

Context-free Languages and Context-free Grammars

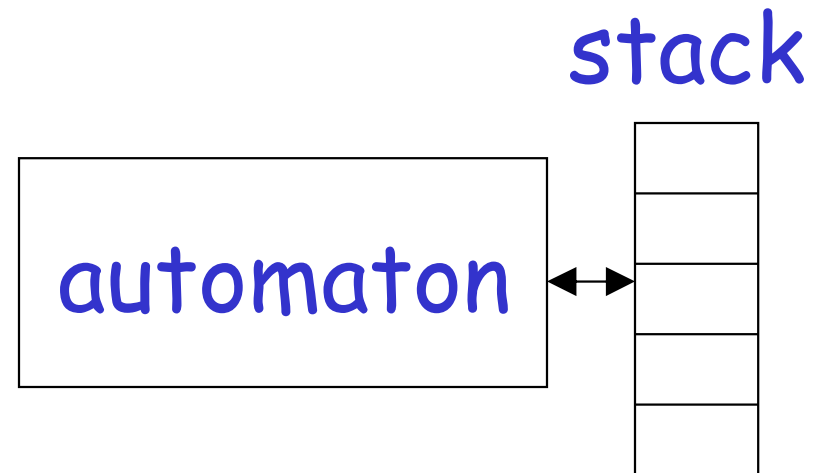
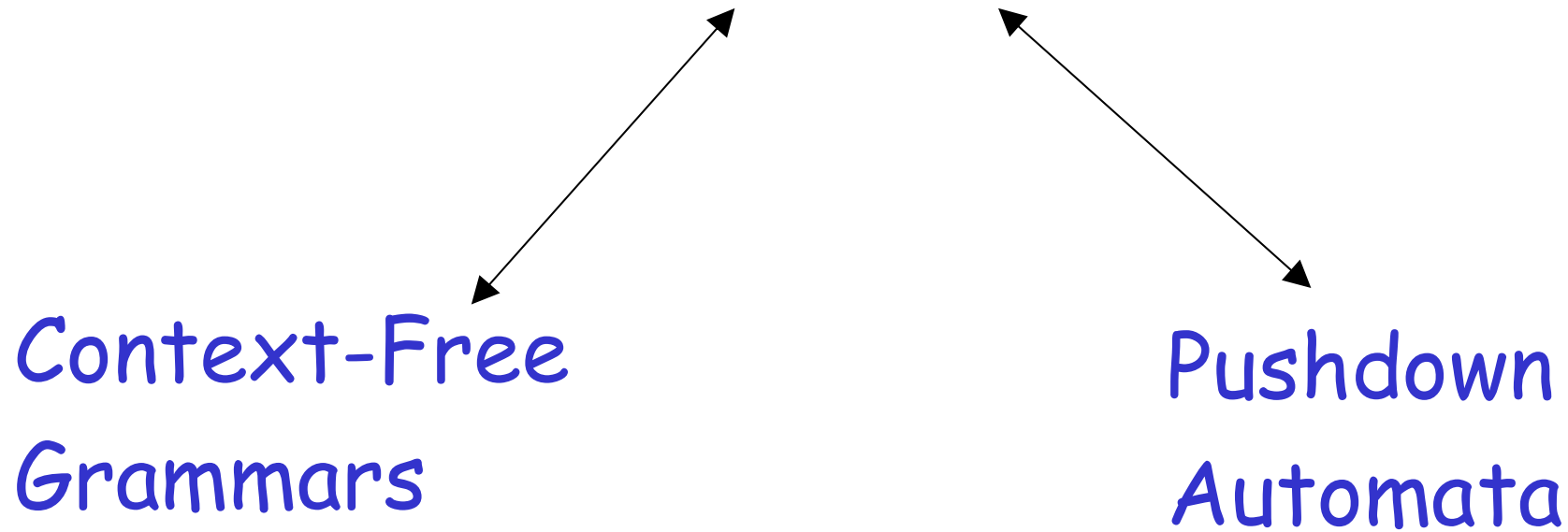
Context-Free Languages

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

Regular Languages

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

Context-Free Languages

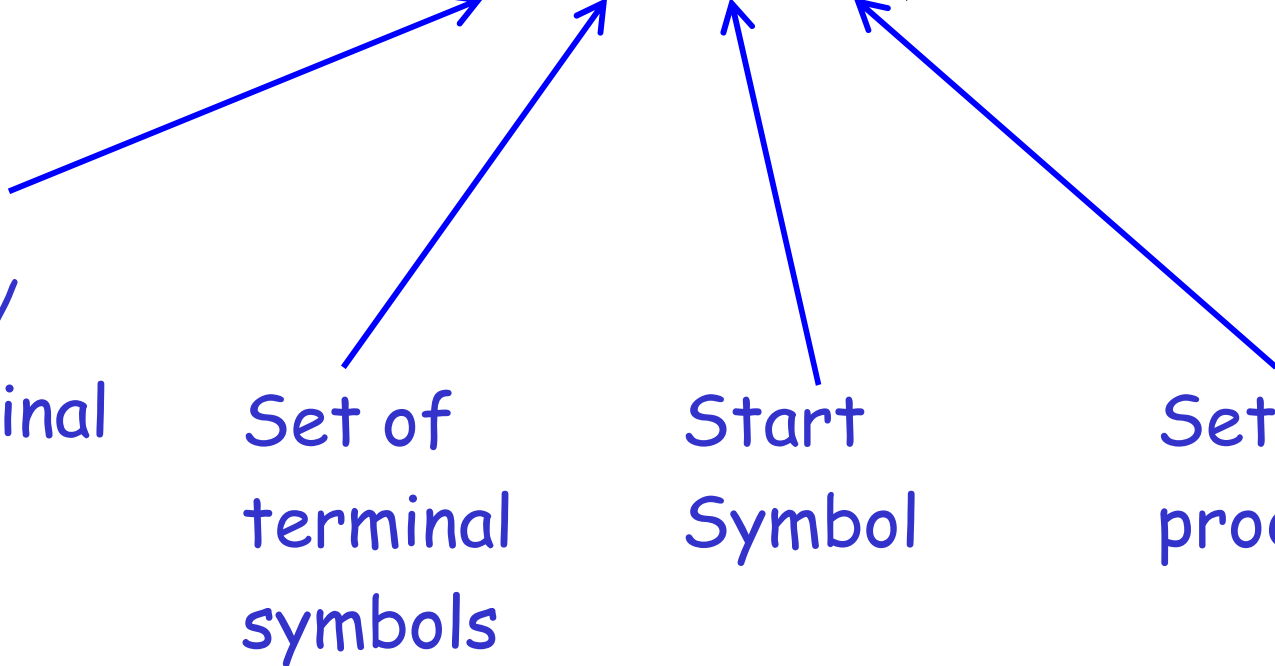


Context-Free Grammars

Formal Definitions

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

Set of
Variables/
Non-terminal



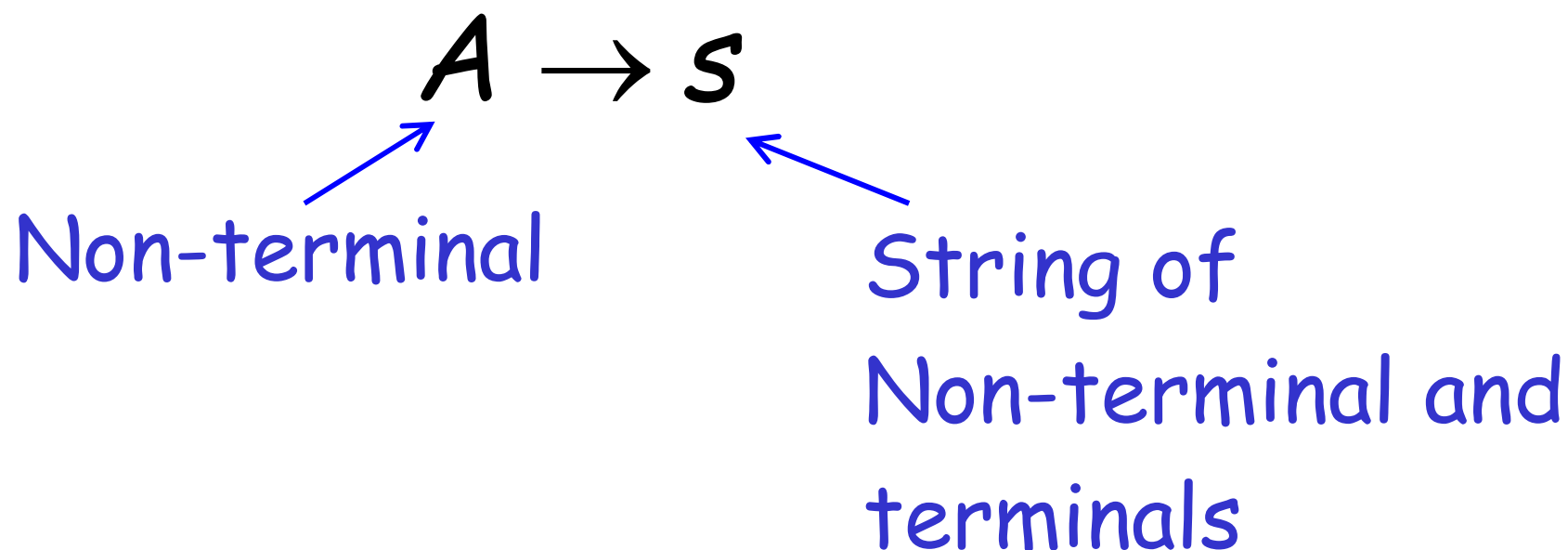
Set of
terminal
symbols

Start
Symbol

Set of
productions

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

All productions in P are of the form



Example of Context-Free Grammar

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

productions

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

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

$V = \{S\}$
variables

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

start variable

Language of a Grammar:

For a grammar G with start variable S

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

String of terminals or λ

Example

Sequence of
terminals and non-terminals

Grammar:

$$S \rightarrow \overbrace{aSb}$$

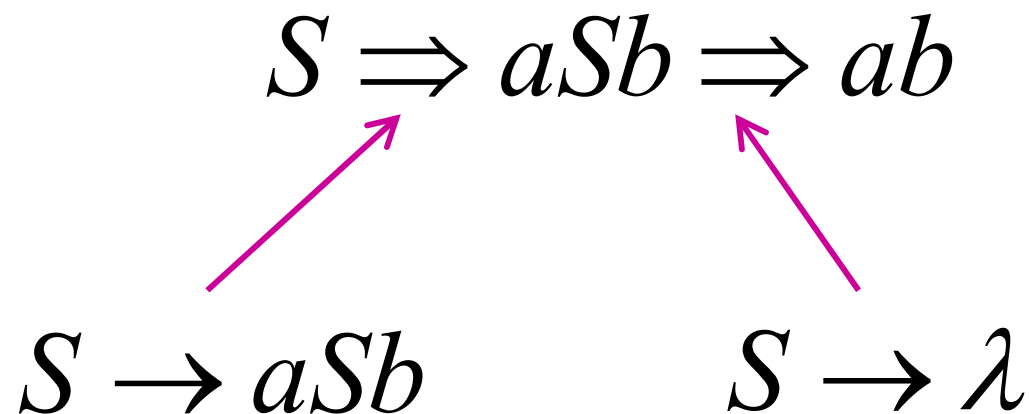
$$S \rightarrow \lambda$$

Non-terminals The right side
may be λ

Grammar: $S \rightarrow aSb$

$S \rightarrow \lambda$

Derivation of string ab :

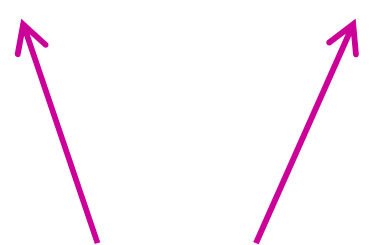


Grammar: $S \rightarrow aSb$

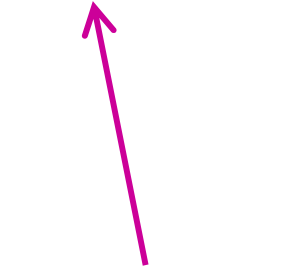
$S \rightarrow \lambda$

Derivation of string $aabb$:

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



$S \rightarrow aSb$



$S \rightarrow \lambda$

Grammar: $S \rightarrow aSb$

$$S \rightarrow \lambda$$

Language of the grammar:

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

A Convenient Notation

We write: $S \xRightarrow{*} aaabbb$

for zero or more derivation steps

Instead of:

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaasbbb \Rightarrow aaabbbb$

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

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

in zero or more derivation steps

Trivially: $w \xRightarrow{*} w$

Context-Free Language:

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

Example:

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

is a context-free language

since context-free grammar G :

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

generates $L(G) = L$

Another Example

Context-free grammar G :

