Introduction to Languages and Grammars

Alphabets and Languages

An alphabet is a finite non-empty set.

Let S and T be alphabets.

$S \cdot T = \{ s\ t \mid s \in S, t \in T \}$

(We'll often write ST for $S \cdot T$.)

λ = empty string, string of length one

$S^0 = \{\lambda\}$

$S^1 = S$

$S^n = S^{(n-1)} \cdot S,\ n > 1$

$S^+ = S^1 \cup S^2 \cup S^3 \cup \ldots$

$S^* = S^0 \cup S^+$

A language L over an alphabet S is a subset of S^*.

How Many Languages Are There?

- How many languages over a particular alphabet are there?

  Uncountably infinitely many!
- Then, any finite method of describing languages can not include all of them.
- Formal language theory gives us techniques for defining some languages over an alphabet.

Why should I care about ?

- Concepts of syntax and semantics used widely in computer science:
- Basic compiler functions
- Development of computer languages
- Exploring the capabilities and limitations of algorithmic problem solving

Methods for Defining Languages

- Grammar
  - Rules for defining which strings over an alphabet are in a particular language
- Automaton (plural is automata)
  - A mathematical model of a computer which can determine whether a particular string is in the language

Definition of a Grammar

- A grammar G is a 4 tuple G = (N, Σ, P, S), where
  - N is an alphabet of nonterminal symbols
  - Σ is an alphabet of terminal symbols
  - N and Σ are disjoint
  - S is an element of N; S is the start symbol or initial symbol of the grammar
  - P is a set of productions of the form α -> β where
    - α is in (N U Σ)* N (N U Σ)*
    - β is in (N U Σ)*

Definition of a Language Generated by a Grammar

We define $\Rightarrow$ by

$\gamma \alpha \delta \Rightarrow \gamma \delta \beta$ if

$\alpha \rightarrow \beta$ is in P, and

γ and δ are in $(N \cup \Sigma)^*$

$\Rightarrow^+$ is the transitive closure of $\Rightarrow$

$\Rightarrow^*$ is the reflexive transitive closure of $\Rightarrow$

The language L generated by grammar G = (N, Σ, P, S), is defined by

$L = L(G) = \{ x \mid S \Rightarrow^* x \text{ and } x \text{ is in } \Sigma^* \}$

Classes of Grammars (The Chomsky Hierarchy)

- Type 0, Phrase Structure (same as basic grammar definition)
- Type 1, Context Sensitive
  - (1) $\alpha \rightarrow \beta$ where α is in $(N \cup \Sigma)^* N (N \cup \Sigma)^*$,
    - β is in $(N \cup \Sigma)+$, and length(α) $\leq$ length(β)
  - (2) $\gamma A \delta \rightarrow \gamma \beta \delta$ where A is in N, β is in $(N \cup \Sigma)^+$, and
    - γ and δ are in $(N \cup \Sigma)^*$
- Type 2, Context Free
  - $A \rightarrow \beta$ where A is in N, β is in $(N \cup \Sigma)^*$
- Linear
  - $A \rightarrow x$ or $A \rightarrow x B y$, where A and B are in N and x and y are in Σ^*
- Type 3, Regular Expressions
  - (1) left linear $A \rightarrow B a$ or $A \rightarrow a$, where A and B are in N and a is in Σ
  - (2) right linear $A \rightarrow a B$ or $A \rightarrow a$, where A and B are in N and a is in Σ

Comments on the Chomsky Hierarchy (1)

- Definitions (1) and (2) for context sensitive are equivalent.
- Definitions (1) and (2) for regular expressions are equivalent.
- If a grammar has productions of all three of the forms described in definitions (1) and (2) for regular expressions, then it is a linear grammar.
- Each definition of context sensitive is a restriction on the definition of phrase structure.
- Every context free grammar can be converted to a context sensitive grammar with satisfies definition (2) which generates the same language except the language generated by the context sensitive grammar cannot contain the empty string λ.
- The definition of linear grammar is a restriction on the definition of context free.
- The definitions of left linear and right linear are restrictions on the definition of linear.

Comments on the Chomsky Hierarchy

- Every language generated by a left linear grammar can be generated by a right linear grammar, and every language generated by a right linear grammar can be generated by a left linear grammar.
- Every language generated by a left linear or right linear grammar can be generated by a linear grammar.
- Every language generated by a linear grammar can be generated by a context free grammar.
- Let L be a language generated by a context free grammar. If L does not contain λ, then L can be generated by a context sensitive grammar. If L contains λ, then $L-\{\lambda\}$ can be generated by a context sensitive grammar.
- Every language generated by a context sensitive grammar can be generated by a phrase structure grammar.

Example : A Left Linear Grammar for Identifiers

- S -> S a
- S -> S b
- S -> S 1
- S -> S 2
- S -> a
- S -> b

- S => a
- S => S 1 => a 1
- S => S 2 => S b 2
  => S 1 b 2 => a 1 b 2

Example : A Right Linear Grammar for Identifiers

- S -> a T T -> 1 T
- S -> b T T -> 2 T
- S -> a T -> a
- S -> b T -> b
- T -> a T T -> 1
- T -> b T T -> 2

- S => a
- S => a T => a 1
- S => a T => a 1 T
  => a 1 b T => a 1 b 2

Example:
A Right Linear Grammar for $\{a^n\ b^m\ c^p \mid n, m, p > 0\}$

- S -> a A
- A -> a A
- A -> b B
- B -> b B
- B -> c C
- B -> c
- C -> c C
- C -> c

S => a A => a a A
=> a a a A
=> a a a b B
=> a a a b c C
=> a a a b c c

Example : A Linear Grammar for { a^n b^n | n >0 }

- S -> a S b
- S -> a b

S => a S b
=> a a S b b
=> a a a S b b b
=> a a a a b b b b

Example : A Linear Grammar for $\{a^n b^m c^m d^n \mid n > 0\}$ (Context Free Grammar)

- S -> a S b
- S -> a T b
- T -> c T d
- T -> c d

S => a S b
=> a a S b b
=> a a a T b b b
=> a a a c T d b b b
=> a a a c c d d b b b

Another Example :
A Context Free Grammar for $\{a^n b^n c^m d^m \mid n > 0\}$

- S -> R T
- R -> a R b
- R -> a b
- T -> c T d
- T -> c d

S => R T
=> a R b T
=> a a R b b T
=> a a a b b b T
=> a a a b b b c T d
=> a a a b b b c c d d

A Context Free Grammar for Expressions

- S -> E
- E -> E + T
- E -> E - T
- E -> T
- T -> T * F
- T -> T / F
- T -> F

F -> (E)
F -> a
F -> b
F -> c
F -> d
F -> e

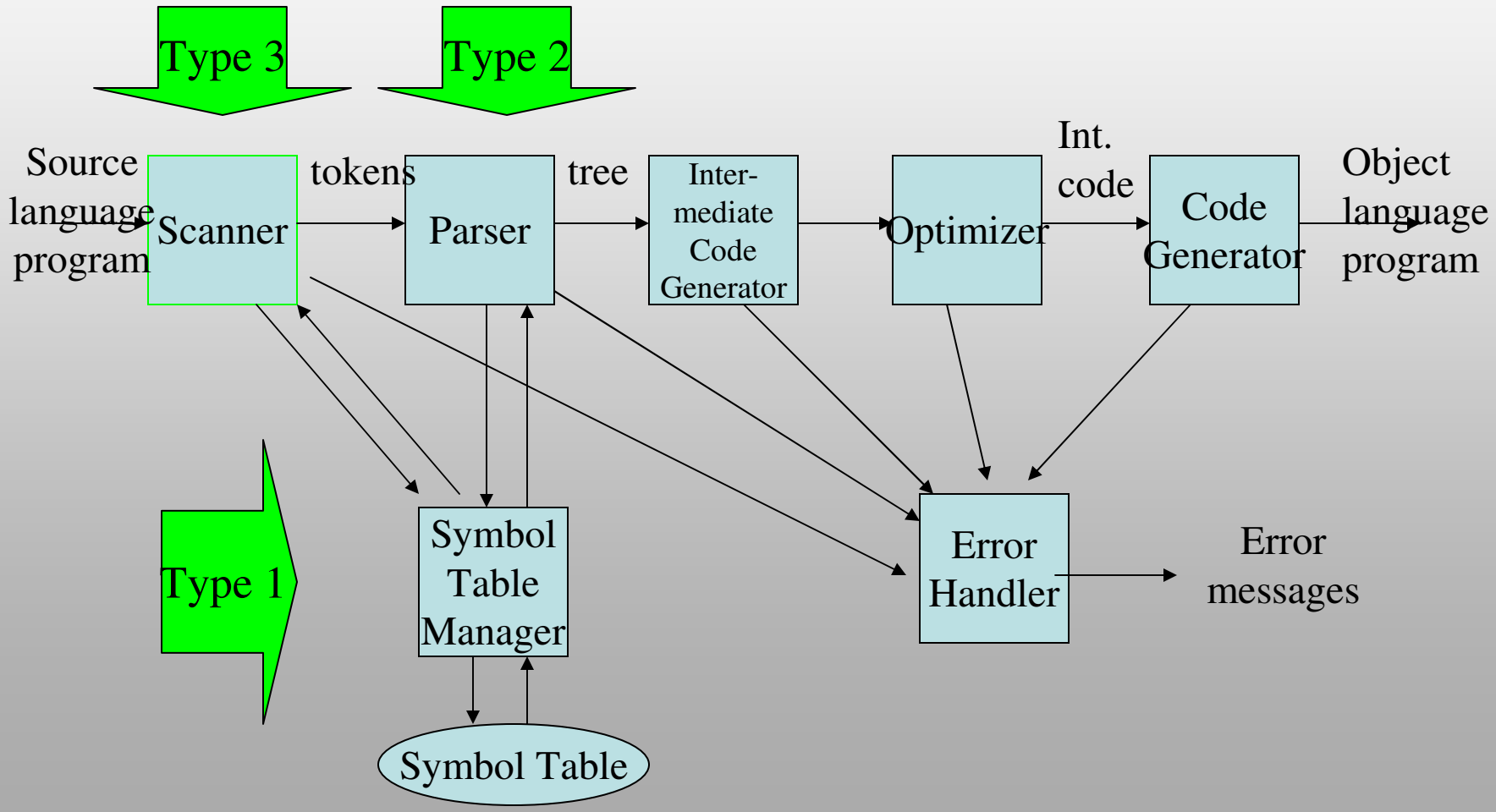
- S => E => E + T
  - => E - T + T
  - => T - T + T
  - => F - T + T
  - => a - T + T
  - => a - T * F + T
  - => a - F * F + T
  - => a - b * F + T
  - => a - b * c + T
  - => a - b * c + F
  - => a - b * c - d

A Context Sensitive Grammar for $\{a^n\ b^n\ c^n \mid n > 0\}$

- S -> a S B C
- S -> a B C
- a B -> a b
- b B -> b b
- C B -> B C
- b C -> b c
- c C -> c c

S => a S B C
=> a a B C B C
=> a a b C B C
=> a a b B C C
=> a a b b C C
=> a a b b c C
=> a a b b c c

The Chomsky Hierarchy and the Block Diagram of a Compiler



Automata

- Turing machine (Tm)
- Linear bounded automaton (lba)
- 2-stack pushdown automaton (2pda)
- (1-stack) pushdown automaton (pda)
- 1 turn pushdown automaton
- finite state automaton (fsa)

