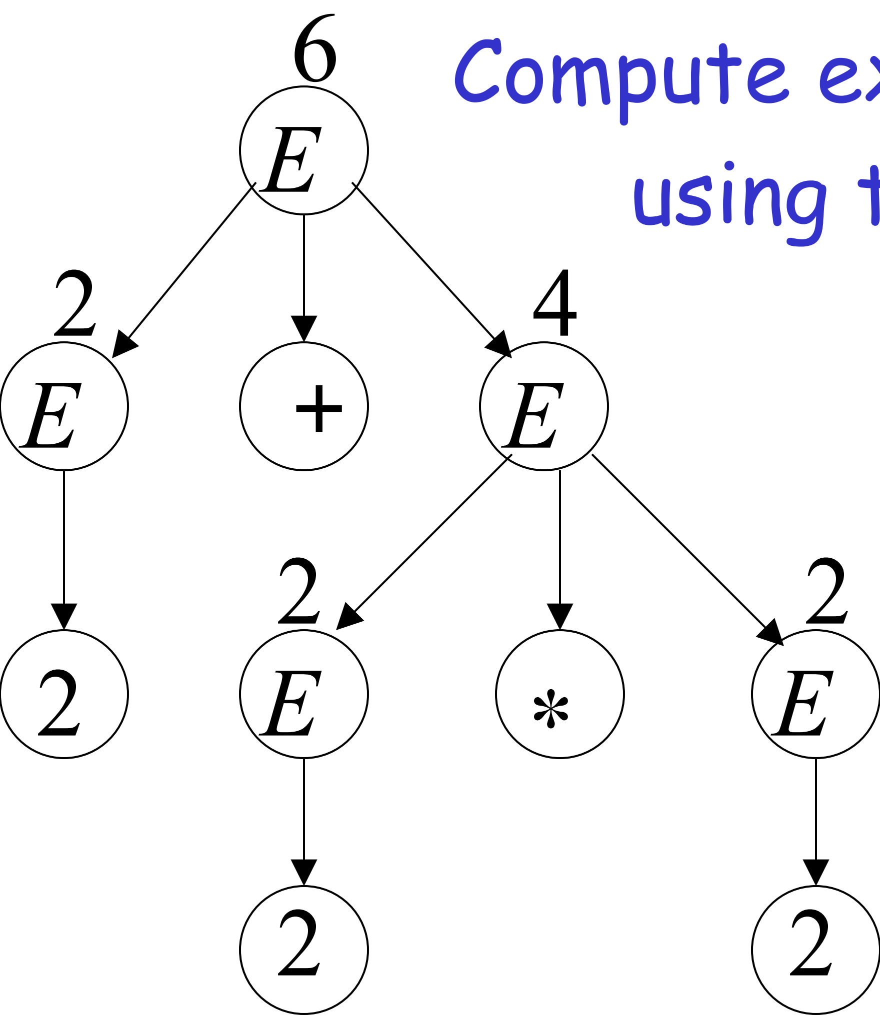
Good Tree

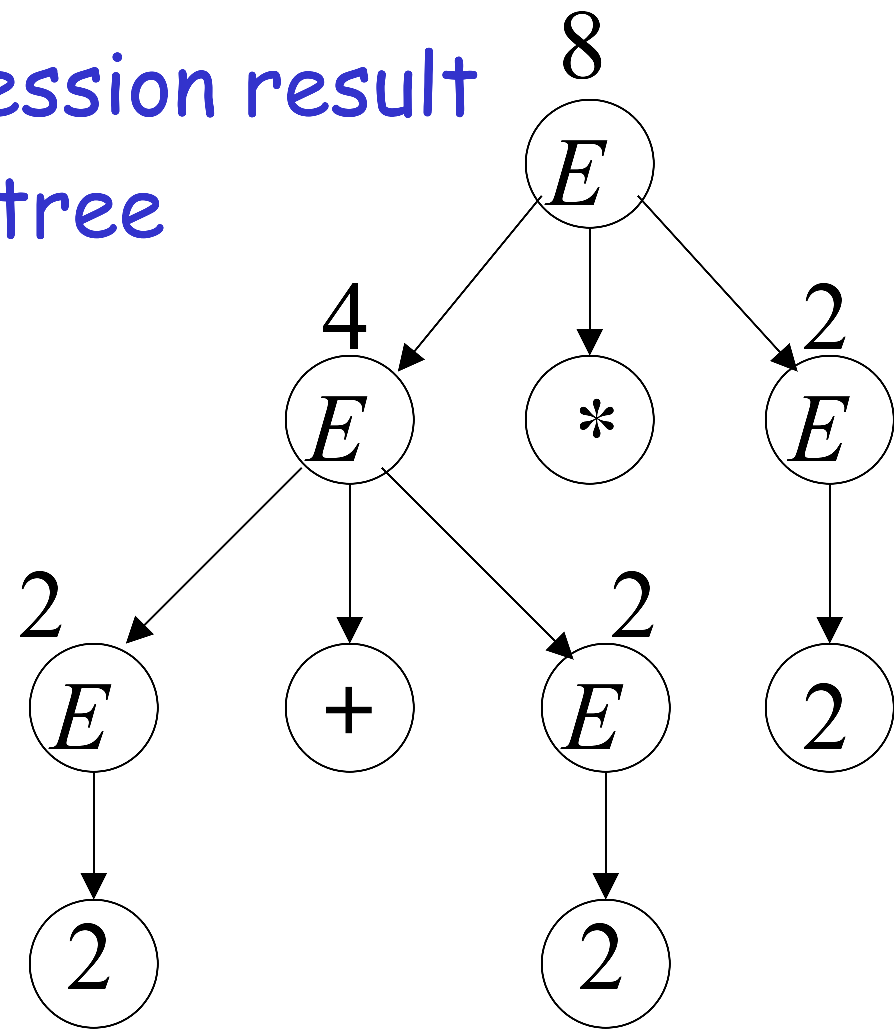
$$2+2*2=6$$

Bad Tree

$$2+2*2=8$$

Compute expression result using the tree





Two different derivation trees
may cause problems in applications which
use the derivation trees:

- Evaluating expressions

- In general, in compilers
  for programming languages

Ambiguous Grammar:

A context-free grammar G is ambiguous if there is a string $w \in L(G)$ which has:

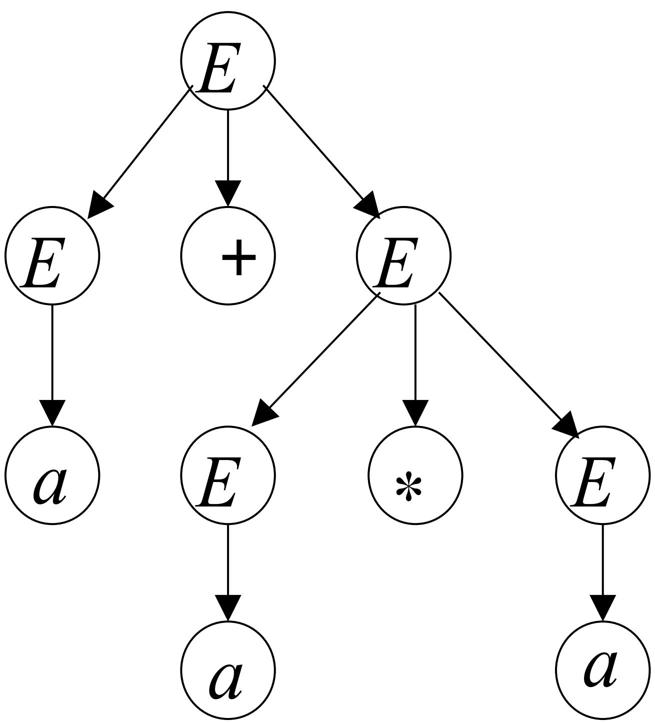
two different derivation trees
or
two leftmost derivations