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

Example derivations:

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

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

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

Palindromes of even length

Another Example

Context-free grammar G :

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

Example derivations:

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

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

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

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

in any prefix v \}

Describes
matched

parentheses:

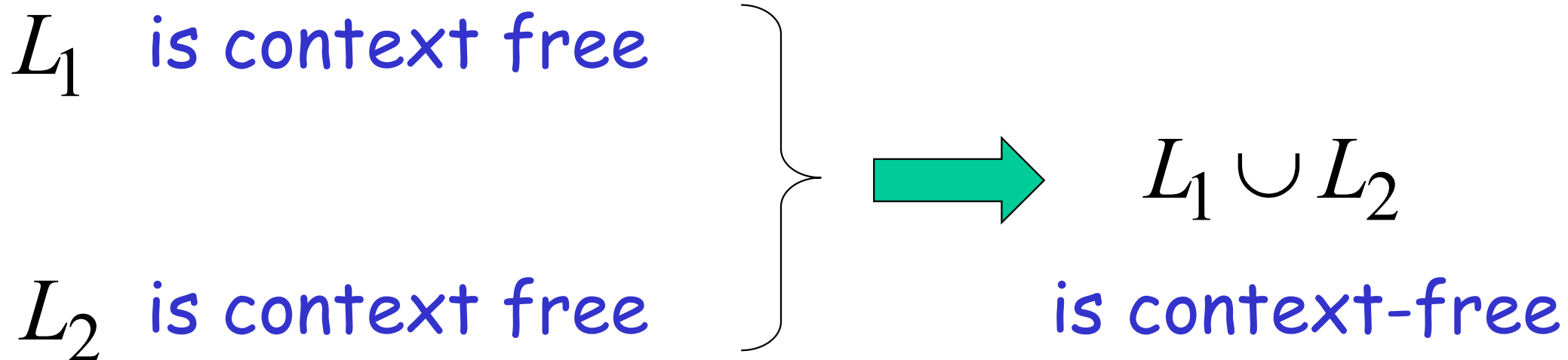
$() ((())) (())$

$a = (, \quad b =)$

Properties of Context-Free languages

Union

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



Example

Language

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

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

Grammar

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

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

Union

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

$$S \rightarrow S_1 \mid S_2$$

In general:

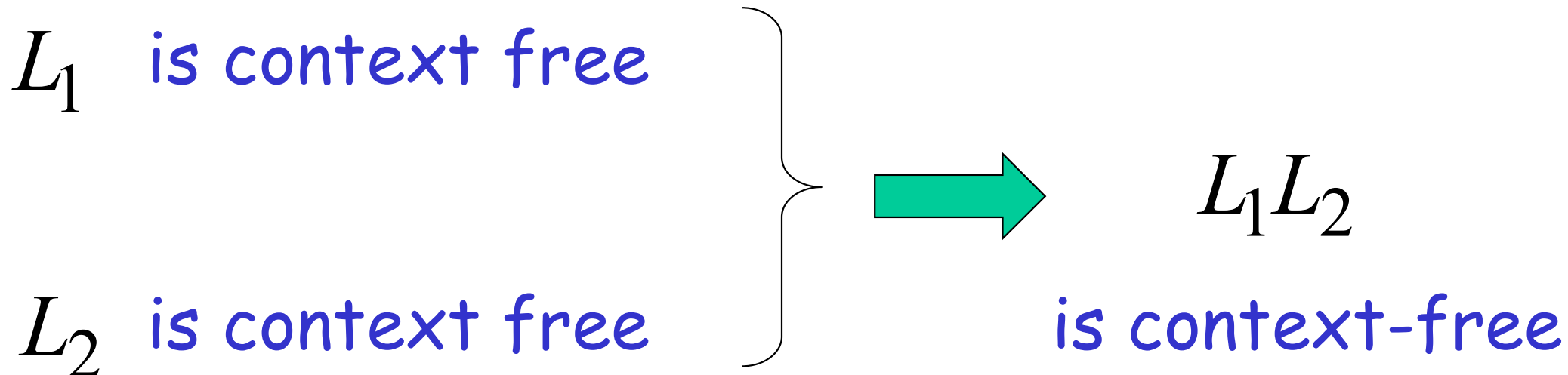
For context-free languages	L_1, L_2
with context-free grammars	G_1, G_2
and start variables	S_1, S_2

The grammar of the union	$L_1 \cup L_2$
has new start variable	S
and additional production	$S \rightarrow S_1 \mid S_2$

Concatenation

Context-free languages
are closed under:

Concatenation



Example

Language

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

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

Grammar

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

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

Concatenation

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

$$S \rightarrow S_1 S_2$$

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_1L_2
has new start variable	S
and additional production	$S \rightarrow S_1S_2$

Star Operation

Context-free languages
are closed under:

Star-operation

L is context free  L^* is context-free

Example

Language

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

Grammar

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

Star Operation

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

$$S_1 \rightarrow SS_1 \mid \lambda$$

In general:

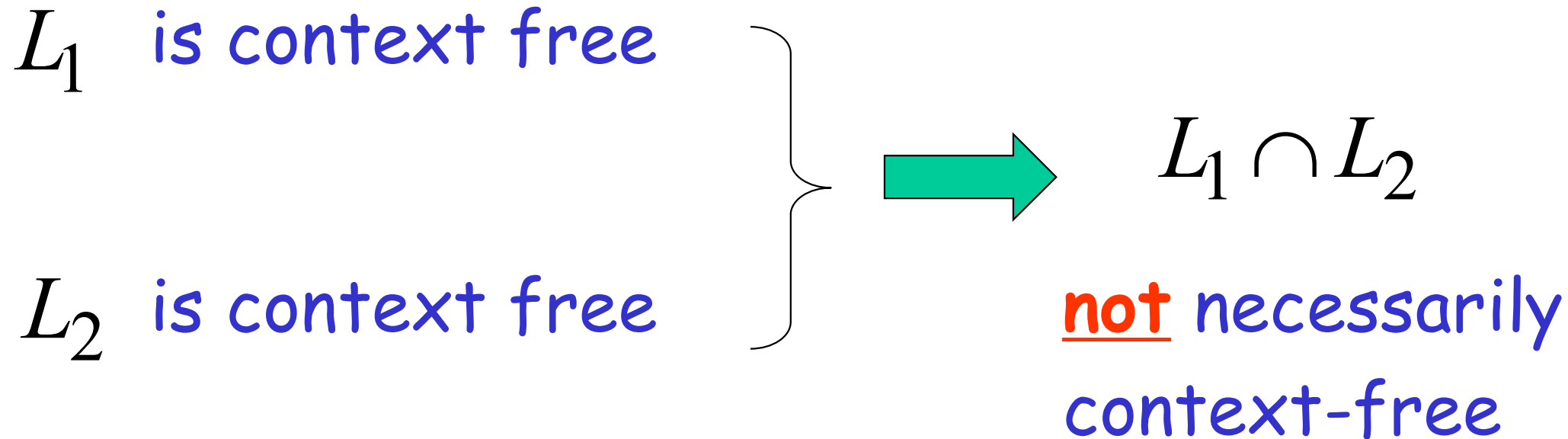
For context-free language L
with context-free grammar G
and start variable S

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

Context-free Languages
don't have the following
Properties

Intersection

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



Example

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

Context-free:

$$S \rightarrow AC$$

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

$$C \rightarrow cC \mid \lambda$$

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

Context-free:

$$S \rightarrow AB$$

$$A \rightarrow aA \mid \lambda$$


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

Intersection

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

Complement

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

L is context free  \overline{L} **not** 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:

$$S \rightarrow AC$$

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

$$C \rightarrow cC \mid \lambda$$

Context-free:

$$S \rightarrow AB$$

$$A \rightarrow aA \mid \lambda$$

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

Complement

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

NOT context-free

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 :

$$\begin{array}{ccccccccc} & 1 & & 2 & & 3 & & 4 & & 5 \\ S & \Rightarrow & AB & \Rightarrow & aaAB & \Rightarrow & aaB & \Rightarrow & aaBb & \Rightarrow & aab \end{array}$$

At each step, we substitute the
leftmost variable

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

Rightmost derivation order of string aab :

$$\begin{array}{ccccccccc}
 & 1 & & 4 & & 5 & & 2 & & 3 \\
 S & \Rightarrow & AB & \Rightarrow & ABb & \Rightarrow & Ab & \Rightarrow & aaAb & \Rightarrow & aab
 \end{array}$$

At each step, we substitute the
rightmost variable

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

Leftmost derivation of aab :

$$\begin{array}{ccccccccc}
 1 & & 2 & & 3 & & 4 & & 5 \\
 S & \Rightarrow & AB & \Rightarrow & aaAB & \Rightarrow & aaB & \Rightarrow & aaBb & \Rightarrow & aab
 \end{array}$$

Rightmost derivation of aab :

$$\begin{array}{ccccccccc}
 1 & & 4 & & 5 & & 2 & & 3 \\
 S & \Rightarrow & AB & \Rightarrow & ABb & \Rightarrow & Ab & \Rightarrow & aaAb & \Rightarrow & aab
 \end{array}$$

Derivation Trees

Consider the same example grammar:

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

And a derivation of aab :

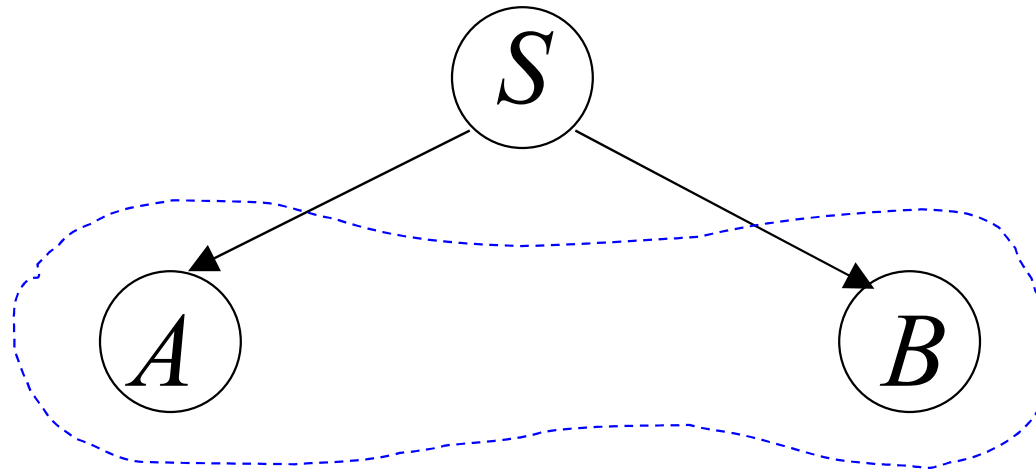
$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb \Rightarrow aaBb \Rightarrow aab$$