Recursive Definition

Primitive regular expressions: $\emptyset, \quad \lambda, \quad \alpha$

Given regular expressions r_1 and r_2

$r_1 + r_2$

$r_1 \cdot r_2$

r_1^*

(r_1)

} Are regular expressions

Example $(a+b)\cdot a^*$

$$
\begin{aligned}
L((a+b)\cdot a^*) &= L((a+b))\,L(a^*) \\
&= L(a+b)\,L(a^*) \\
&= (L(a)\cup L(b))\,(L(a))^* \\
&= (\{a\}\cup\{b\})\,(\{a\})^* \\
&= \{a,b\}\{\lambda,a,aa,aaa,\ldots\} \\
&= \{a,aa,aaa,\ldots,b,ba,baa,\ldots\}
\end{aligned}
$$

Example $(a+b)\cdot a^*$

$$
\begin{aligned}
L((a+b)\cdot a^*) &= L((a+b))\,L(a^*) \\
&= L(a+b)\,L(a^*) \\
&= (L(a)\cup L(b))\,(L(a))^* \\
&= (\{a\}\cup\{b\})\,(\{a\})^* \\
&= \{a,b\}\{\lambda,a,aa,aaa,\ldots\} \\
&= \{a,aa,aaa,\ldots,b,ba,baa,\ldots\}
\end{aligned}
$$

Example

- Regular expression $r = (aa)^*(bb)^*b$

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

Example

Regular expression $r = (0+1)^*00\,(0+1)^*$

$L(r)$ = { all strings with at least two consecutive 0 }

Example

- Regular expression $r = (1+01)^*(0+\lambda)$

$L(r)$ = { all strings without two consecutive 0 }

Equivalent Regular Expressions

- Definition:

  Regular expressions r_1 and r_2

  are **equivalent** if $L(r_1) = L(r_2)$

Example

L = { all strings without two consecutive 0 }

$$r_1 = (1+01)^*(0+\lambda)$$

$$r_2 = (1^*011^*)^*(0+\lambda)+1^*(0+\lambda)$$

$$L(r_1) = L(r_2) = L \Longrightarrow$$

r_1 and r_2 are equivalent regular expr.

Linear Grammars

Grammars with
at most one variable at the right side
of a production

Examples:

$$S \to aSb \qquad S \to Ab$$

$$S \to \lambda \qquad A \to aAb$$

$$A \to \lambda$$

A Non-Linear Grammar

Grammar G :

$$S \to SS$$

$$S \to \lambda$$

$$S \to aSb$$

$$S \to bSa$$

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

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: $A \rightarrow xB$

  or

  $A \rightarrow x$

  string of terminals

- Example:

  $S \rightarrow abS$

  $S \rightarrow a$

Left-Linear Grammars

All productions 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

$S \to Aab$

$A \to Aab \mid B$

$B \to a$

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

Closure under language operations

- Theorem. The set of regular languages is closed under the all the following operations. In other words if L_1 and L_2 are regular, then so is:
  - Union: $L_1 \cup L_2$
  - Intersection: $L_1 \cap L_2$
  - Complement: $L_1^c = \Sigma^* \setminus L_1$
  - Difference: $L_1 \setminus L_2$
  - Concatenation: $L_1 L_2$
  - Kleene star: L_1^*

Regular expressions versus regular languages

- We defined a *regular language* to be one that is accepted by some DFA (or NFA).
- We can prove that a language is regular by this definition if and only if it corresponds to some regular expression. Thus,
  - 1) Given a DFA or NFA, there is some regular expression to describe the language it accepts
  - 2) Given a regular expression, we can construct a DFA or NFA to accept the language it represents

Application: Lexical-analyzers and lexical-analyzer generators

- *Lexical analyzer:*
  - Input: Character string comprising a computer program in some language
  - Output: A string of symbols representing *tokens* – elements of that language
- Ex in C++ or Java :
  - Input: if (x == 3) y = 2;
  - Output (sort of): if-token, expression-token, variable-name-token, assignment-token, numeric-constant token, statement-separator-token.

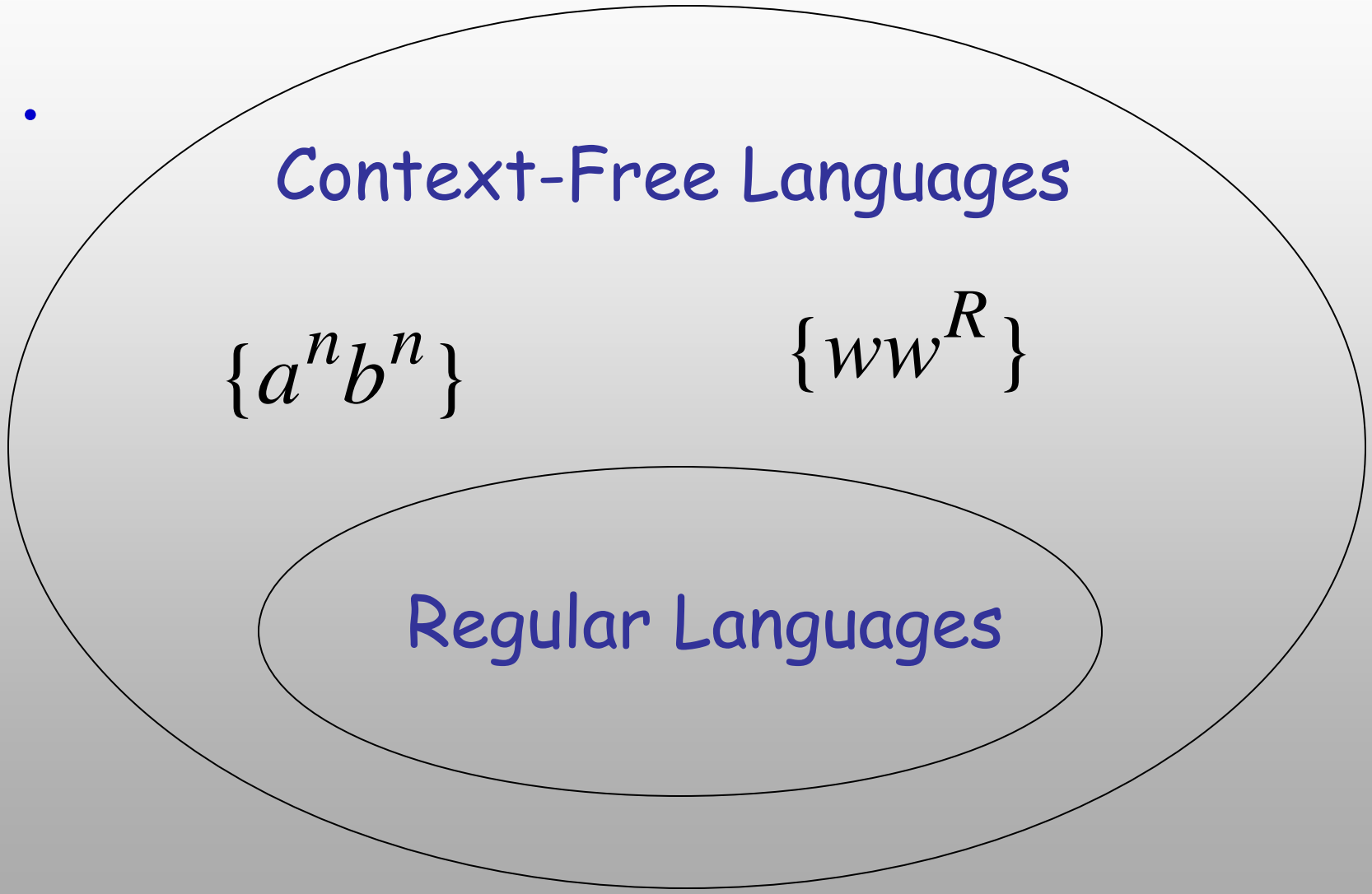
Lexical-analyzer generators

- Input: A list of tokens in a programming language, described as regular expressions
- Output: A lexical analyzer for that language
- Technique: Builds an NFA recognizing the language tokens, then converts to DFA.

Regular Expressions: Applications and Limitations

- Regular expressions have many applications:
  - Specification of syntax in programming languages
  - Design of lexical analyzers for compilers
  - Representation of patterns to match in search engines
  - Provide an abstract way to talk about *programming problems* (language correpsonds to inputs that produce output *yes* in a yes/no programming problem)
- Limitations
  - There are lots of reasonable languages that are *not* regular – hence lots of programming problems that can't be solved using power of a DFA or NFA

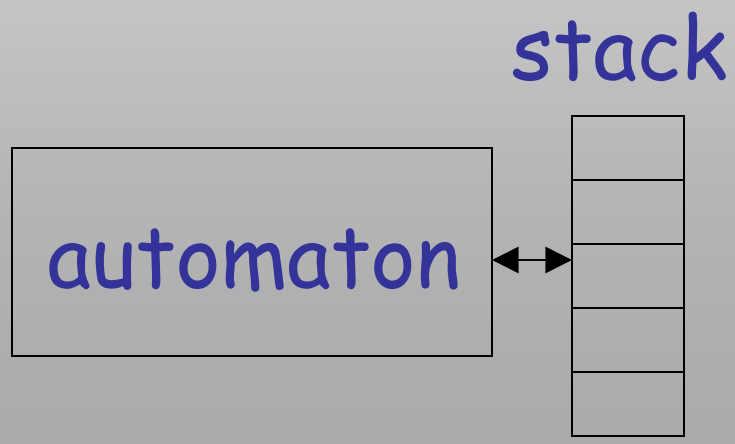
CFL



Context-Free Languages

Context-Free Grammars

Pushdown Automata



Example

A context-free grammar G:

$$S \to aSb$$

$$S \to \lambda$$

A derivation:

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

$$S \to aSb$$

$$S \to \lambda$$

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

Describes parentheses: (((( ))))

Example

A context-free grammar G:

$$S \to aSa$$

$$S \to bSb$$

$$S \to \lambda$$

A derivation:

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

$$S \to aSa$$

$$S \to bSb$$

$$S \to \lambda$$

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

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:

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

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

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

$$\Rightarrow abbbbB \Rightarrow abbbb$$

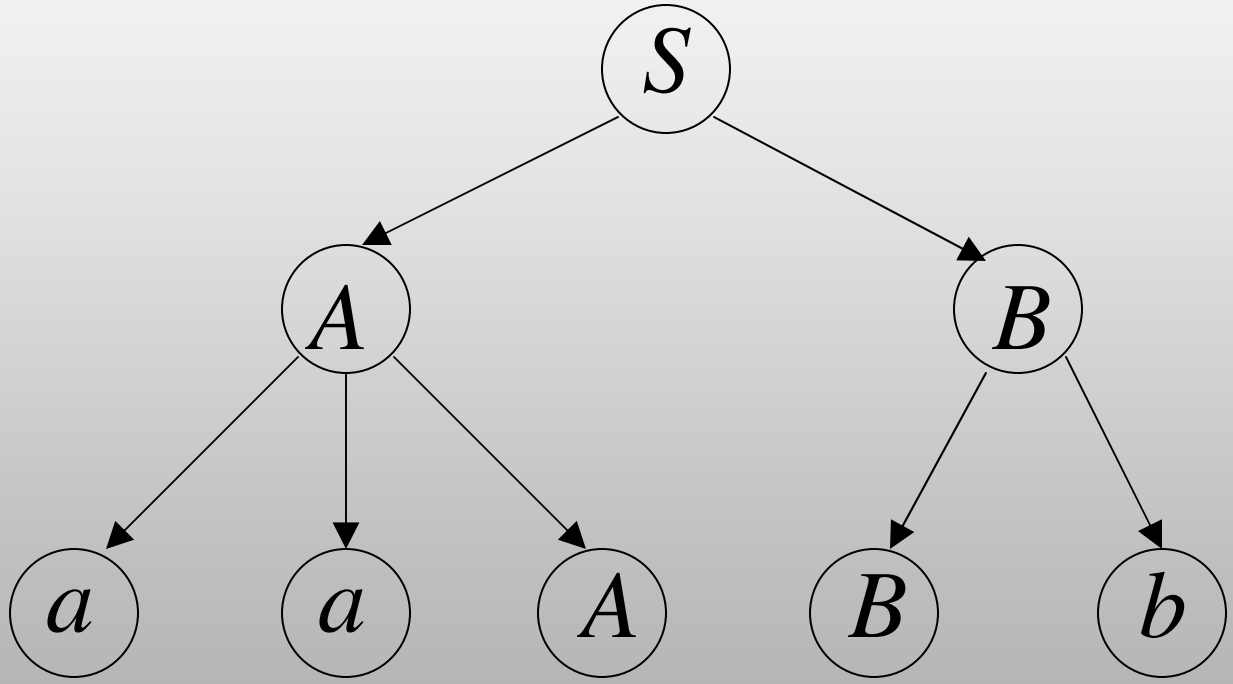
Rightmost derivation:

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

$$\Rightarrow abbBbb \Rightarrow abbbb$$

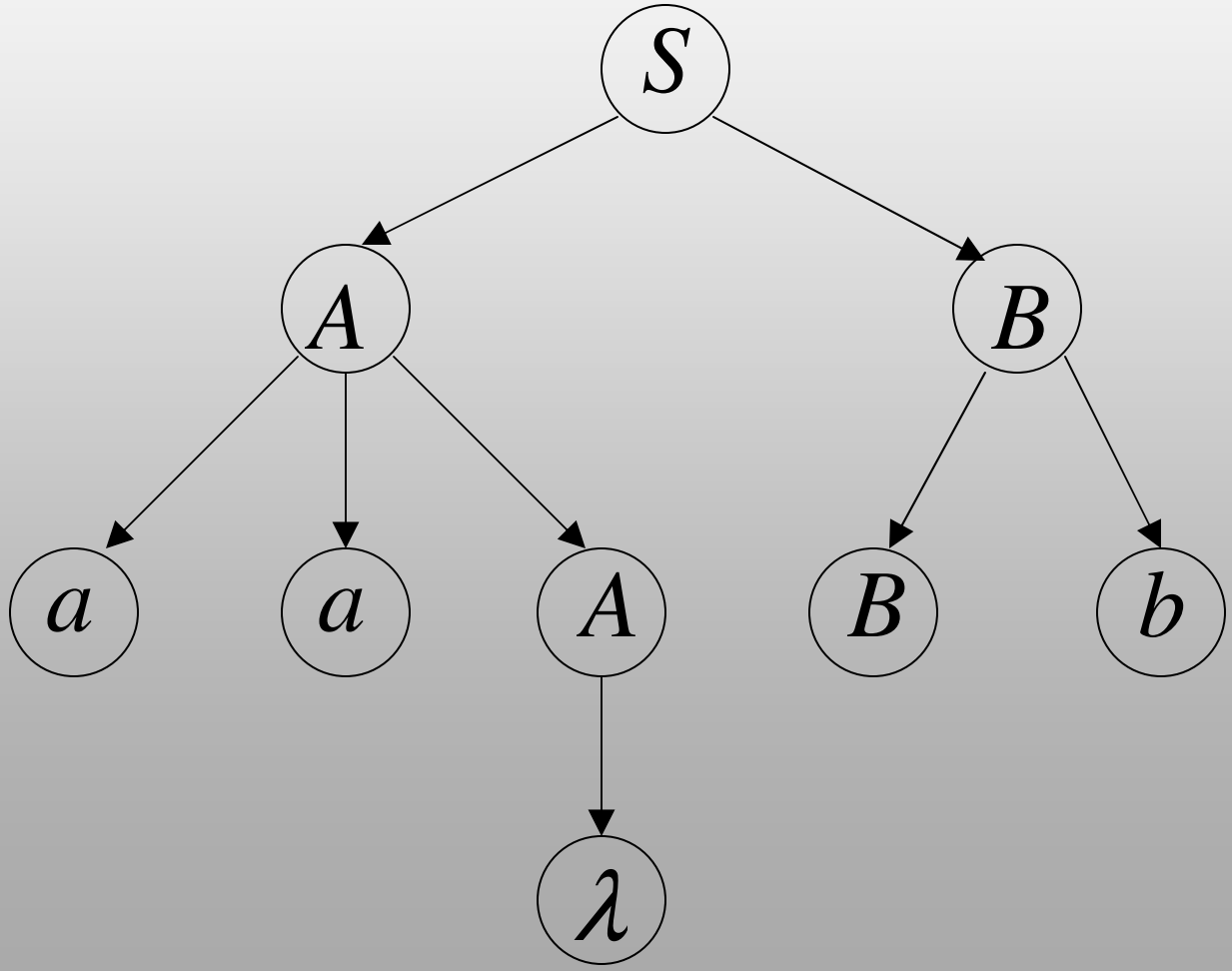
$S \rightarrow AB$ $A \rightarrow aaA \mid \lambda$ $B \rightarrow Bb \mid \lambda$

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



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

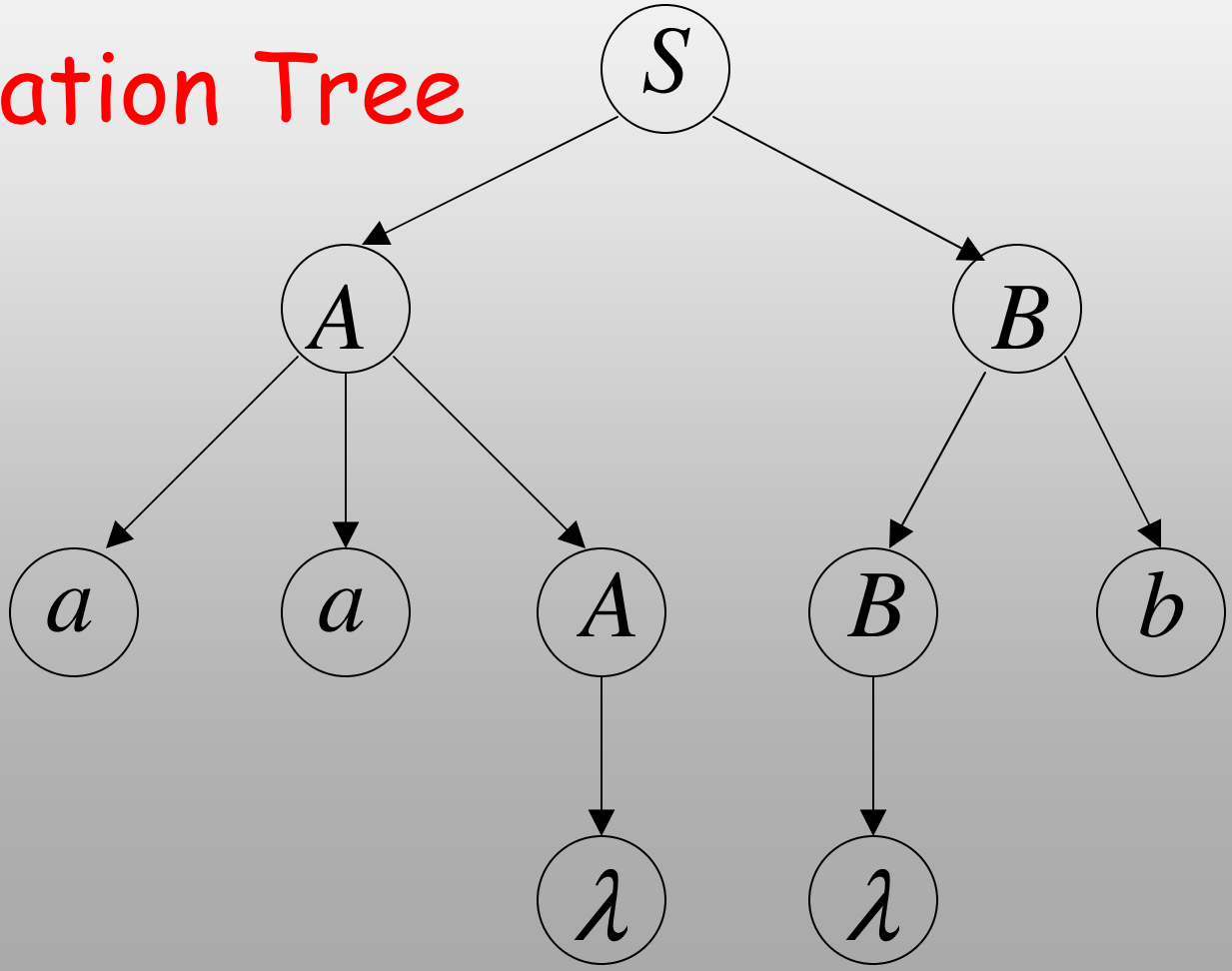
$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb \Rightarrow aaBb$