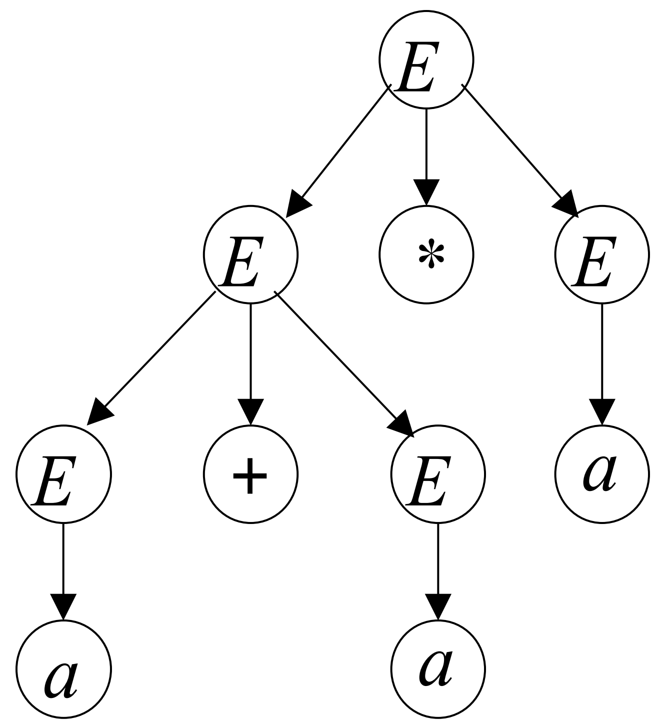
(Two different derivation trees give two different leftmost derivations and vice-versa)

Example: $\boxed{E \to E + E \mid E * E \mid (E) \mid a}$

this grammar is ambiguous since

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





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

this grammar is ambiguous also because

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

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

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

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

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

Another ambiguous grammar:

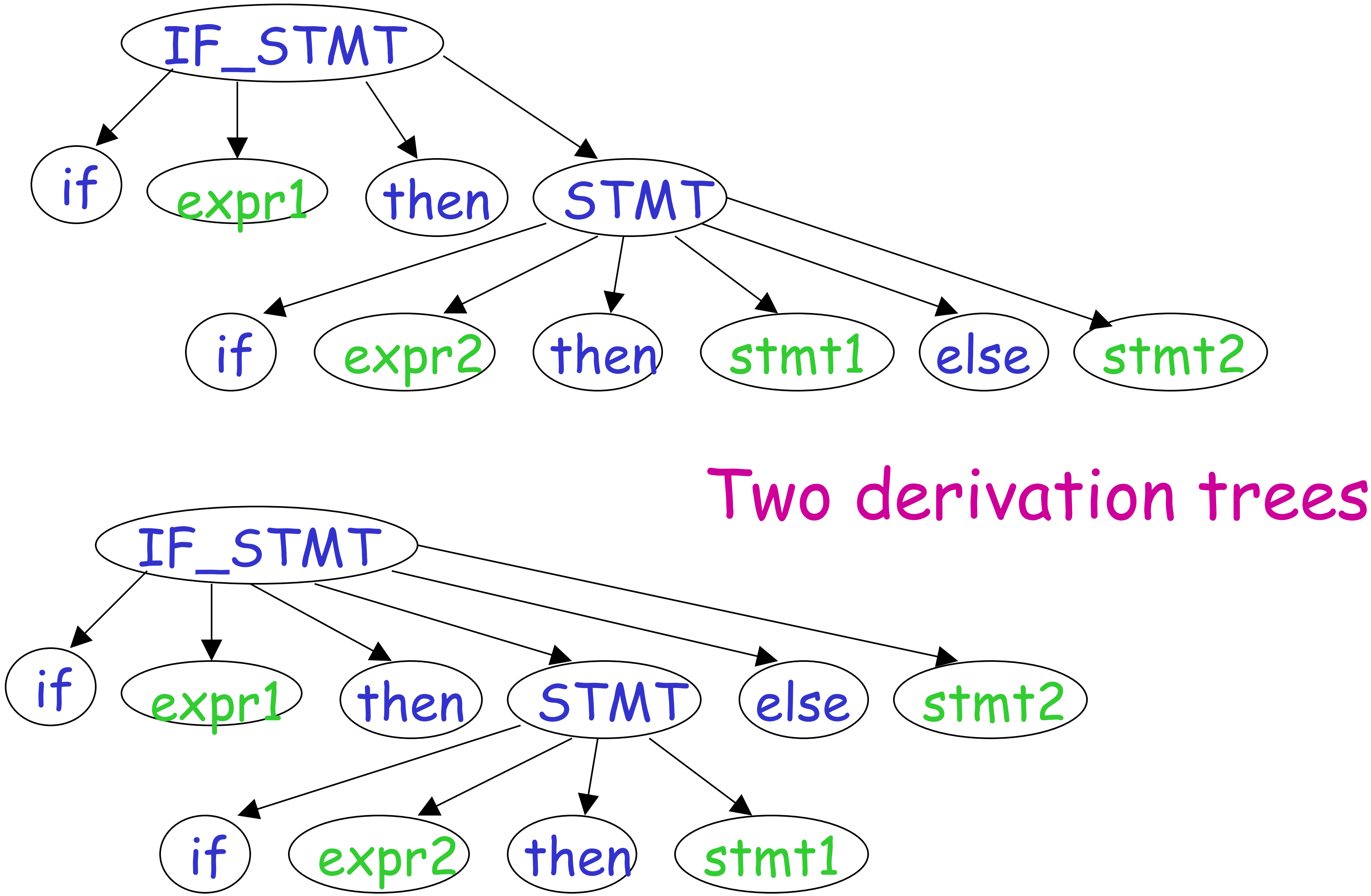
$$IF\_STMT \rightarrow \text{if EXPR then STMT}$$
$$\quad | \ \text{if EXPR then STMT else STMT}$$