$$S \rightarrow AB$$

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

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

$$S \Rightarrow AB$$



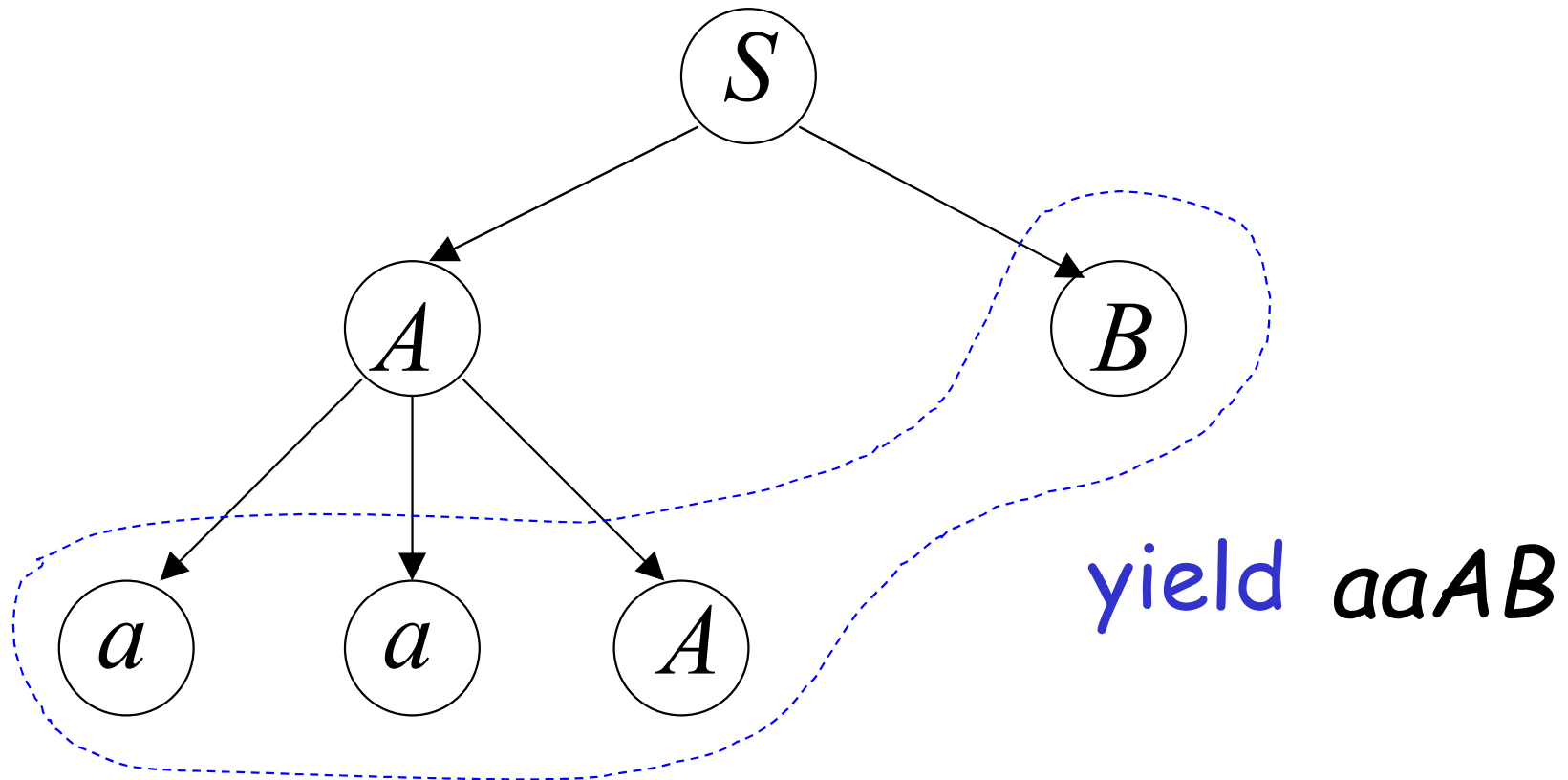
yield AB

$$S \rightarrow AB$$

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

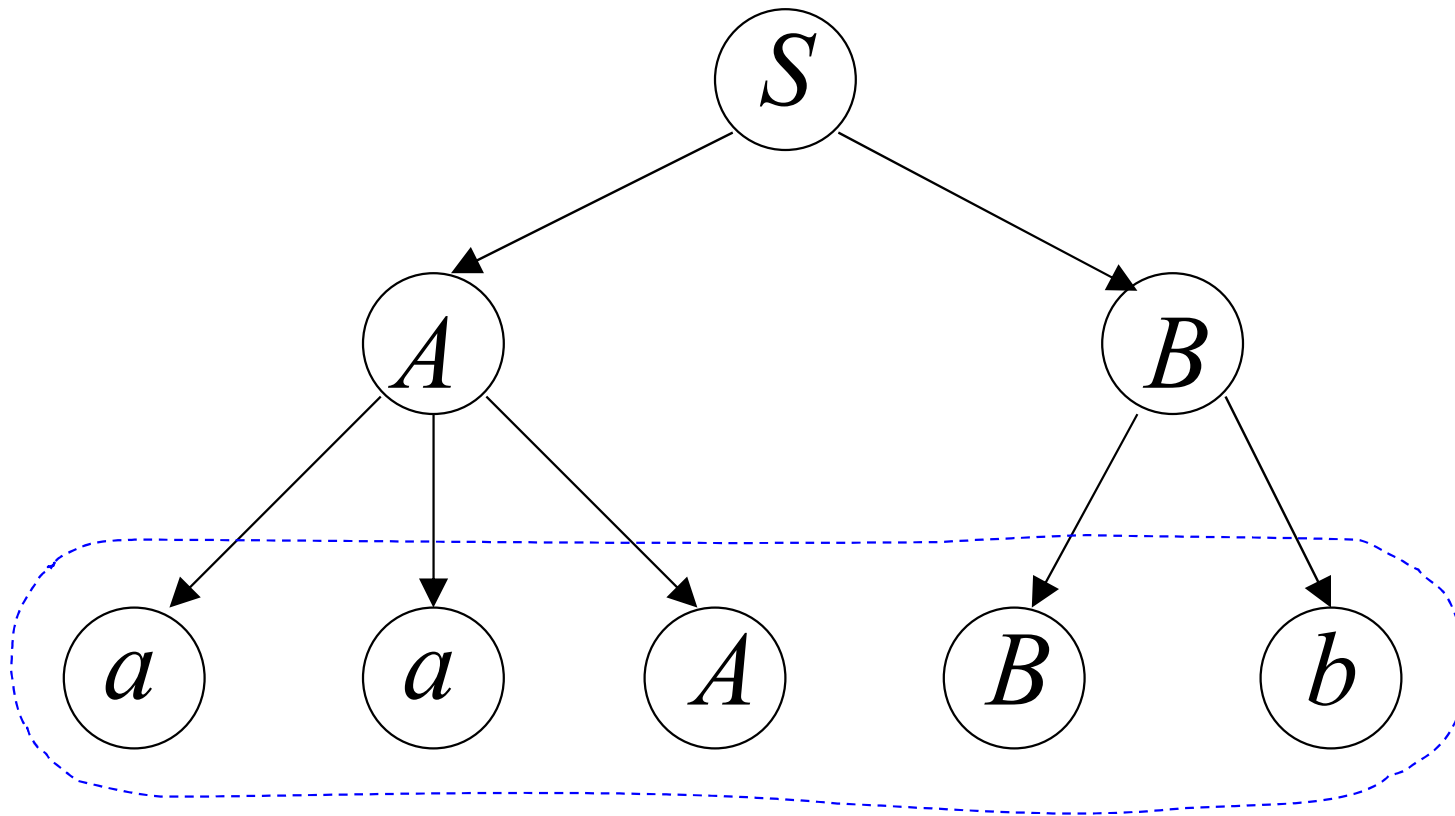
$$B \rightarrow Bb \mid \lambda$$

$$S \Rightarrow AB \Rightarrow aaAB$$



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

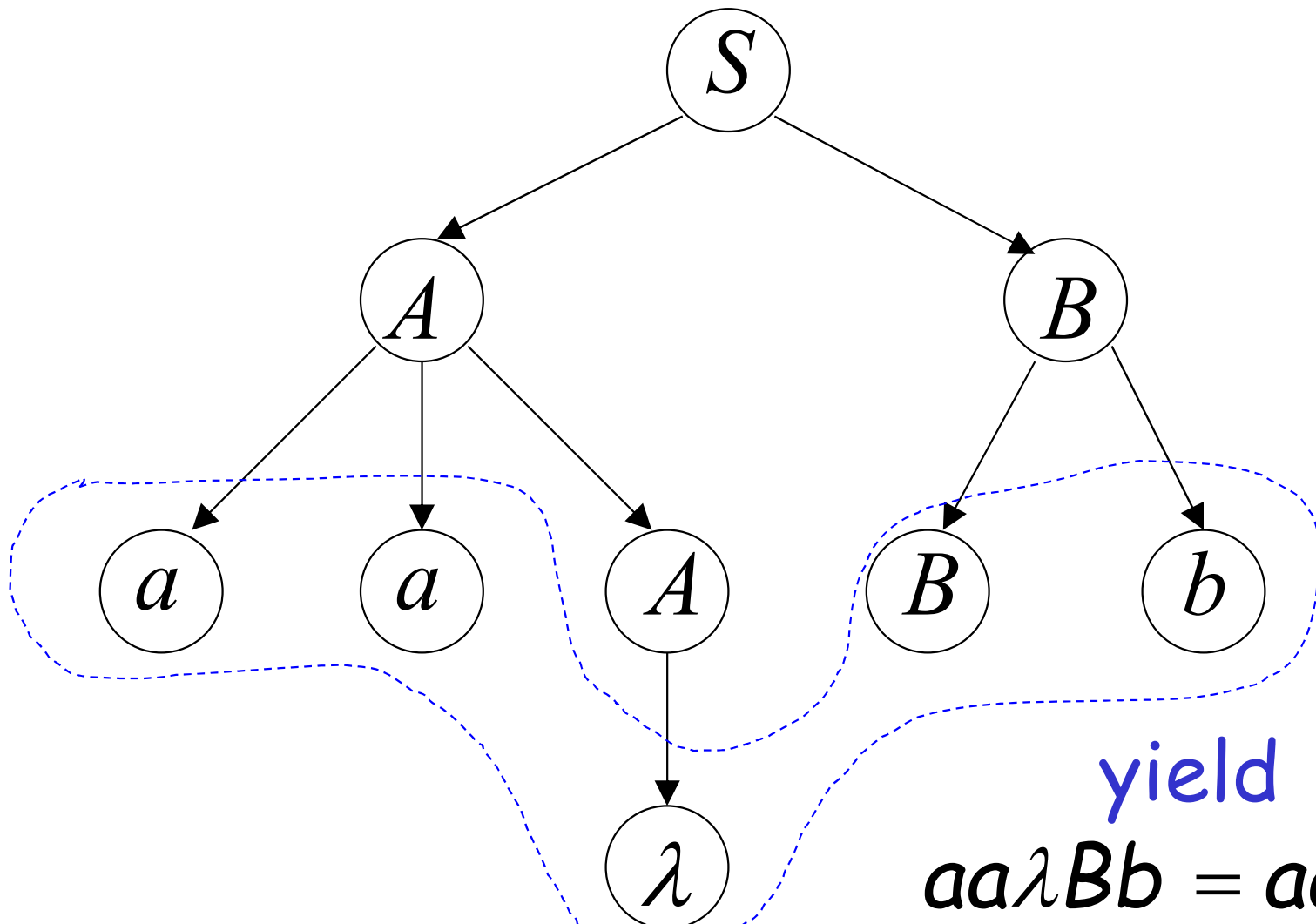
$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb$$



yield $aaABb$

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

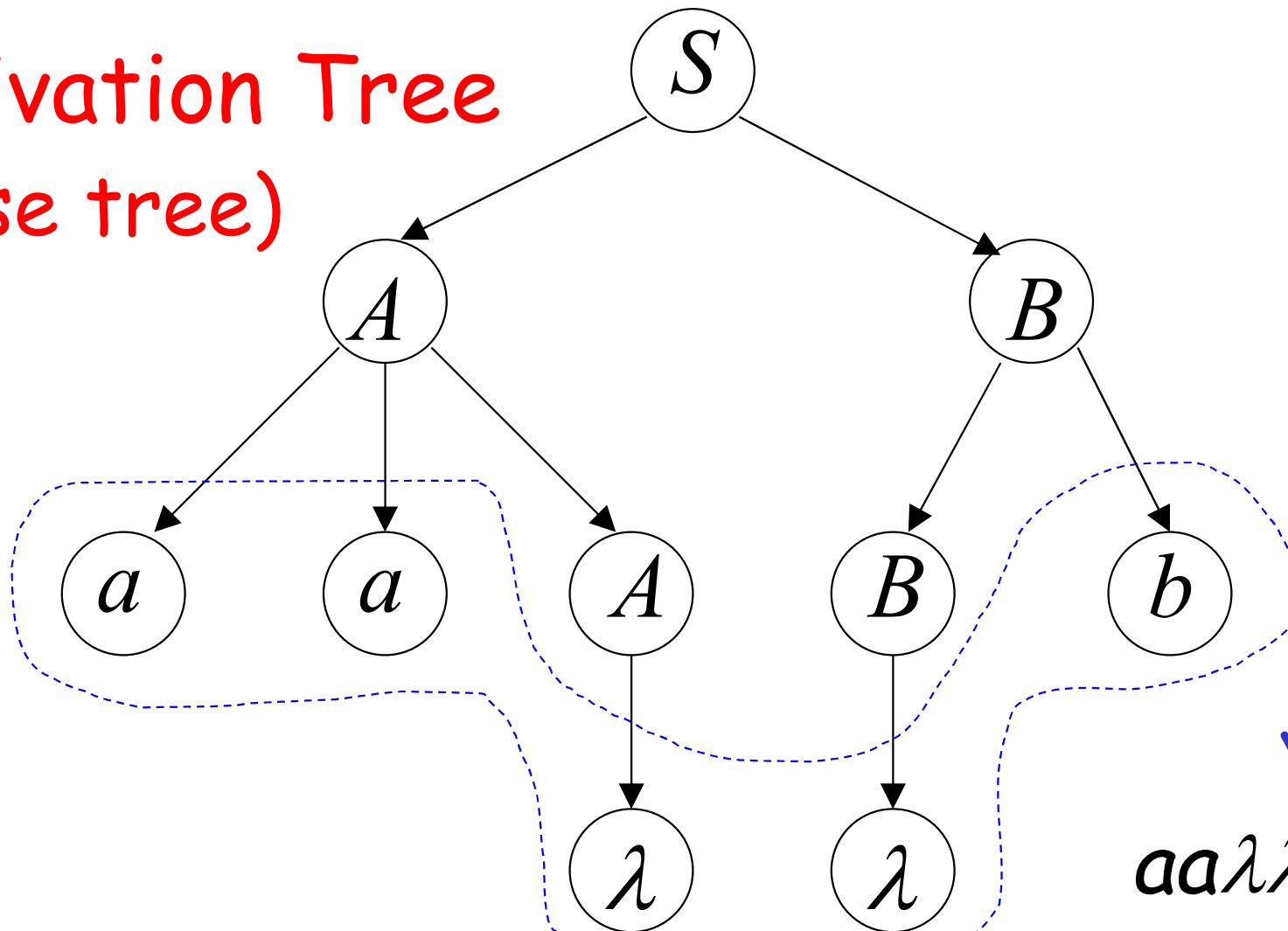
$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb \Rightarrow aaBb$$