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

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

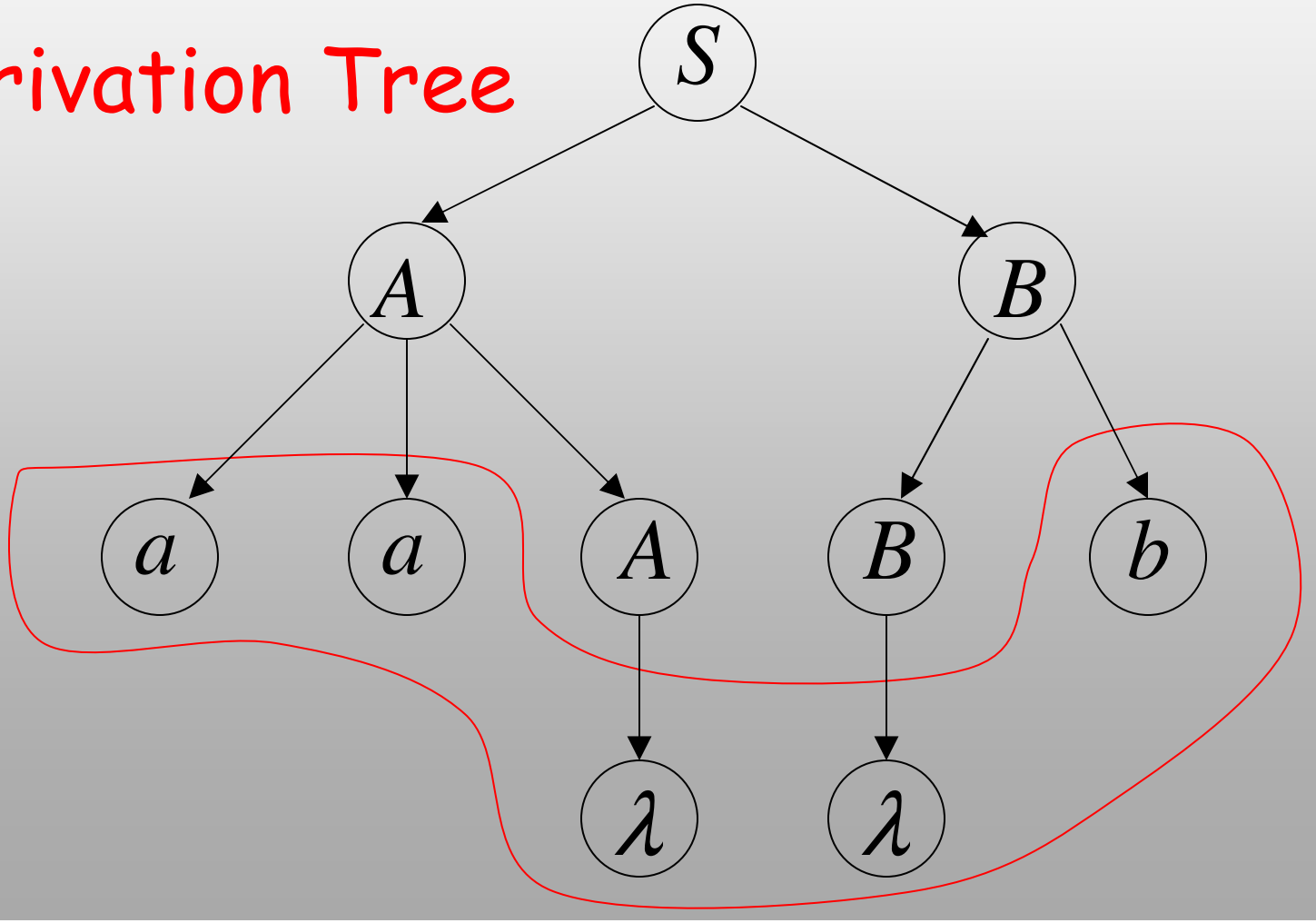
Derivation Tree



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

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

Derivation Tree



yield

$aa\lambda\lambda b$

$= aab$

Sometimes, derivation order doesn't matter

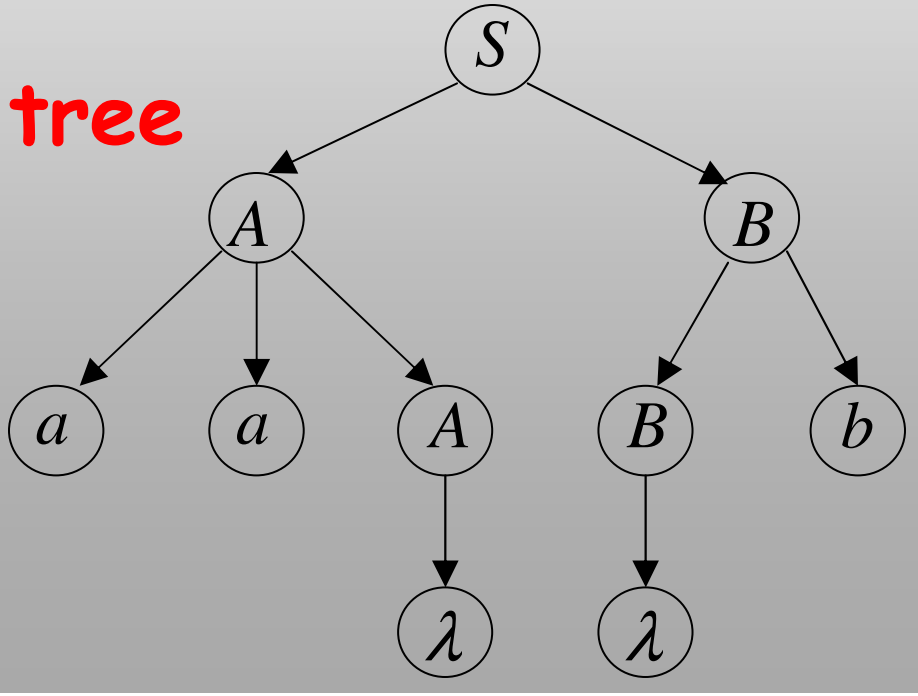
Leftmost:

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

Rightmost:

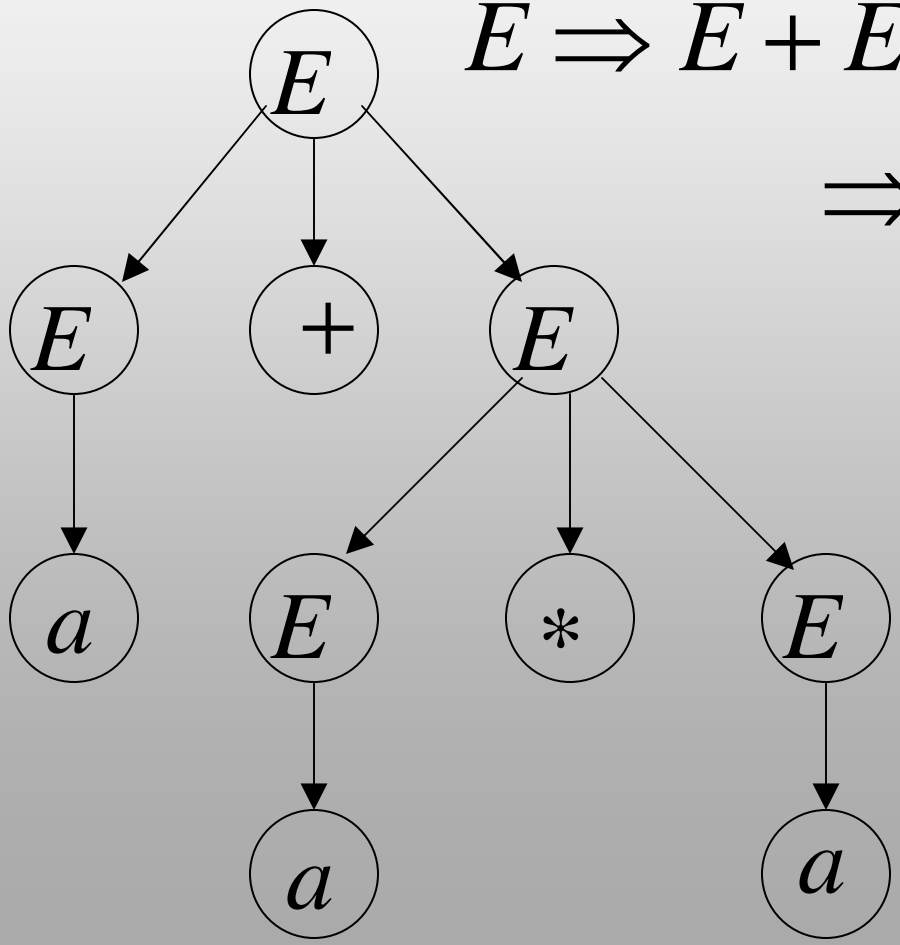
$$S \Rightarrow AB \Rightarrow ABb \Rightarrow Ab \Rightarrow aaAb \Rightarrow aab$$

Same derivation tree



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

$$a+a*a$$



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

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

leftmost derivation

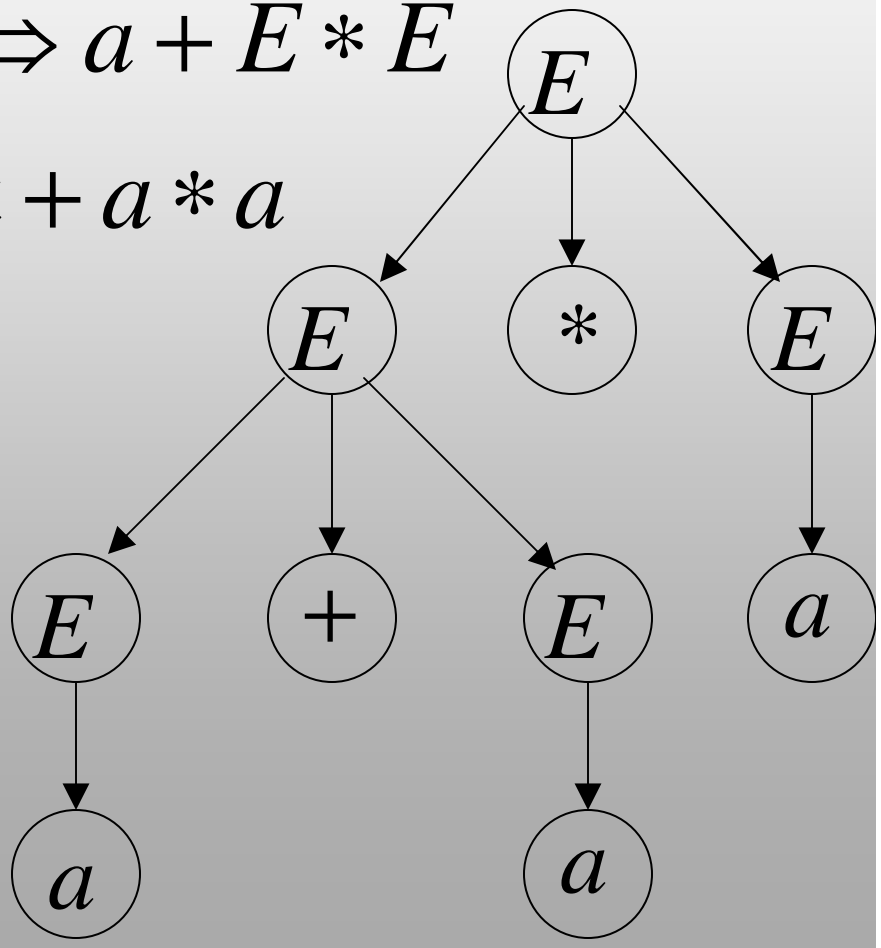
$$E \rightarrow E+E \mid E*E \mid (E) \mid a$$