Variables

Terminals

Very common piece of grammar in programming languages

If expr1 then if expr2 then stmt1 else stmt2



In general, ambiguity is bad
and we want to remove it

Sometimes it is possible to find
a non-ambiguous grammar for a language

But, in general we cannot do so

A successful example:

Ambiguous Grammar

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

Equivalent Non-Ambiguous Grammar

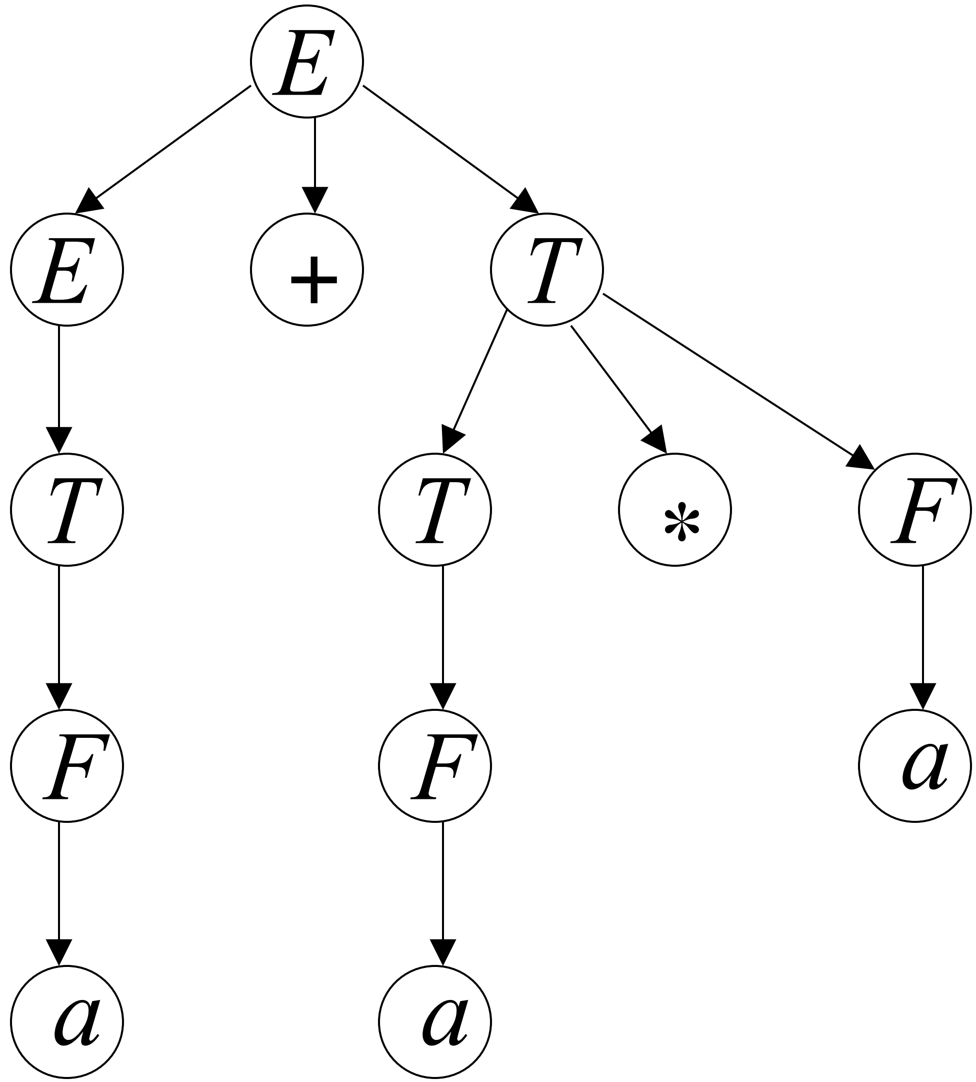
$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (E) \mid a$$