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

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

Derivation Tree
(parse tree)



yield

$$aa\lambda\lambda b = aab$$

Sometimes, derivation order doesn't matter

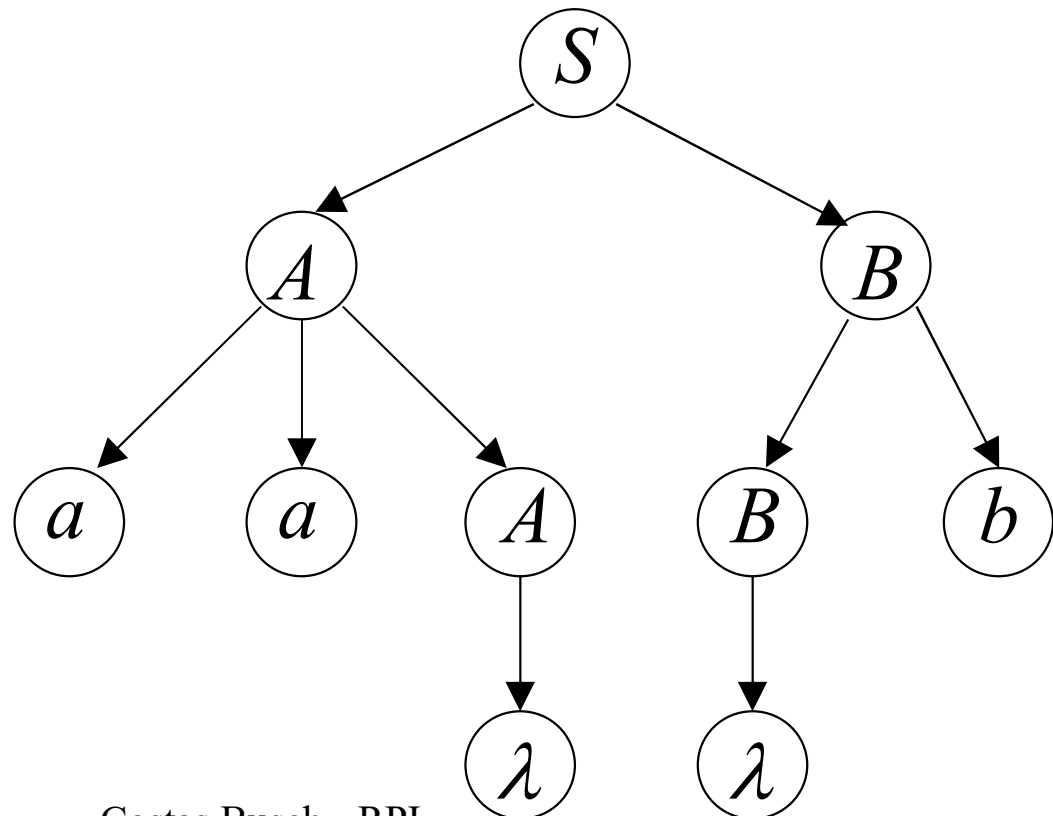
Leftmost derivation:

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

Rightmost derivation:

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

Give same
derivation tree



Ambiguity

Grammar for mathematical expressions

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

Example strings:

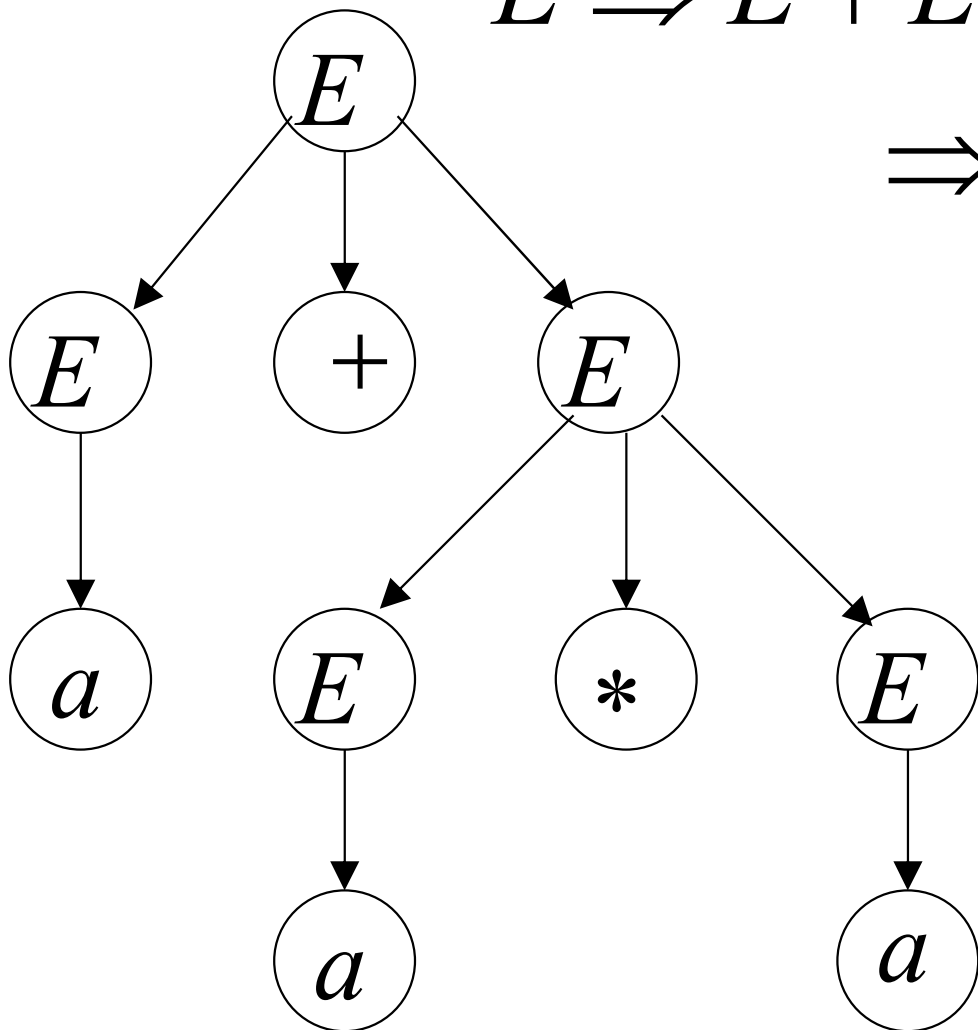
$$(a + a) * a + (a + a * (a + a))$$



Denotes any number

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

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

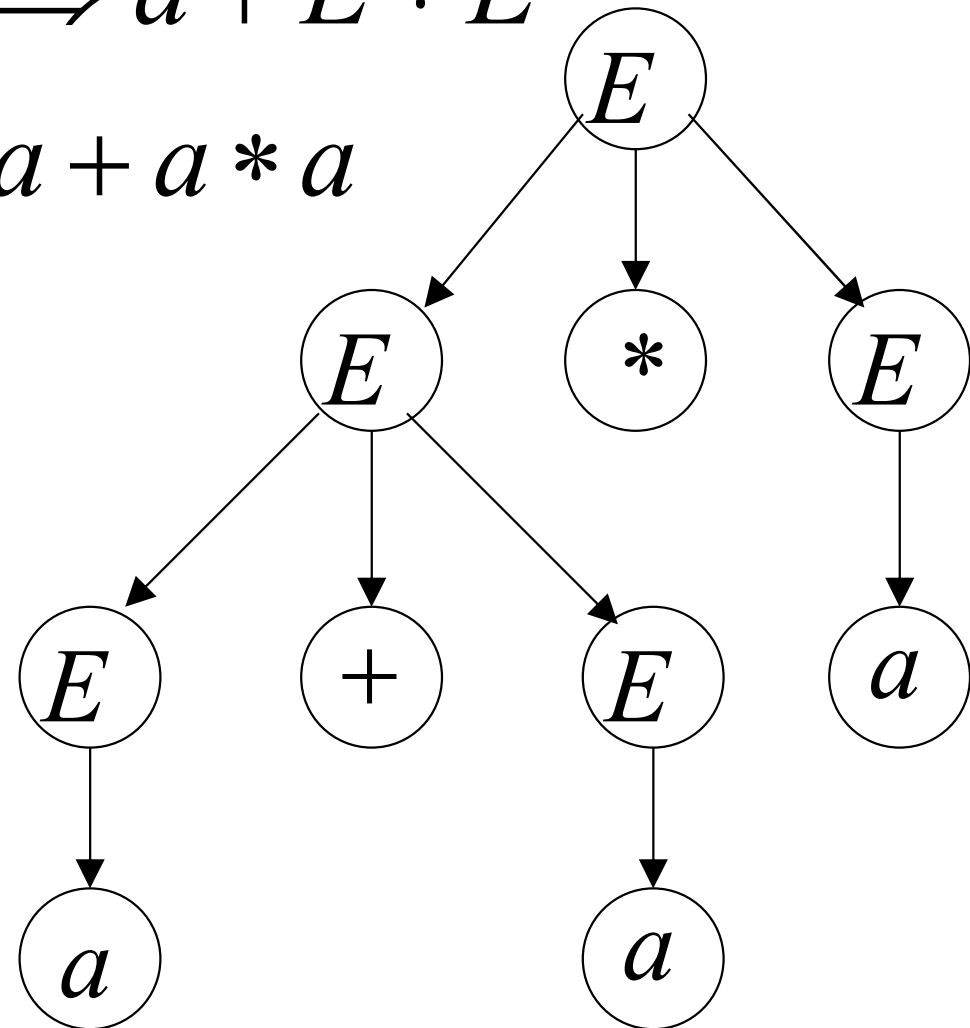


A leftmost derivation
for $a + a * a$

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

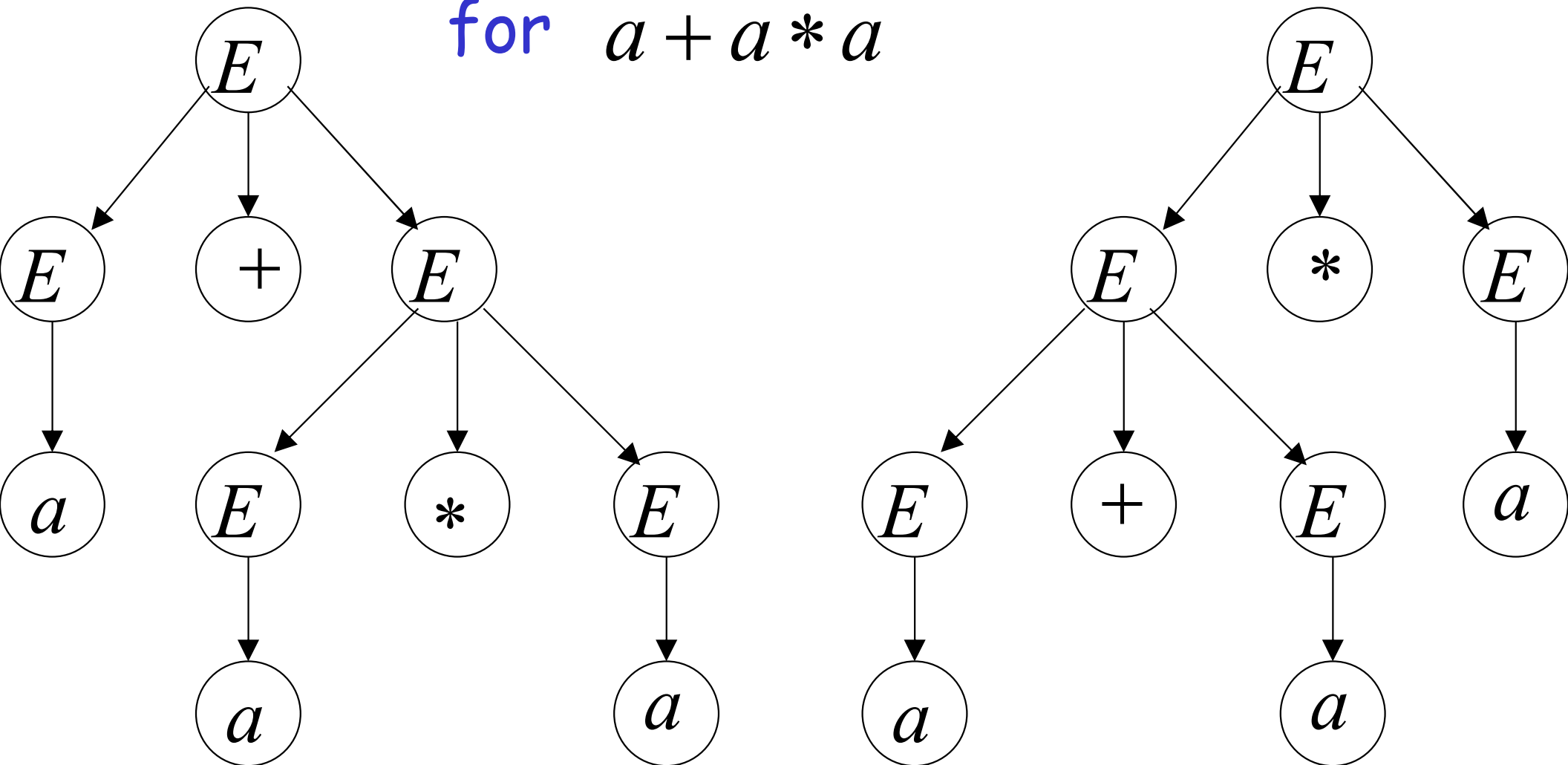
$$\begin{aligned} E &\Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E \\ &\Rightarrow a + a * E \Rightarrow a + a * a \end{aligned}$$