$$a+a*a$$

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

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

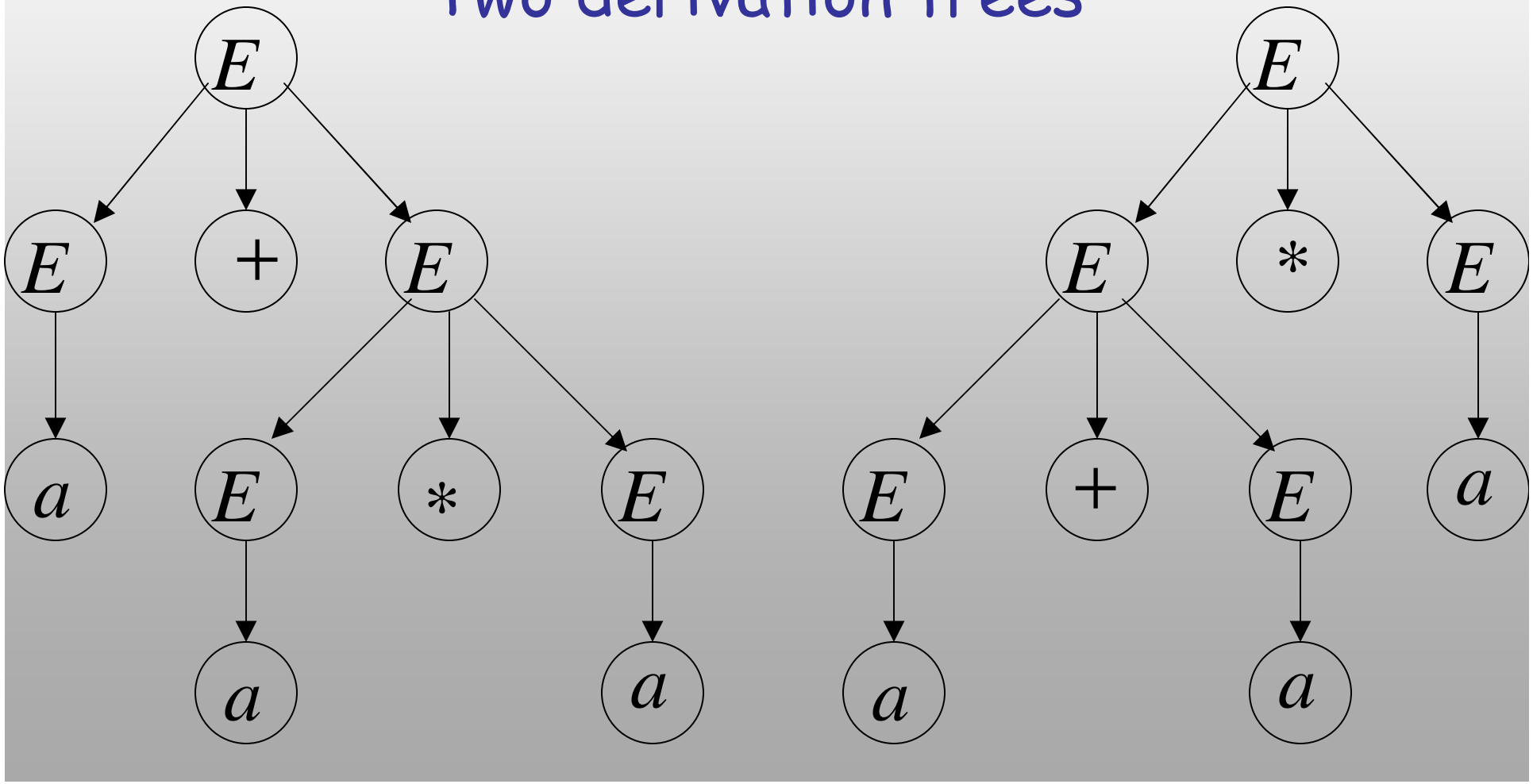
leftmost derivation



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

$$a+a*a$$

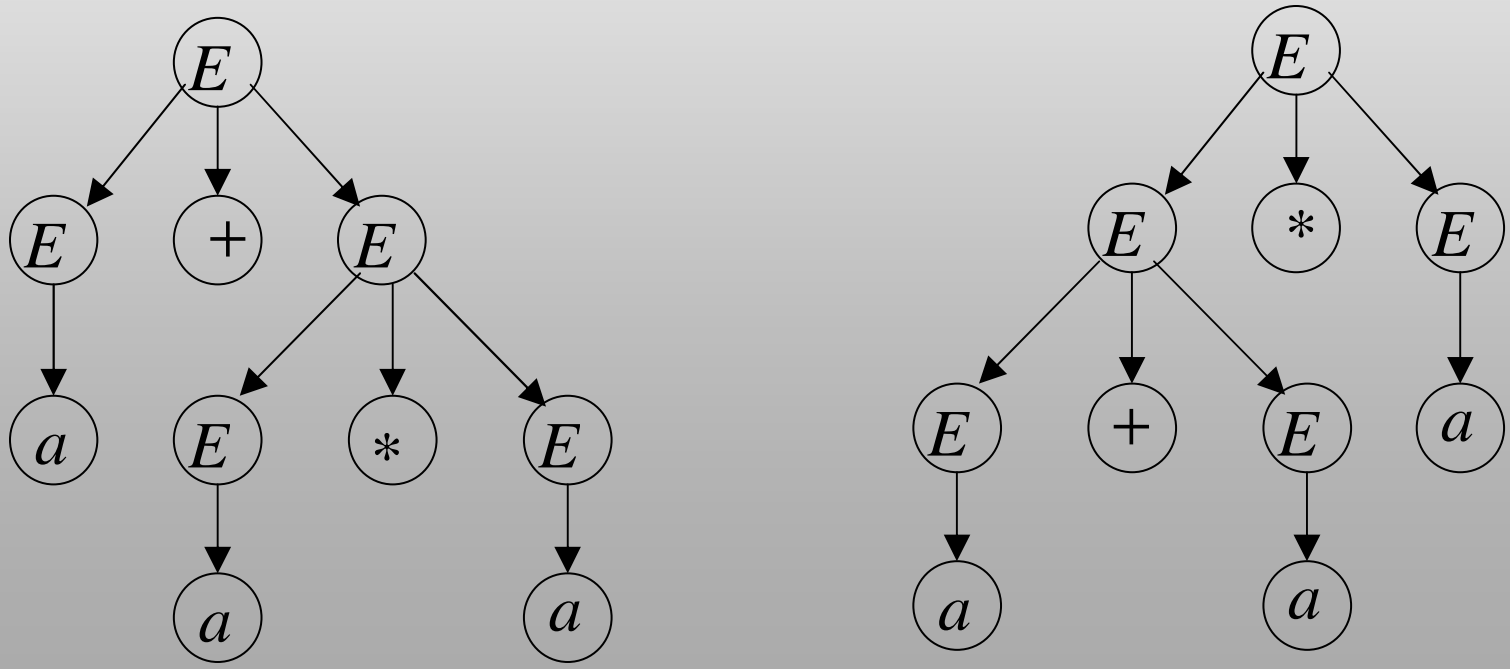
Two derivation trees



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

is ambiguous:

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



Definition:

A context-free grammar G is **ambiguous**

if some string $w \in L(G)$ has:

two or more derivation trees

In other words:

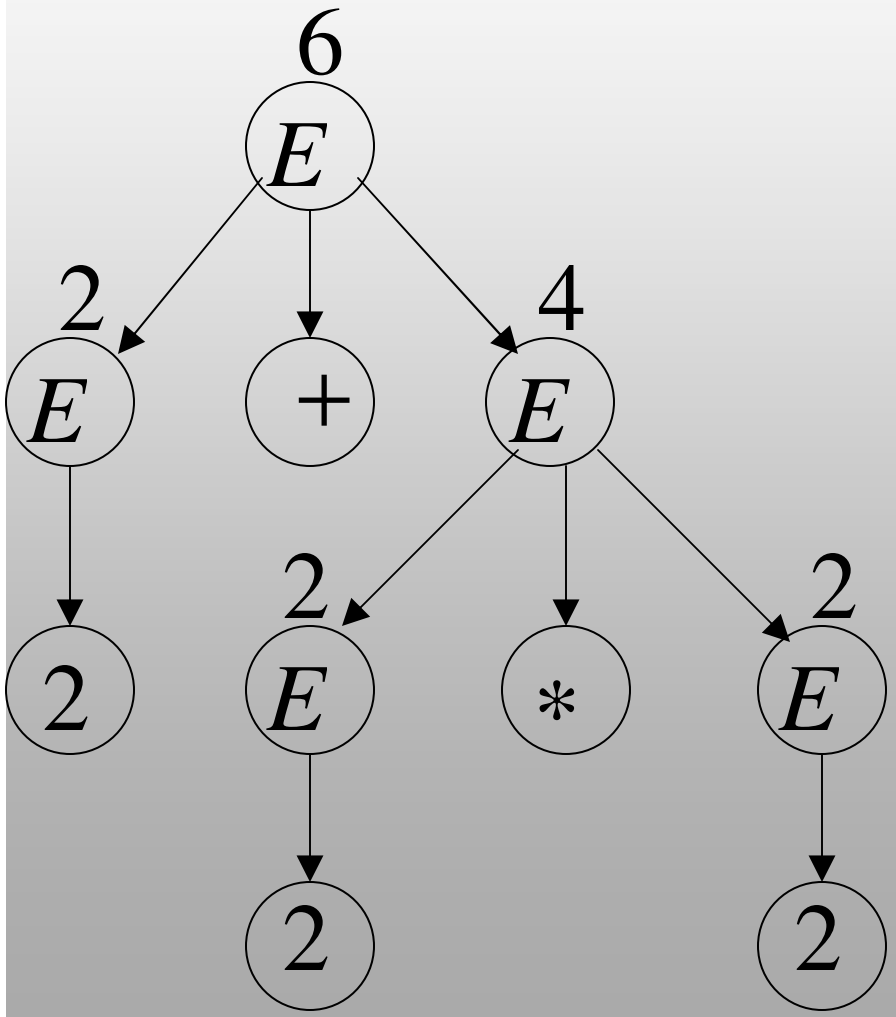
A context-free grammar G is **ambiguous**

if some string $w \in L(G)$ has:

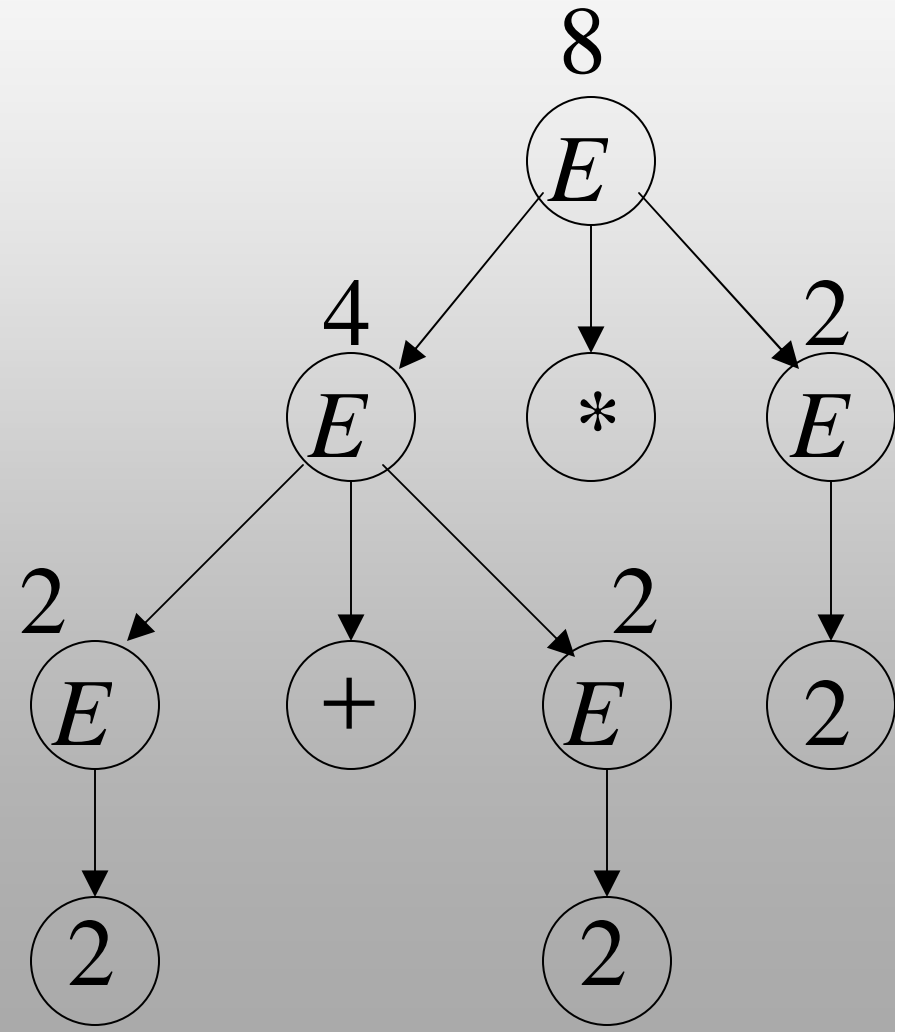
two or more leftmost derivations

(or rightmost)