generates the same language

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

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

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



Unique derivation tree for $a+a*a$

An un-successful example:

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

$$n, m \geq 0$$

L is inherently ambiguous:

every grammar that generates this language is ambiguous

Example (ambiguous) grammar for L:

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

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

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

Dealing with Ambiguity

- There are several ways to handle ambiguity

- Most direct method is to rewrite grammar unambiguously

$$E \rightarrow E' + E \mid E'$$

$$E' \rightarrow id * E' \mid id \mid (E) * E' \mid (E)$$

- Enforces precedence of * over +

Ambiguity

No general techniques for handling ambiguity

Impossible to convert automatically an ambiguous grammar to an unambiguous one

Used with care, ambiguity can simplify the grammar. Sometimes allows more natural definitions. However, we still need to eventually eliminate ambiguity. How?

Precedence and Associativity Declarations

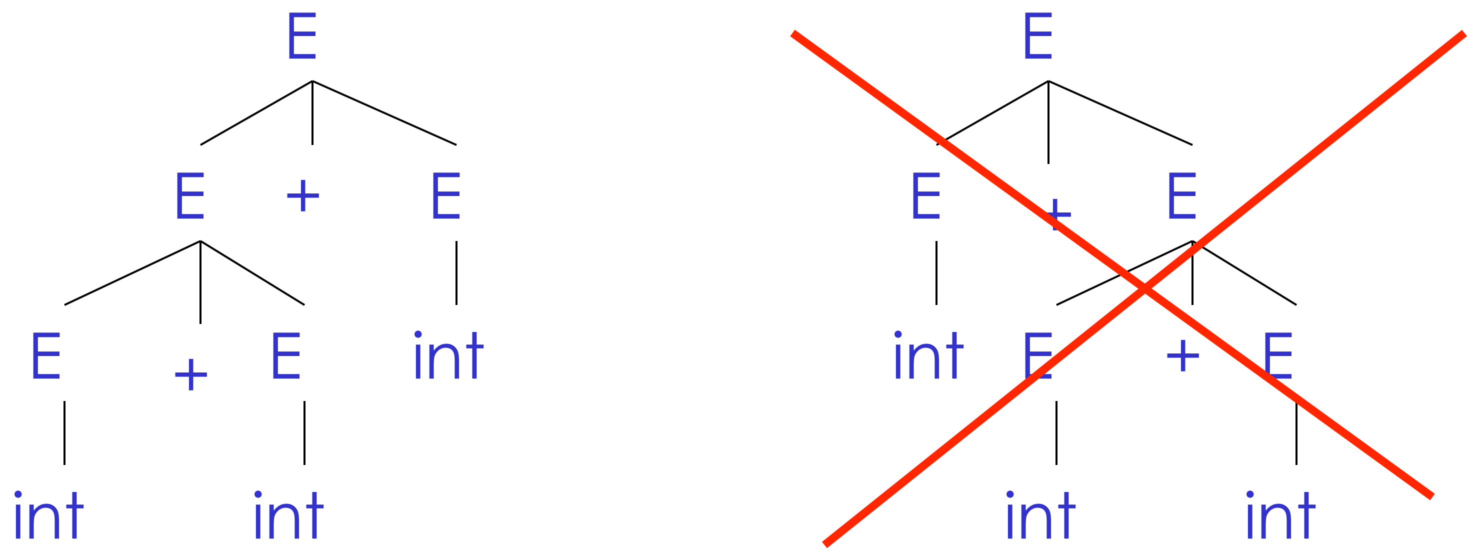
Instead of rewriting the grammar

- Use the more natural (ambiguous) grammar
- Along with disambiguating declarations