Another
leftmost derivation
for $a + a * a$



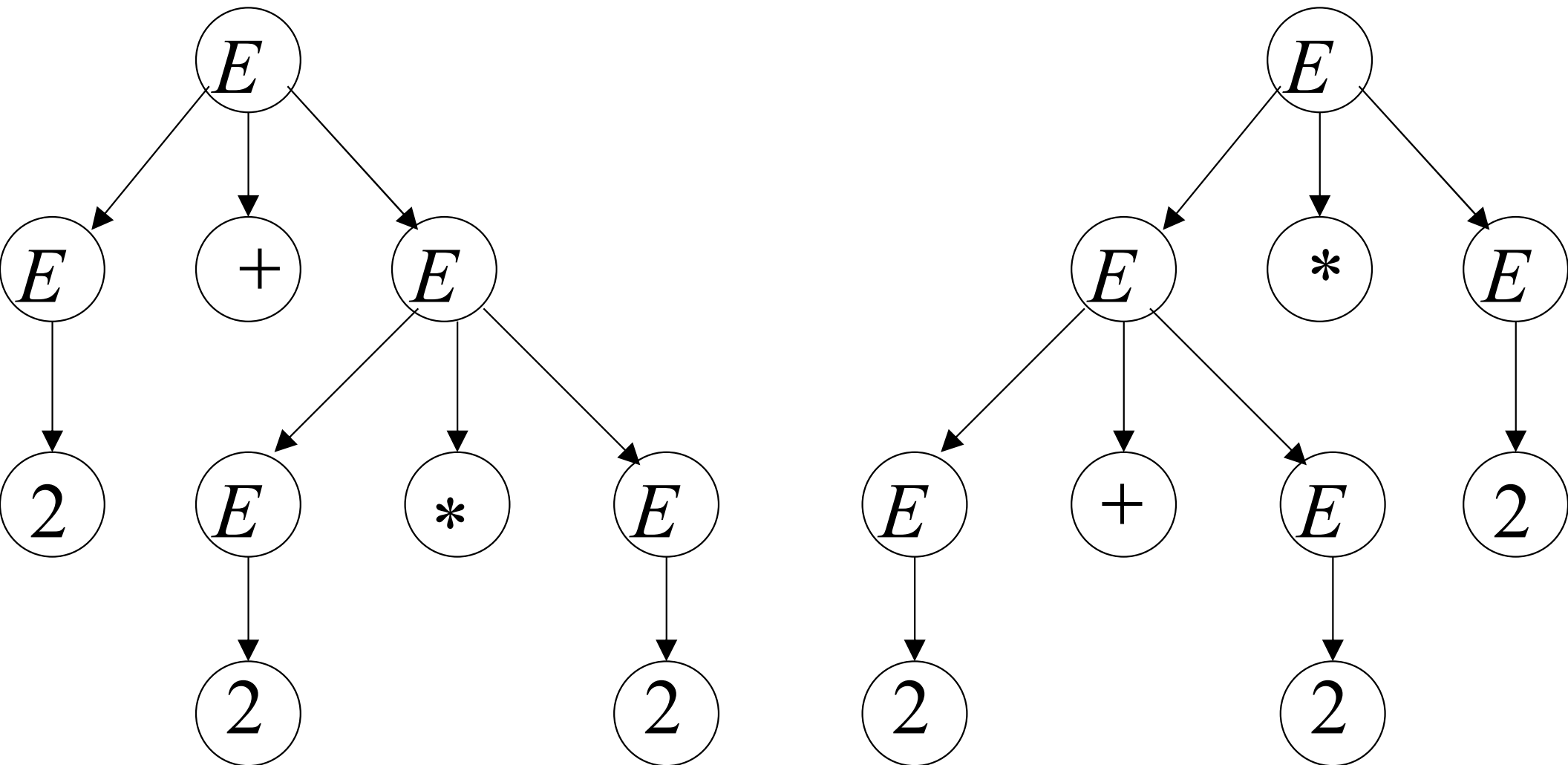
$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

Two derivation trees
for $a + a * a$



take $a = 2$

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



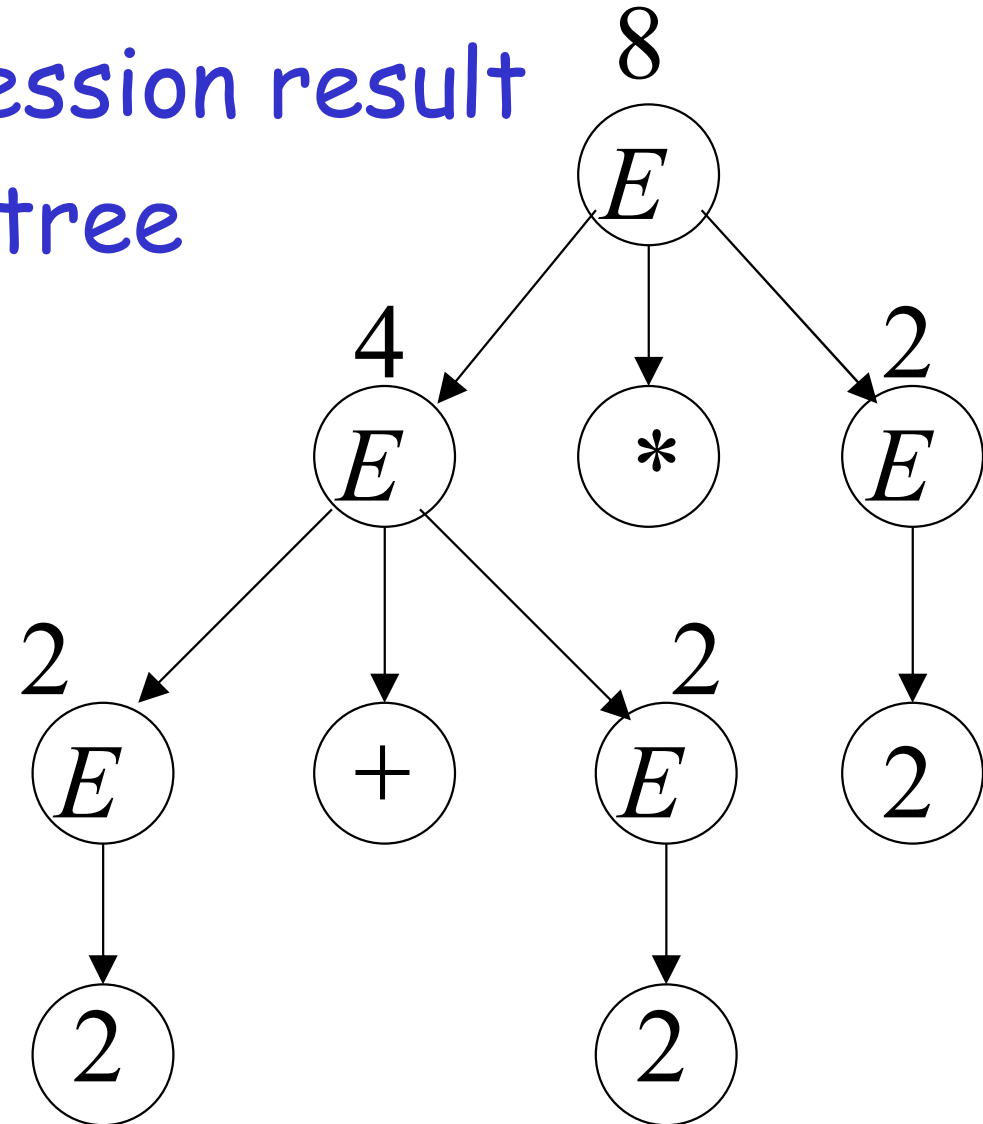
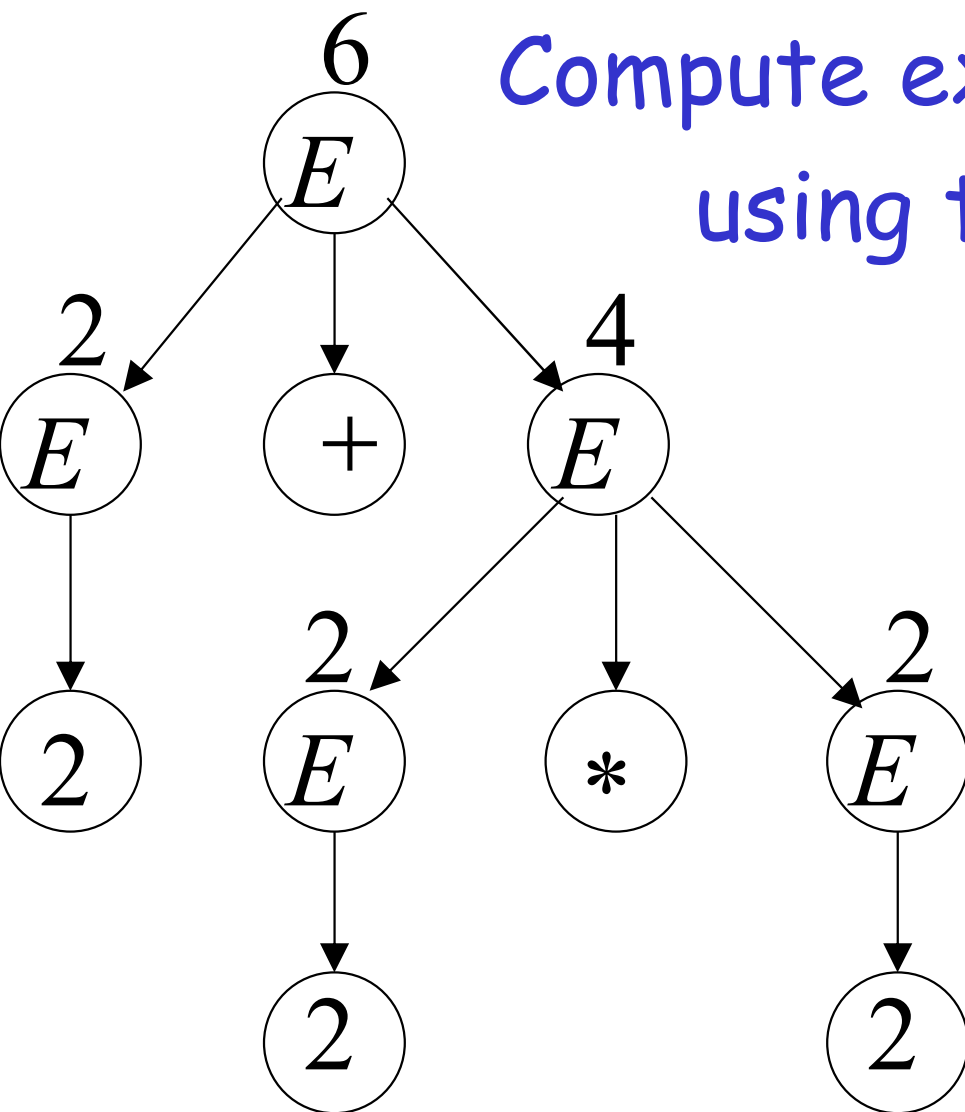
Good Tree