$2+2*2=6$



$2+2*2=8$



- Ambiguity is **bad** for programming languages

- 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:

$$E \to E+T$$

$$E \to T$$

$$T \to T*F$$

$$T \to F$$

$$F \to (E)$$

$$F \to a$$

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

$a+a*a$

$E \rightarrow E+T$

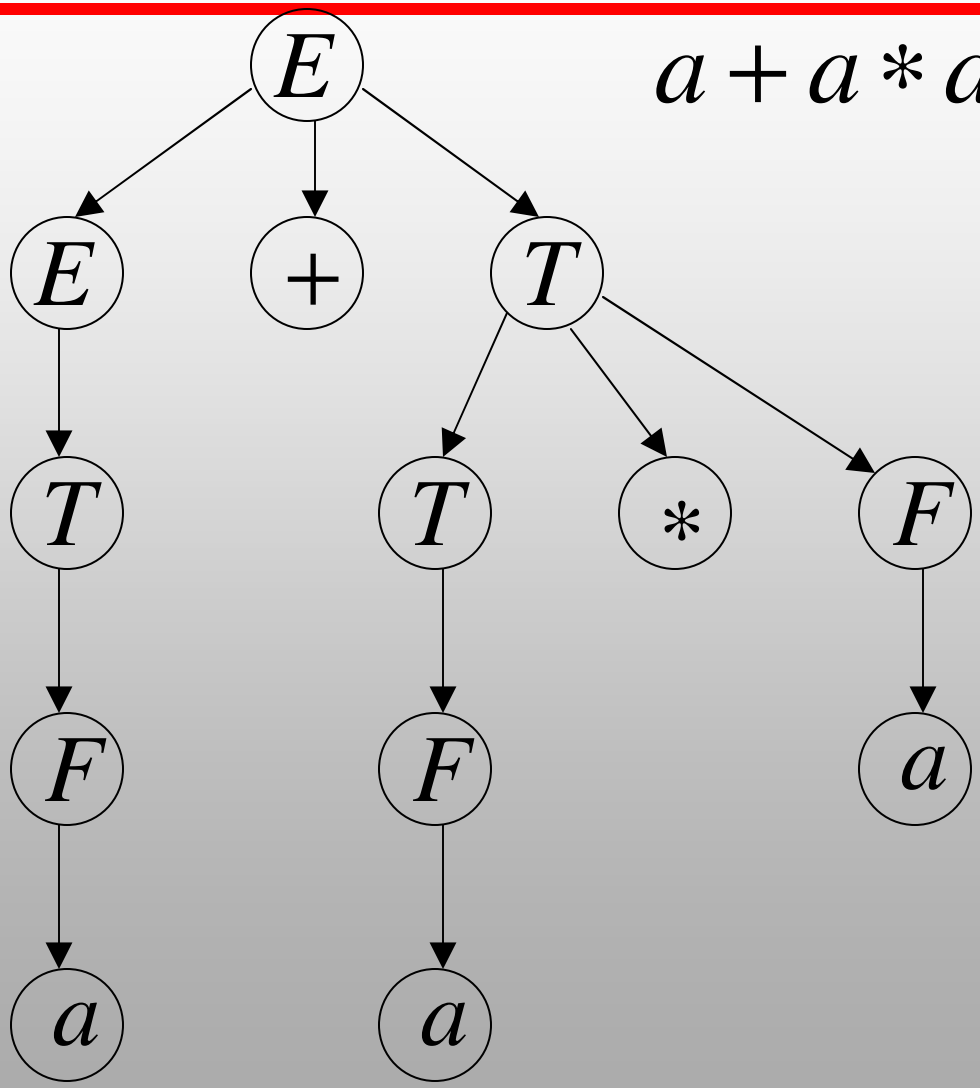
$E \rightarrow T$

$T \rightarrow T*F$

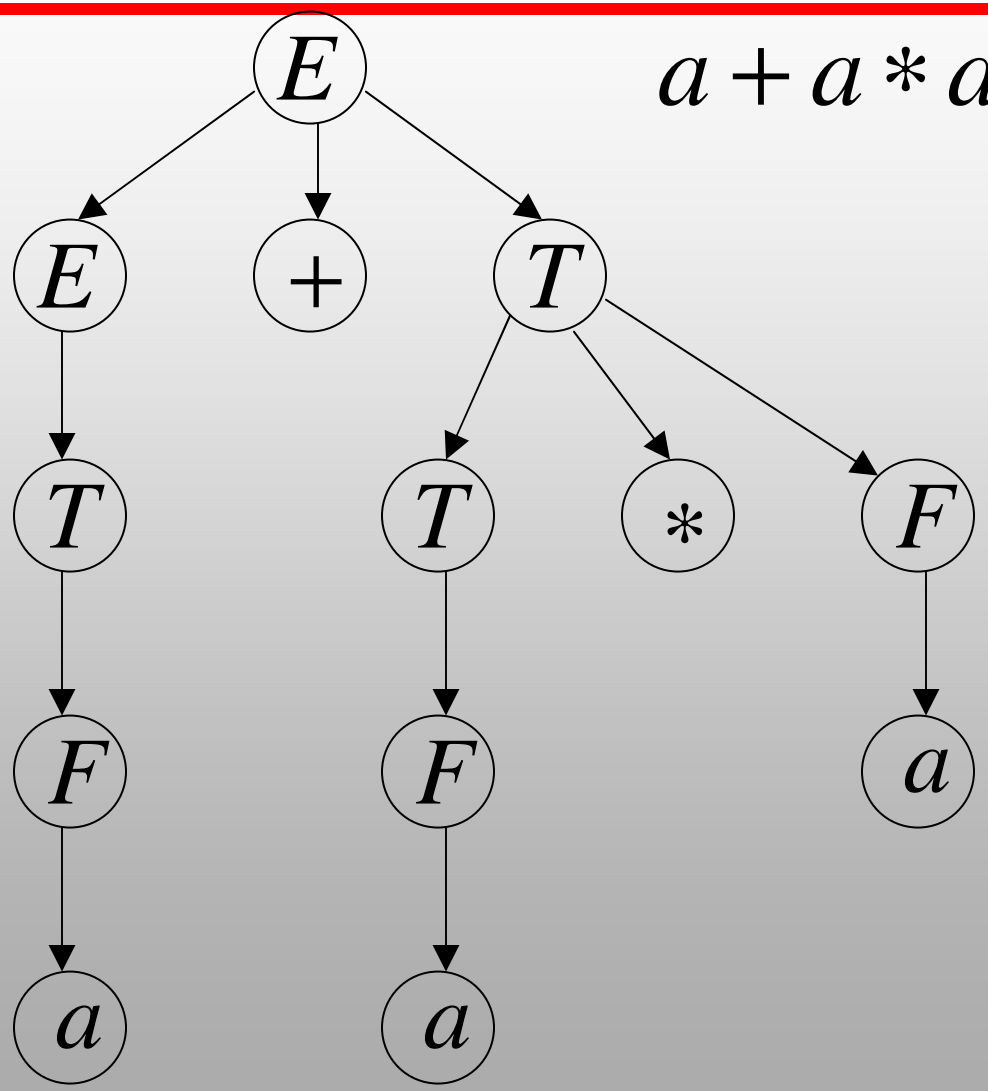
$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow a$



Unique derivation tree



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:

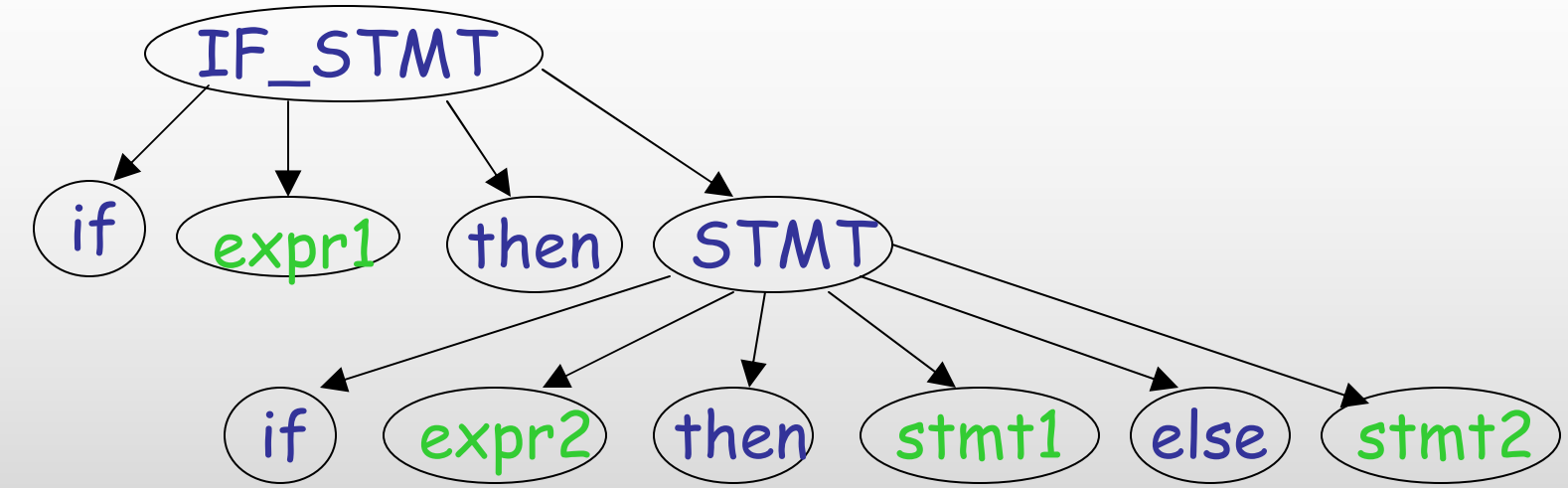
Every string $w \in L(G)$ has a unique derivation tree