Most parser-generators allow <u>precedence and associativity declarations</u> to disambiguate grammars

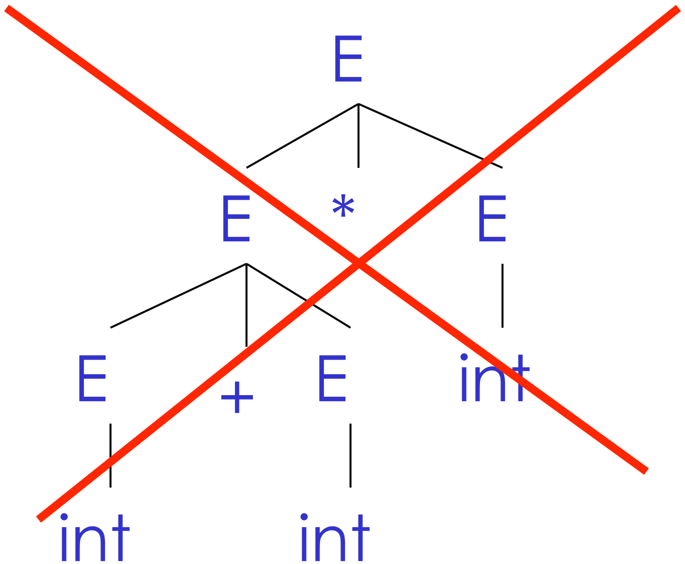
Associativity Declarations

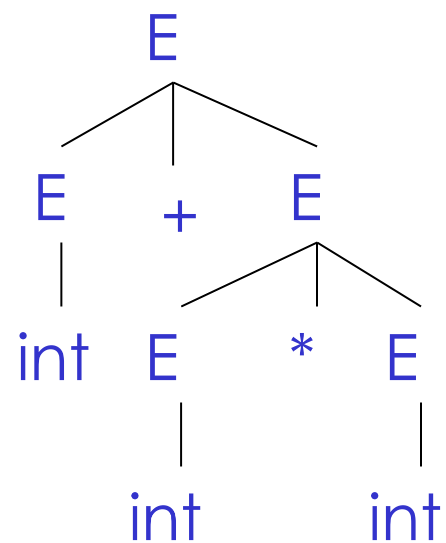
- Ambiguous
  - two parse trees of int + int + int
- Left associativity declaration: %left +



Precedence Declarations

- Consider the string int + int * int
- Precedence declarations: %left +
  %left *





Properties of Context-Free languages

Union

Context-free languages
are closed under: **Union**

L_1 is context free

L_2 is context free

$\Rightarrow$ $L_1 \cup L_2$ is context-free

Example

Language | **Grammar**

$$L_1 = \{a^n b^n\}$$

$$S_1 \rightarrow aS_1b \mid \lambda$$

$$L_2 = \{ww^R\}$$

$$S_2 \rightarrow aS_2a \mid bS_2b \mid \lambda$$

Union

$$L = \{a^n b^n\} \cup \{ww^R\}$$

$$S \rightarrow S_1 \mid S_2$$

In general:

For context-free languages $L_1, \ L_2$

with context-free grammars $G_1, \ G_2$

and start variables $S_1, \ S_2$

The grammar of the **union** $L_1 \cup L_2$

has new start variable S

and additional production $S \to S_1 \mid S_2$

Concatenation

Context-free languages are closed under: **Concatenation**

L_1 is context free

L_2 is context free

$\Rightarrow$ $L_1 L_2$ is context-free

Example

Language | **Grammar**

$$L_1 = \{a^n b^n\}$$

$$S_1 \to aS_1b \mid \lambda$$

$$L_2 = \{ww^R\}$$

$$S_2 \to aS_2a \mid bS_2b \mid \lambda$$

Concatenation

$$L = \{a^n b^n\}\{ww^R\}$$

$$S \to S_1 S_2$$

In general:

For context-free languages $L_1, \ L_2$

with context-free grammars $G_1, \ G_2$

and start variables $S_1, \ S_2$

The grammar of the **concatenation** $L_1 L_2$

has new start variable S

and additional production $S \to S_1 S_2$

Star Operation

Context-free languages
are closed under: **Star-operation**

L is context free $\Rightarrow$ L^* is context-free

Example

Language

Grammar

$$L = \{a^n b^n\}$$

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

<u>**Star Operation**</u>

$$L = \{a^n b^n\}^*$$

$$S_1 \to SS_1 \mid \lambda$$

In general:

For context-free language L
with context-free grammar G
and start variable S

The grammar of the **star operation** L^*
has new start variable S_1
and additional production $S_1 \to SS_1 \mid \lambda$

Context-free Languages don't have the following Properties

Intersection

Context-free languages are **not** closed under: **intersection**

L_1 is context free

L_2 is context free

$\Rightarrow$ $L_1 \cap L_2$ **not** necessarily context-free

Example

$$L_1 = \{a^n b^n c^m\}$$

Context-free:

$$S \to AC$$

$$A \to aAb \mid \lambda$$

$$C \to cC \mid \lambda$$

$$L_2 = \{a^n b^m c^m\}$$

Context-free:

$$S \to AB$$

$$A \to aA \mid \lambda$$

$$B \to bBc \mid \lambda$$

Intersection

$L_1 \cap L_2 = \{a^n b^n c^n\}$ NOT context-free

Complement

Context-free languages
are **<u>not</u>** closed under: **complement**

L is context free $\Longrightarrow$ $\overline{L}$ <u>**not**</u> necessarily context-free.

I.e., the complement of a context-free language may or may not be context-free.

Example

$$L_1 = \{a^n b^n c^m\}$$

$$L_2 = \{a^n b^m c^m\}$$

Context-free:

$$S \to AC$$

$$A \to aAb \mid \lambda$$

$$C \to cC \mid \lambda$$

Context-free:

$$S \to AB$$

$$A \to aA \mid \lambda$$

$$B \to bBc \mid \lambda$$

Complement

$$\overline{\overline{L_1} \cup \overline{L_2}} = L_1 \cap L_2 = \{a^n b^n c^n\}$$

NOT context-free

Intersection of Context-free languages and Regular Languages

The intersection of

a context-free language and

a regular language

is a context-free language

L_1 context free

L_2 regular

$\Rightarrow$ $L_1 \cap L_2$ context-free

Applications of Regular Closure

The intersection of

a context-free language and

a regular language

is a context-free language

L_1 context free

L_2 regular

$\Rightarrow$

Regular Closure

$L_1 \cap L_2$

context-free

An Application of Regular Closure

Prove that: $L = \{a^n b^n : \quad n \neq 100, n \geq 0\}$

is context-free

We know:

$\{a^n b^n : n \geq 0\}$ is context-free

We also know:

$$L_1 = \{a^{100}b^{100}\}$$ is regular

$$\Downarrow$$

$$\overline{L_1} = \{(a+b)^*\} - \{a^{100}b^{100}\}$$ is regular

$\{a^n b^n\}$ $\overline{L_1} = \{(a+b)^*\} - \{a^{100}b^{100}\}$

context-free regular

(regular closure) $\{a^n b^n\} \cap \overline{L_1}$ context-free

$$\{a^n b^n\} \cap \overline{L_1} = \{a^n b^n : \quad n \neq 100, n \geq 0\} = L$$

is context-free

Another Application of Regular Closure

Prove that: $L = \{w: \ n_a = n_b = n_c\}$

is **not** context-free

If $L = \{w: \; n_a = n_b = n_c\}$ is context-free

(regular closure)

Then $L \cap \{a^* b^* c^*\} = \{a^n b^n c^n\}$

context-free regular context-free

Impossible!!!

Therefore, L is **not** context free