$$2 + 2 * 2 = 6$$

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

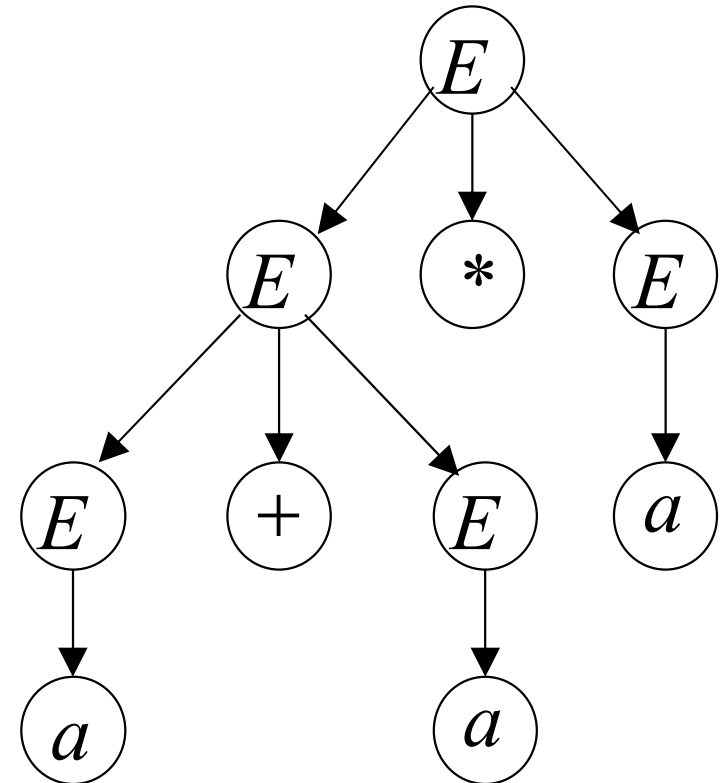
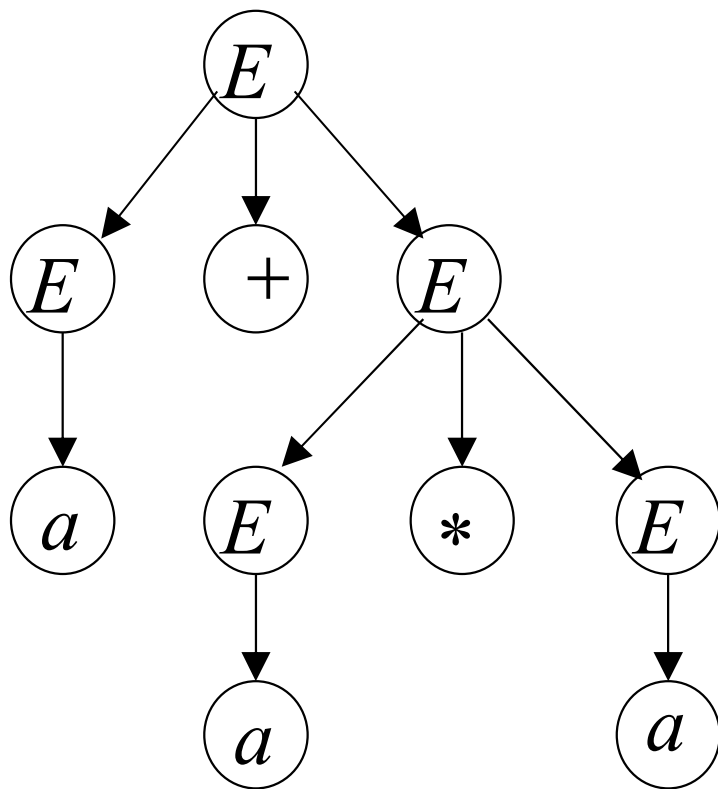
two leftmost derivations

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

Example:

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

this grammar is ambiguous since
string $a + a * a$ has two derivation trees



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

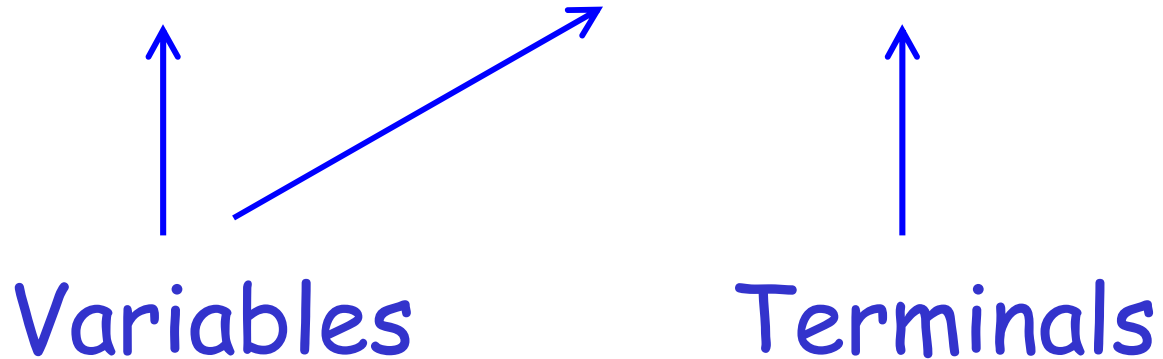
this grammar is ambiguous also because
string $a + a * a$ has two leftmost derivations

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

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

Another ambiguous grammar:

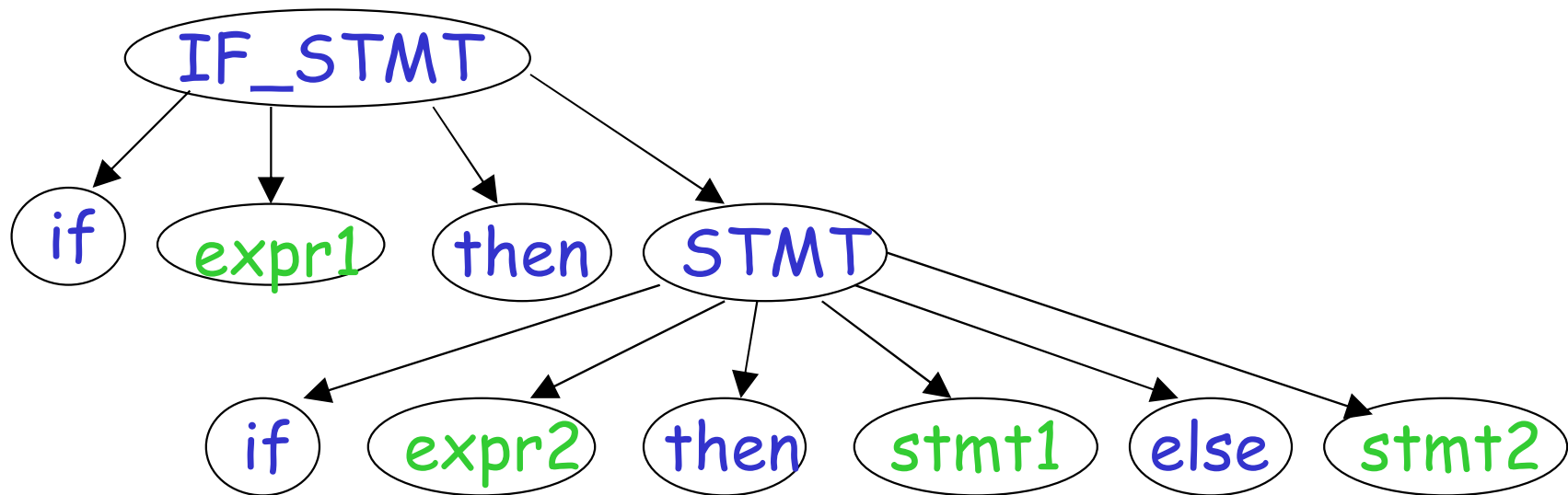
IF_STMT \rightarrow if EXPR then STMT
 | if EXPR then STMT else STMT



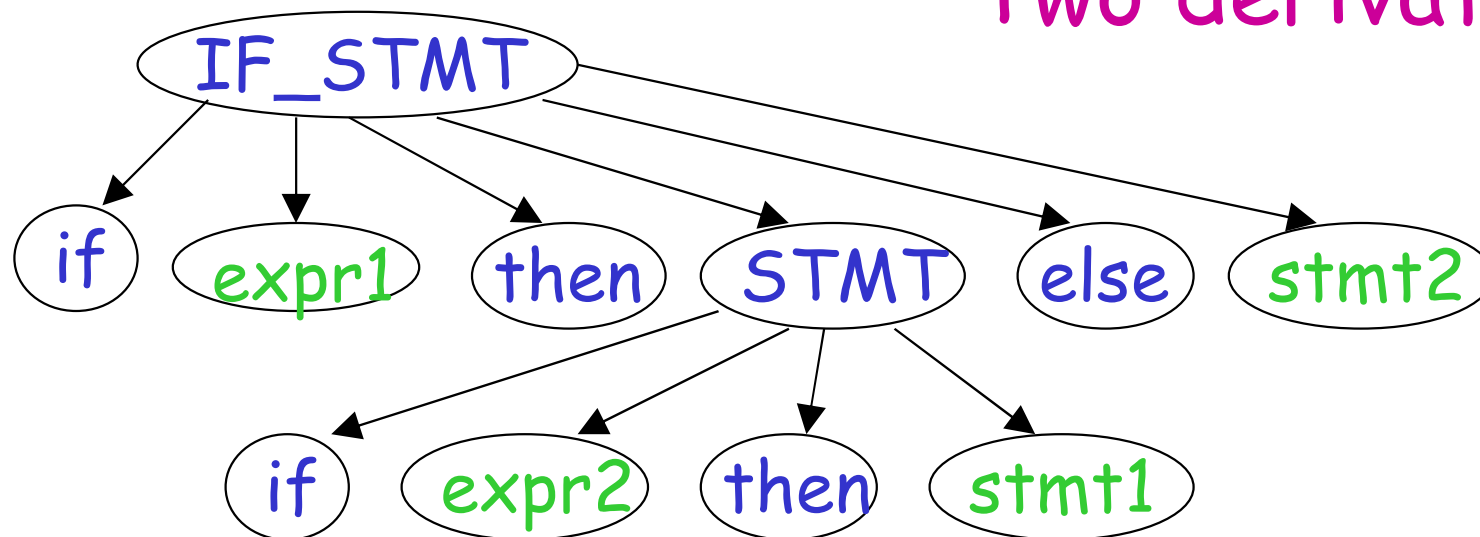
Variables Terminals

Very common piece of grammar
in programming languages

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



Two derivation trees



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. (I.e., there
is no single algorithm that takes as input any
ambiguous grammar and produces an equivalent
Unambiguous grammar.)

A successful example:

Ambiguous Grammar

$$E \rightarrow E + E$$

$$E \rightarrow E * E$$

$$E \rightarrow (E)$$

$$E \rightarrow a$$

Equivalent

Non-Ambiguous Grammar

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

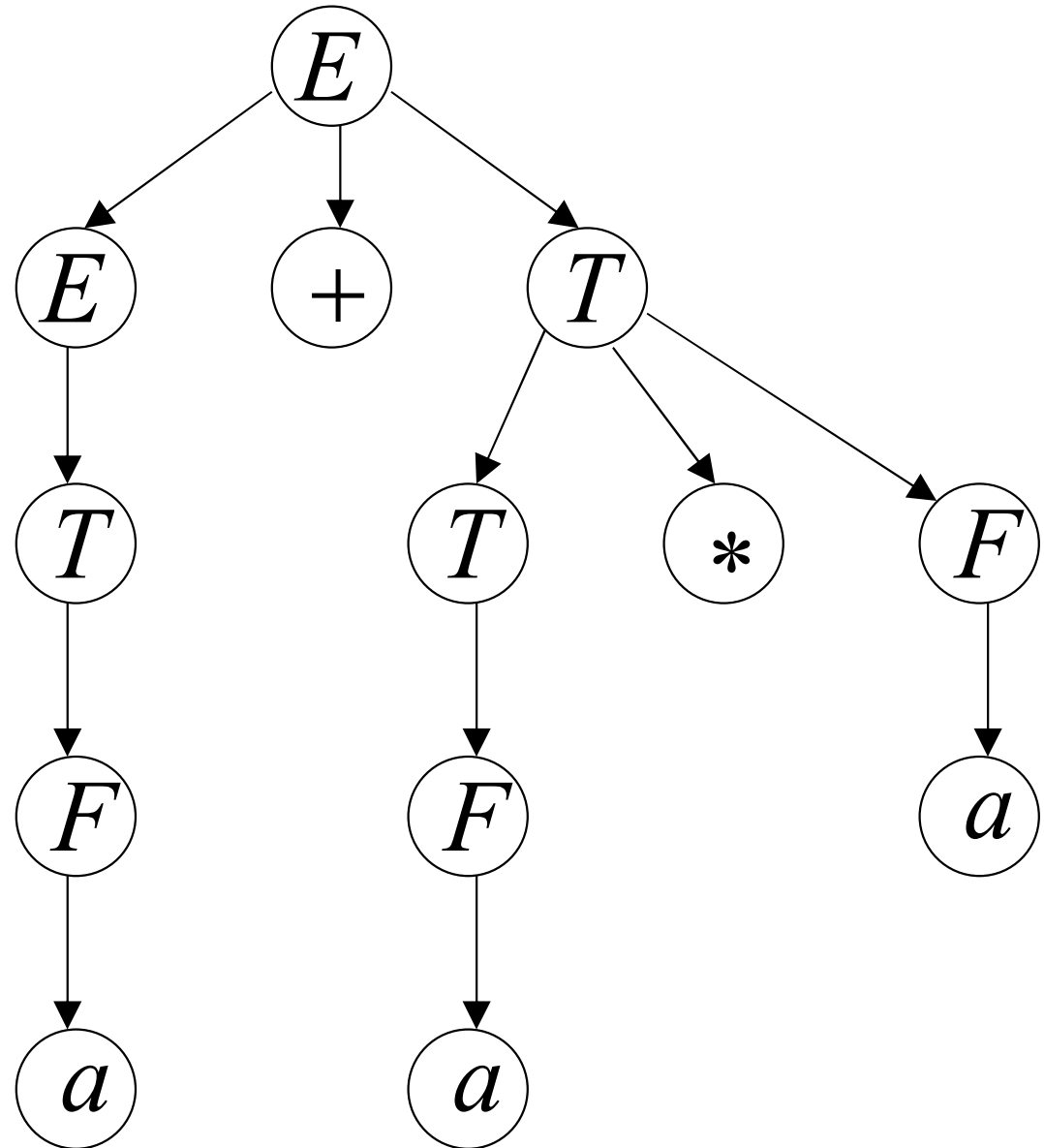
$$F \rightarrow (E) \mid a$$

generates the same language

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

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

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 :

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


$$S \rightarrow S_1 \mid S_2$$


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

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


$$S_2 \rightarrow aS_2 \mid B$$

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

Dealing with Ambiguity

There are several ways to handle ambiguity

Most direct method is to rewrite grammar unambiguously

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

$$E' \rightarrow \text{id} * E' \mid \text{id} \mid (E) * E' \mid (E)$$

Enforces precedence of * over +

Ambiguity

No general techniques for handling ambiguity

In general, impossible to convert automatically an arbitrary 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

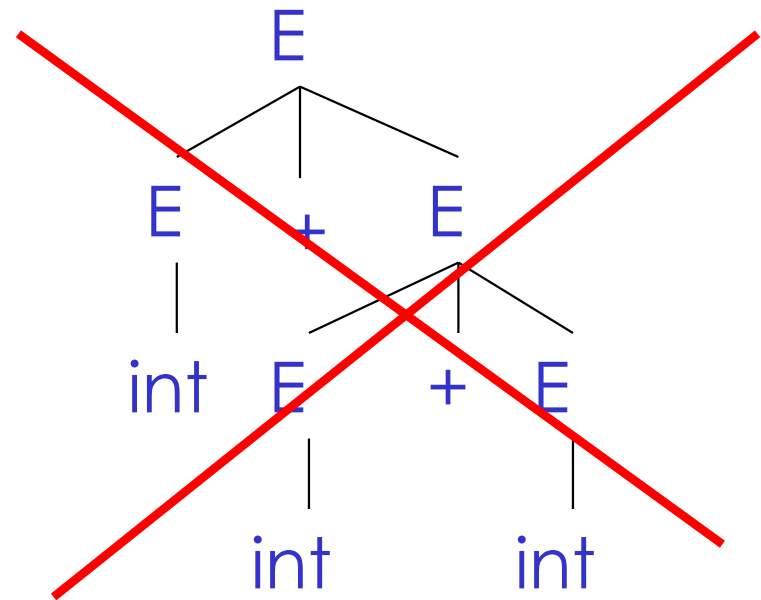
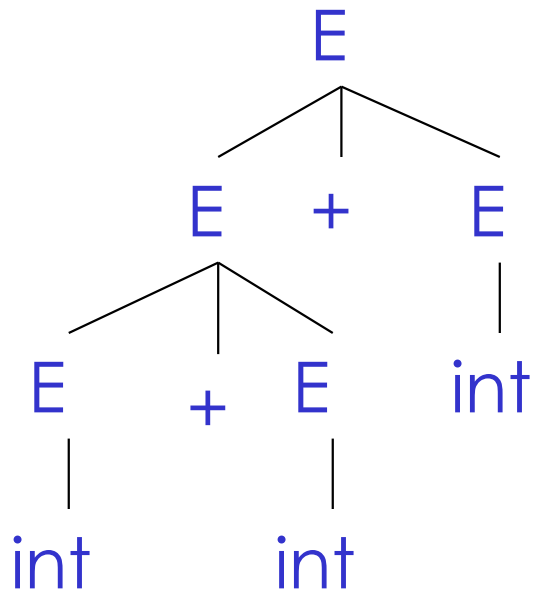
Use the more natural (ambiguous) grammar

Along with disambiguating declarations

Most tools allow precedence and associativity declarations to disambiguate grammars

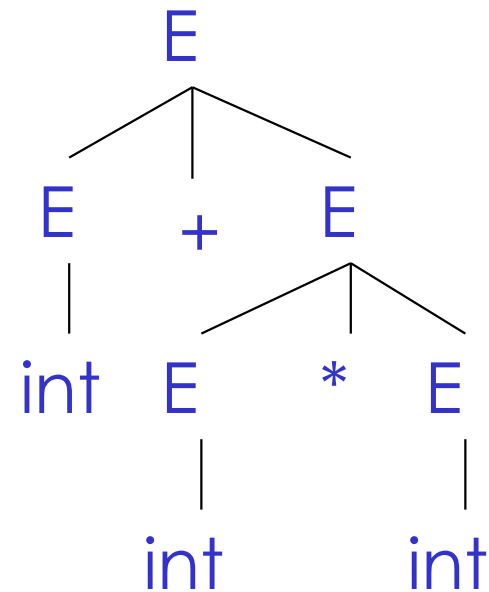
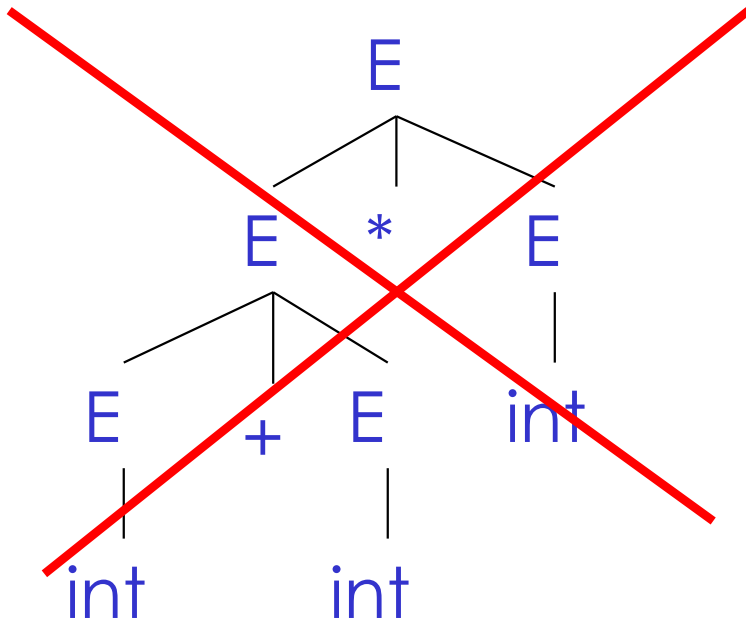
Associativity Declarations

- Ambiguous
 - two parse trees of $\text{int} + \text{int} + \text{int}$
- Left associativity declaration: `%left +`



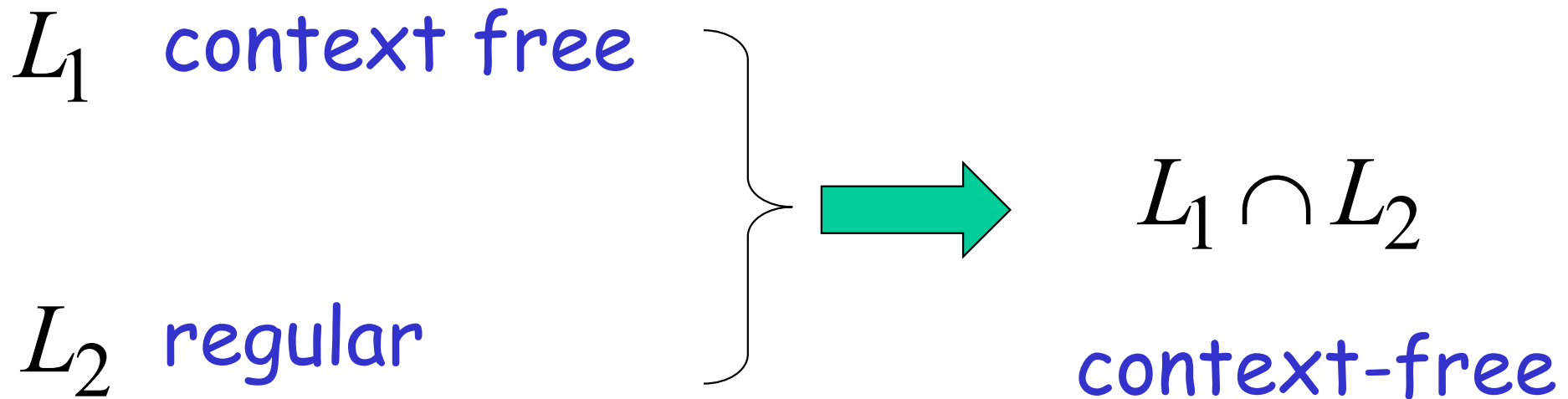
Precedence Declarations

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



Intersection of Context-free languages and Regular Languages

The intersection of
a context-free language and
a regular language
is a context-free language



Machine M_1

NPDA for L_1
context-free

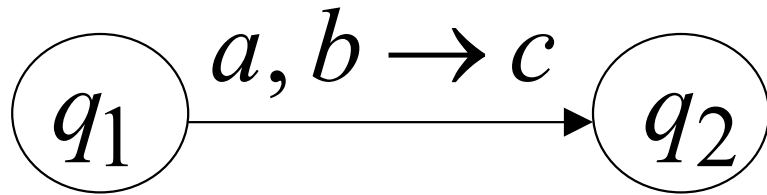
Machine M_2

DFA for L_2
regular

Construct a new NPDA machine M
that accepts $L_1 \cap L_2$

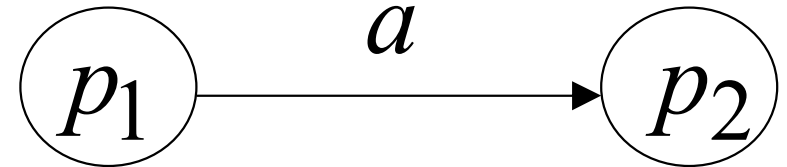
M simulates in parallel M_1 and M_2

NPDA M_1

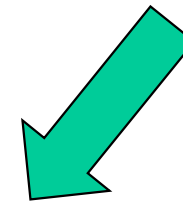
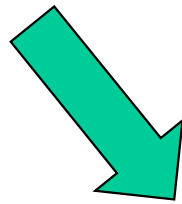


transition

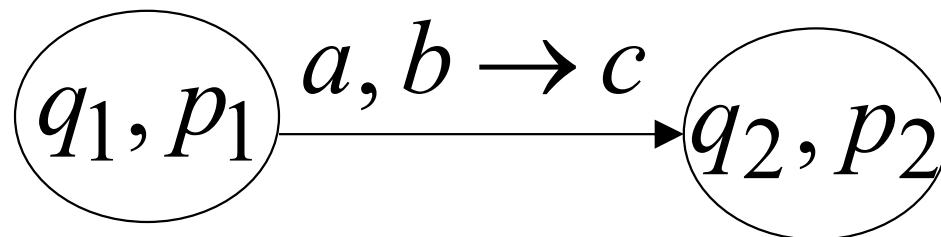
DFA M_2



transition

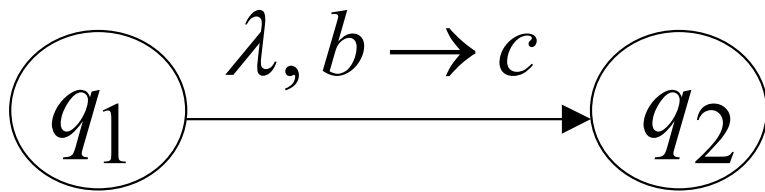


NPDA M



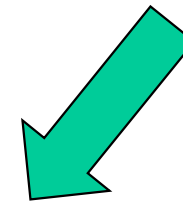
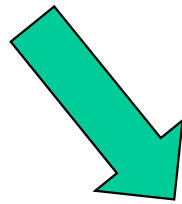
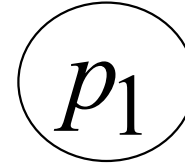
transition