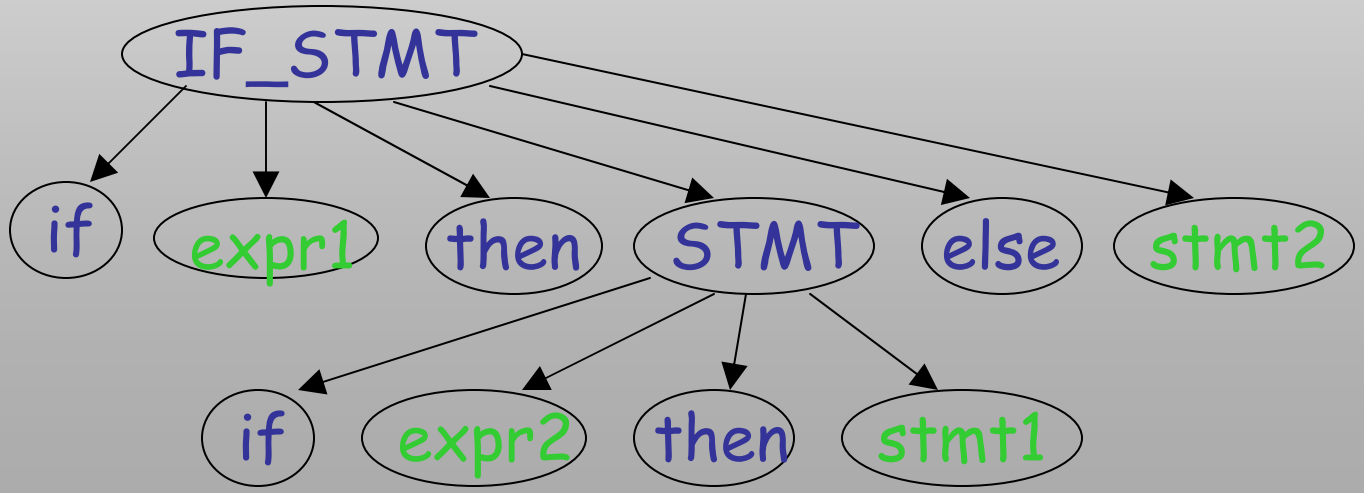
Another Ambiguous Grammar

IF_STMT → if EXPR then STMT

| if EXPR then STMT else STMT

If expr1 then if expr2 then stmt1 else stmt2





Inherent Ambiguity

- Some context free languages
- have only ambiguous grammars

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

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow S_1 c \mid A$

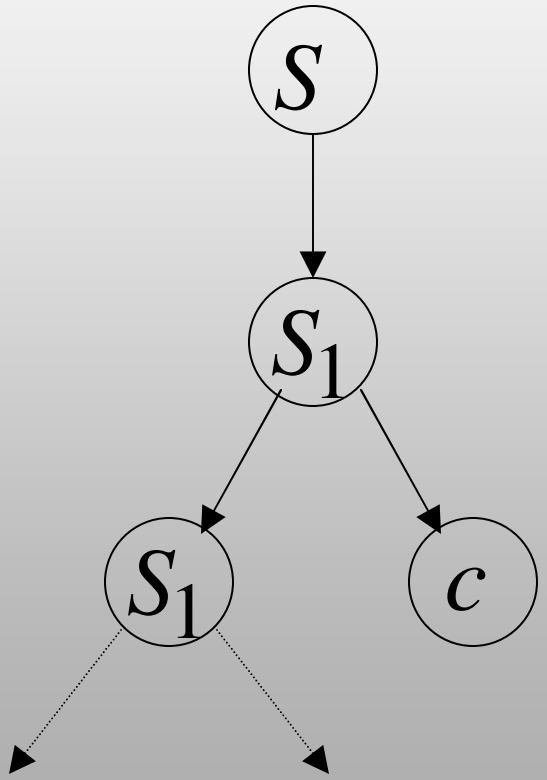
$A \rightarrow aAb \mid \lambda$

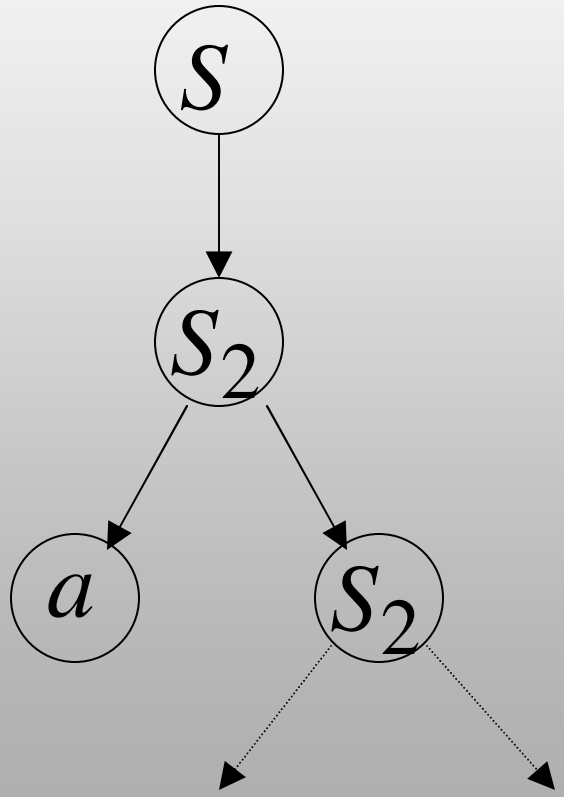
$S_2 \rightarrow aS_2 \mid B$

$B \rightarrow bBc \mid \lambda$

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

has two derivation trees





Simplification of Context Free Grammar

A Substitution Rule

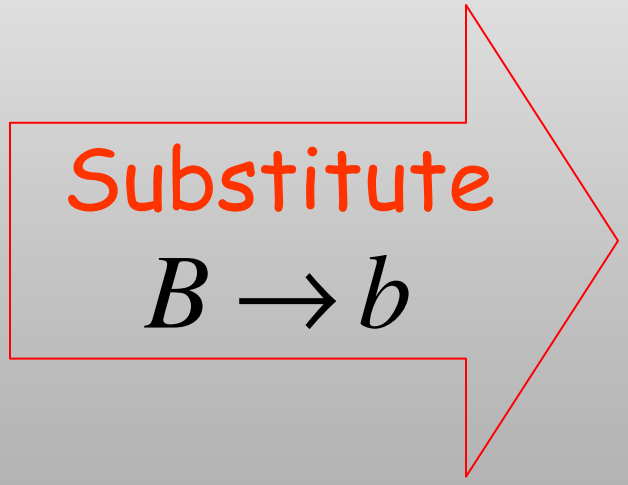
$$S \to aB$$

$$A \to aaA$$

$$A \to abBc$$

$$B \to aA$$

$$B \to b$$



Equivalent grammar

$$S \to aB \mid ab$$

$$A \to aaA$$

$$A \to abBc \mid abbc$$

$$B \to aA$$

A Substitution Rule

$$S \rightarrow aB \mid ab$$

$$A \rightarrow aaA$$

$$A \rightarrow abBc \mid abbc$$

$$B \rightarrow aA$$

Substitute

$$B \rightarrow aA$$

$$S \rightarrow \cancel{aB} \mid ab \mid aaA$$

$$A \rightarrow aaA$$

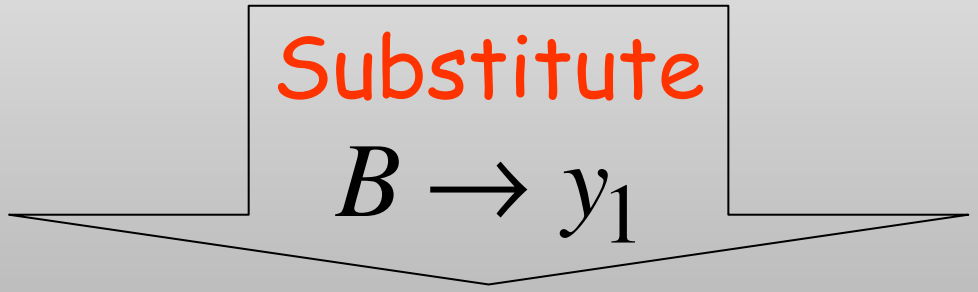
$$A \rightarrow \cancel{abBc} \mid abbc \mid abaAc$$

Equivalent grammar

In general:

$$A \to xBz$$

$$B \to y_1$$



$$A \to xBz \mid xy_1z$$

equivalent grammar

Nullable Variables

λ – production : $A \to \lambda$

Nullable Variable: $A \Rightarrow \ldots \Rightarrow \lambda$

Removing Nullable Variables

Example Grammar:

$$S \rightarrow aMb$$

$$M \rightarrow aMb$$

$$M \rightarrow \lambda$$

Nullable variable

Final Grammar

$S \rightarrow aMb$

$M \rightarrow aMb$

~~$M \rightarrow \lambda$~~

Substitute

$M \rightarrow \lambda$

$S \rightarrow aMb$

$S \rightarrow ab$

$M \rightarrow aMb$

$M \rightarrow ab$

Unit-Productions

Unit Production: $A \to B$

(a single variable in both sides)

Removing Unit Productions

Observation:

$$A \to A$$

Is removed immediately

Example Grammar:

$$S \rightarrow aA$$

$$A \rightarrow a$$

$$A \rightarrow B$$

$$B \rightarrow A$$

$$B \rightarrow bb$$

$S \rightarrow aA$

$A \rightarrow a$

~~$A \rightarrow B$~~

$B \rightarrow A$

$B \rightarrow bb$

Substitute

$A \rightarrow B$

$S \rightarrow aA \mid aB$

$A \rightarrow a$

$B \rightarrow A \mid B$

$B \rightarrow bb$

$S \rightarrow aA \mid aB$

$A \rightarrow a$

$B \rightarrow A \mid B$

$B \rightarrow bb$

Remove

$B \rightarrow B$

$S \rightarrow aA \mid aB$

$A \rightarrow a$

$B \rightarrow A$

$B \rightarrow bb$

$S \rightarrow aA \mid aB$

$A \rightarrow a$

~~$B \rightarrow A$~~

$B \rightarrow bb$

Substitute

$B \rightarrow A$

$S \rightarrow aA \mid aB \mid aA$

$A \rightarrow a$

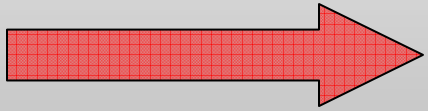
$B \rightarrow bb$

Remove repeated productions

$$S \rightarrow aA \mid aB \mid \cancel{aA}$$

$$A \rightarrow a$$

$$B \rightarrow bb$$



Final grammar

$$S \rightarrow aA \mid aB$$

$$A \rightarrow a$$

$$B \rightarrow bb$$

Useless Productions

$$S \to aSb$$

$$S \to \lambda$$

$$S \to A$$

$$A \to aA$$

Useless Production

Some derivations never terminate…

$$S \Rightarrow A \Rightarrow aA \Rightarrow aaA \Rightarrow \ldots \Rightarrow aa\ldots aA \Rightarrow \ldots$$

Another grammar:

$$S \rightarrow A$$

$$A \rightarrow aA$$

$$A \rightarrow \lambda$$

$$B \rightarrow bA$$

Useless Production

Not reachable from S

In general:

contains only terminals

if $S \Rightarrow \ldots \Rightarrow xAy \Rightarrow \ldots \Rightarrow w$

$w \in L(G)$

then variable A is useful

otherwise, variable A is useless

A production $A \to x$ is useless
if any of its variables is useless

$S \to aSb$

$S \to \lambda$

Variables		Productions
	$S \to A$	useless
useless	$A \to aA$	useless
useless	$B \to C$	useless
useless	$C \to D$	useless

Removing Useless Productions

Example Grammar:

$$S \rightarrow aS \mid A \mid C$$

$$A \rightarrow a$$

$$B \rightarrow aa$$

$$C \rightarrow aCb$$

First: find all variables that can produce strings with only terminals

$S \to aS \mid A \mid C$

$A \to a$

$B \to aa$

$C \to aCb$

Round 1: $\{A, B\}$

$S \to A$

Round 2: $\{A, B, S\}$

Keep only the variables
that produce terminal symbols: $\{A, B, S\}$

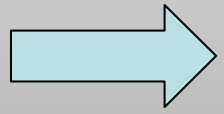
(the rest variables are useless)

$S \rightarrow aS \mid A \mid C$

$A \rightarrow a$

$B \rightarrow aa$

$C \rightarrow aCb$



$S \rightarrow aS \mid A$

$A \rightarrow a$

$B \rightarrow aa$

Remove useless productions

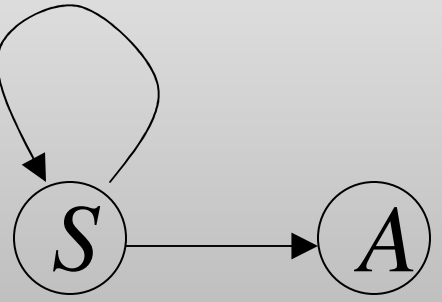
Second: Find all variables reachable from S

$$S \rightarrow aS \mid A$$

$$A \rightarrow a$$

$$B \rightarrow aa$$

Use a Dependency Graph



not reachable

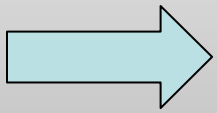
Keep only the variables reachable from S

(the rest variables are useless)