NPDA M_1

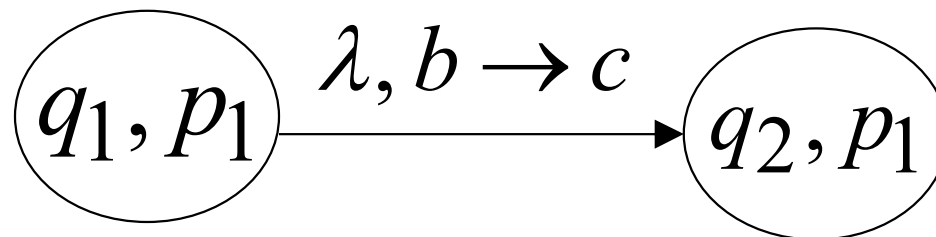


transition

DFA M_2

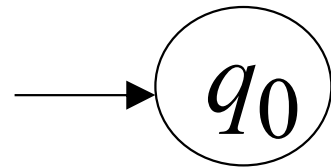


NPDA M



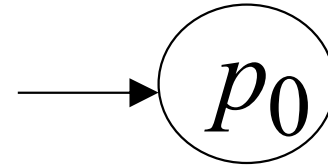
transition

NPDA M_1

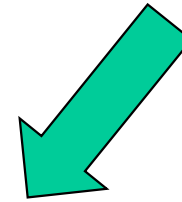
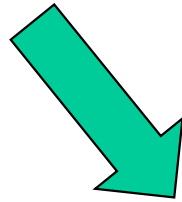


initial state

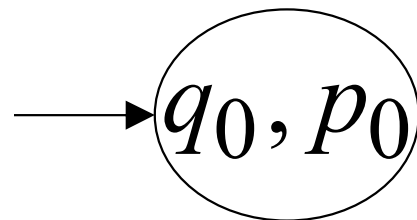
DFA M_2



initial state

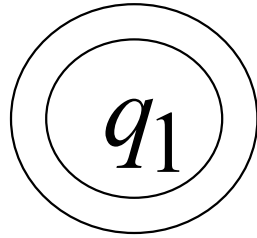


NPDA M



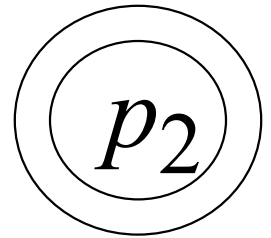
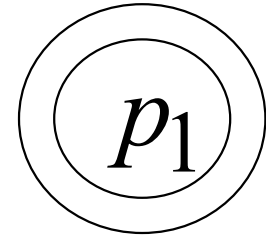
Initial state

NPDA M_1

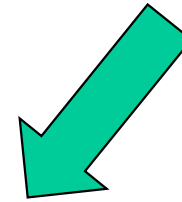
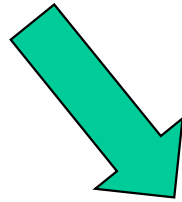


final state

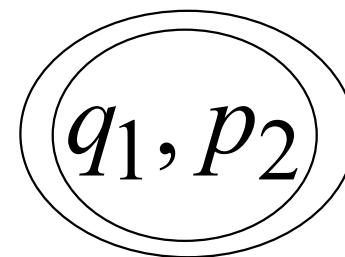
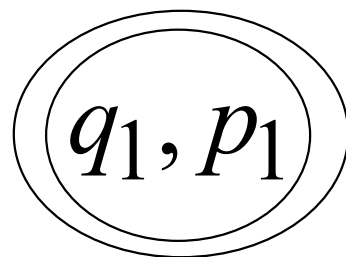
DFA M_2



final states



NPDA M



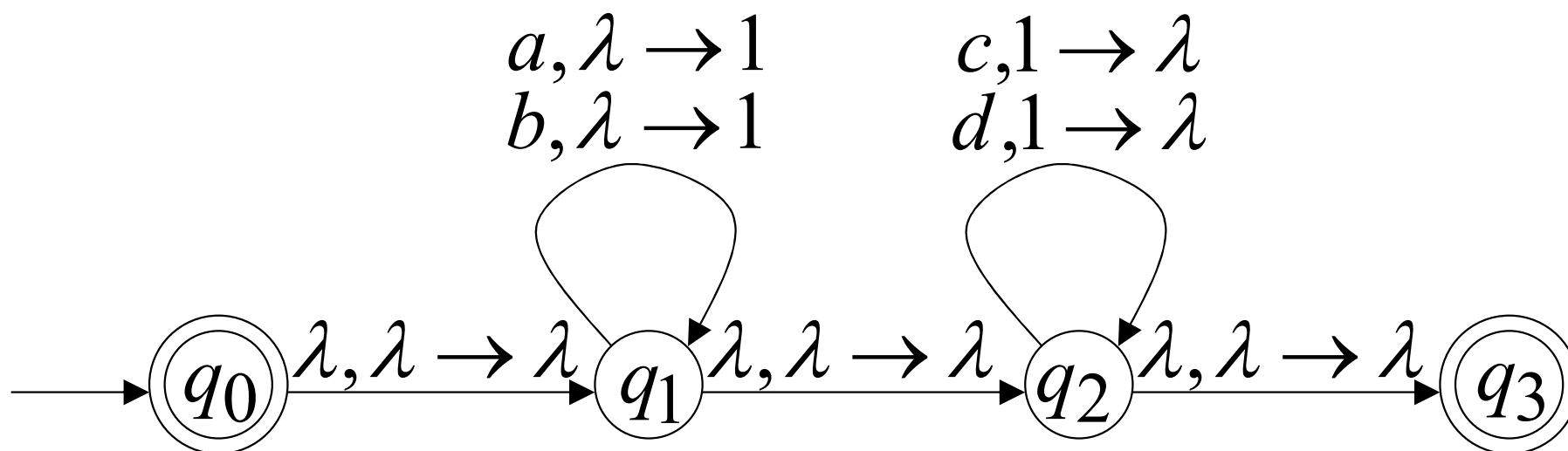
final states

Example:

context-free

$$L_1 = \{w_1 w_2 : |w_1| = |w_2|, w_1 \in \{a, b\}^*, w_2 \in \{c, d\}^*\}$$

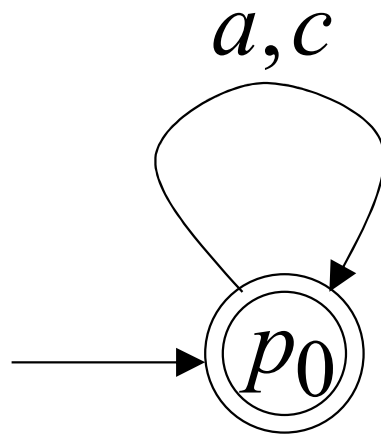
NPDA M_1



regular

$$L_2 = \{a, c\}^*$$

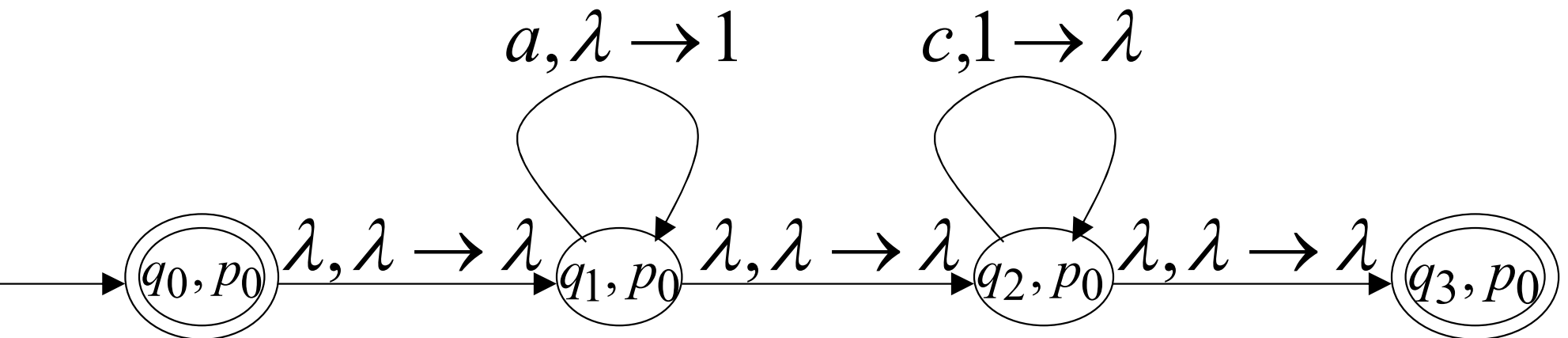
DFA M_2



context-free

Automaton for: $L_1 \cap L_2 = \{a^n c^n : n \geq 0\}$

NPDA M



In General:

M simulates in parallel M_1 and M_2

M accepts string w if and only if

M_1 accepts string w and

M_2 accepts string w

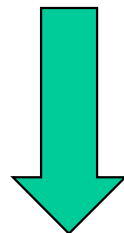
$$L(M) = L(M_1) \cap L(M_2)$$

Therefore:

M is NPDA



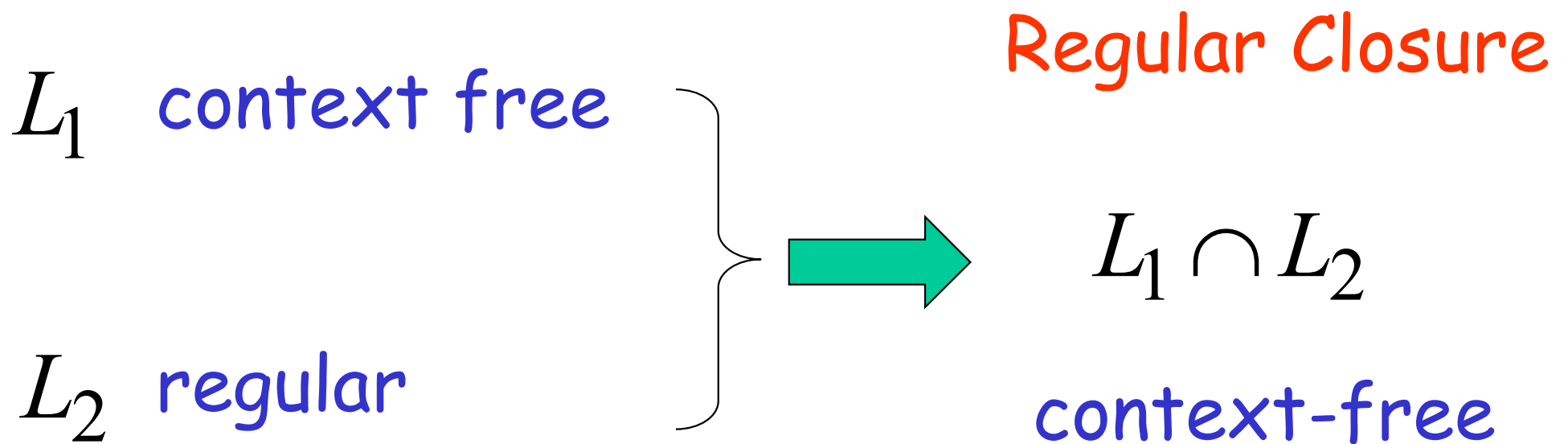
$L(M_1) \cap L(M_2)$ is context-free



$L_1 \cap L_2$ is context-free

Applications of Regular Closure

The intersection of
a context-free language and
a regular language
is a context-free language



An Application of Regular Closure

Prove that: $L = \{a^n b^n : n \neq 100, n \geq 0\}$
is context-free

We know:

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

We also know:

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



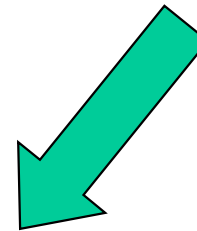
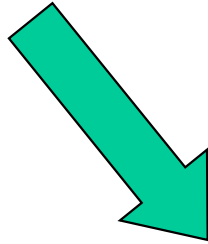
$$\overline{L_1} = \{(a+b)^*\} - \{a^{100}b^{100}\} \quad \text{is regular}$$

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

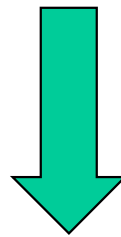
context-free

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

regular



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



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

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