$S \rightarrow aS \mid A$

$A \rightarrow a$

~~$B \rightarrow aa$~~



Final Grammar

$S \rightarrow aS \mid A$

$A \rightarrow a$

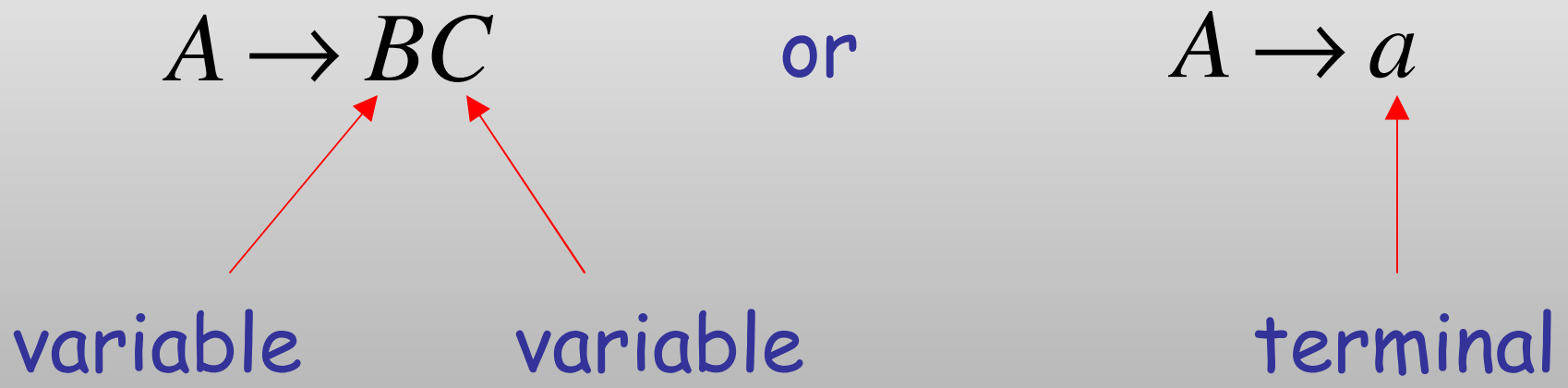
Remove useless productions

Removing All

- Step 1: Remove Nullable Variables
- Step 2: Remove Unit-Productions
- Step 3: Remove Useless Variables

Chomsky Normal Form for CFG

Each productions has form:



Examples:

$S \to AS$	$S \to AS$
$S \to a$	$S \to AAS$
$A \to SA$	$A \to SA$
$A \to b$	$A \to aa$
Chomsky Normal Form	Not Chomsky Normal Form

Convertion to Chomsky Normal Form

- Example:

$$S \rightarrow ABa$$

$$A \rightarrow aab$$

$$B \rightarrow Ac$$

Not Chomsky Normal Form

Introduce variables for terminals: T_a, T_b, T_c

$$S \to ABa$$
$$A \to aab$$
$$B \to Ac$$

$$S \to ABT_a$$
$$A \to T_a T_a T_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

Introduce intermediate variable: V_1

$$S \to ABT_a$$
$$A \to T_a T_a T_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

$$S \to AV_1$$
$$V_1 \to BT_a$$
$$A \to T_a T_a T_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

Introduce intermediate variable: V_2

$$S \to AV_1$$
$$V_1 \to BT_a$$
$$A \to T_aT_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

$$S \to AV_1$$
$$V_1 \to BT_a$$
$$A \to T_aV_2$$
$$V_2 \to T_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

Final grammar in Chomsky Normal Form:

Initial grammar

$$S \to ABa$$

$$A \to aab$$

$$B \to Ac$$

$$S \to AV_1$$

$$V_1 \to BT_a$$

$$A \to T_aV_2$$

$$V_2 \to T_aT_b$$

$$B \to AT_c$$

$$T_a \to a$$

$$T_b \to b$$

$$T_c \to c$$

In general:

From any context-free grammar
(which doesn't produce λ)
not in Chomsky Normal Form

we can obtain:

An equivalent grammar
in Chomsky Normal Form

The Procedure

First remove:

Nullable variables

Unit productions

Then, for every symbol a :

Add production $T_a \to a$

In productions: replace a with T_a

New variable: T_a

Replace any production $A \to C_1 C_2 \cdots C_n$

with

$$A \to C_1 V_1$$

$$V_1 \to C_2 V_2$$

$$\cdots$$

$$V_{n-2} \to C_{n-1} C_n$$

New intermediate variables: $V_1, V_2, \ldots, V_{n-2}$

Observations

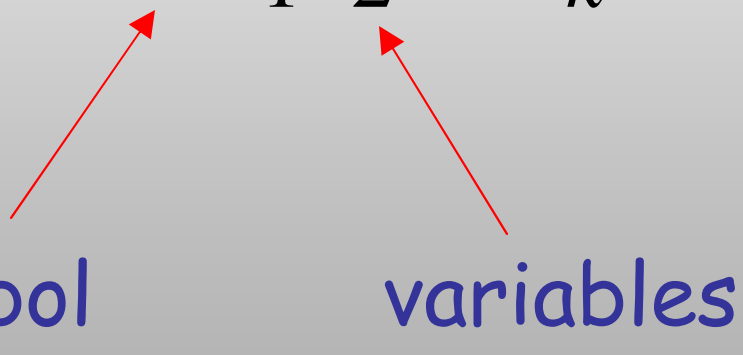
- Chomsky normal forms are good for parsing and proving theorems

- It is very easy to find the Chomsky normal form for any context-free grammar

Greinbach Normal Form

All productions have form:

$$A \rightarrow a\, V_1 V_2 \cdots V_k \qquad k \geq 0$$



Examples:

$$S \rightarrow cAB$$

$$A \rightarrow aA \mid bB \mid b$$

$$B \rightarrow b$$

Greinbach
Normal Form

$$S \rightarrow abSb$$

$$S \rightarrow aa$$

Not Greinbach
Normal Form

Observations

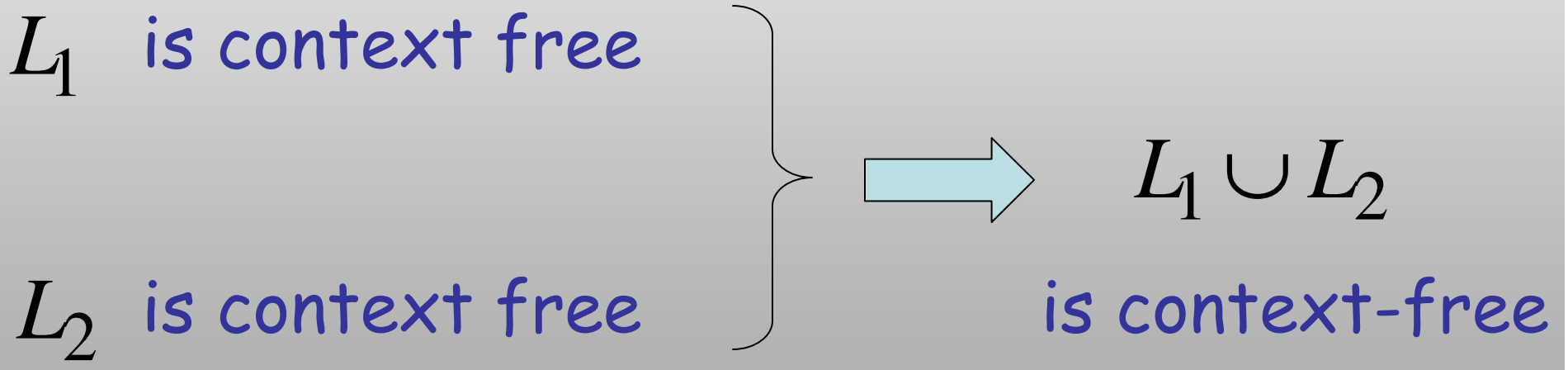
- Greinbach normal forms are very good for parsing

- It is hard to find the Greinbach normal form of any context-free grammar

Properties of CFL

Union

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



Example

Language

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

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

Grammar

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

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

Union

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

$$S \to S_1 \mid S_2$$

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

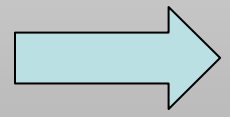
and additional production $S \to S_1 \mid S_2$

Concatenation

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

L_1 is context free

L_2 is context free



$L_1 L_2$ is context-free

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 $S \to S_1S_2$

Star Operation

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

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

Example

Language	Grammar
$L = \{a^n b^n\}$	$S \rightarrow aSb \mid \lambda$

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

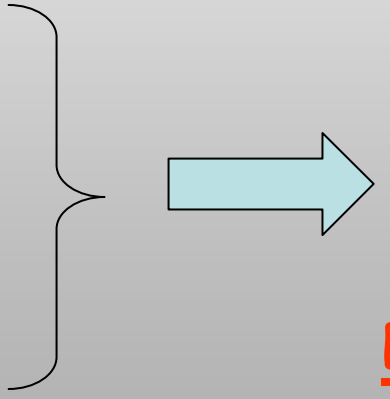
and additional production $S_1 \rightarrow SS_1 \mid \lambda$

Intersection

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

L_1 is context free

L_2 is context free



$L_1 \cap L_2$

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:

$$S \to AB$$

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

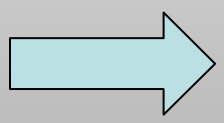
$$B \to bBc \mid \lambda$$

Intersection

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

Complement

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

L is context free  $\overline{L}$ **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:

$$S \to AB$$

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

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

Complement

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

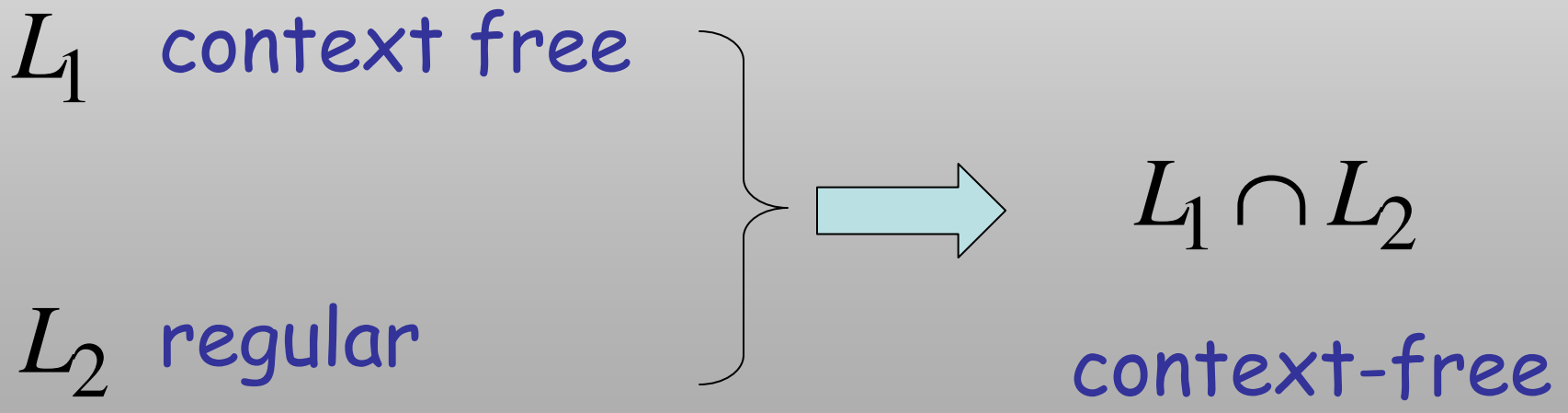
NOT context-free

The intersection of

a context-free language and

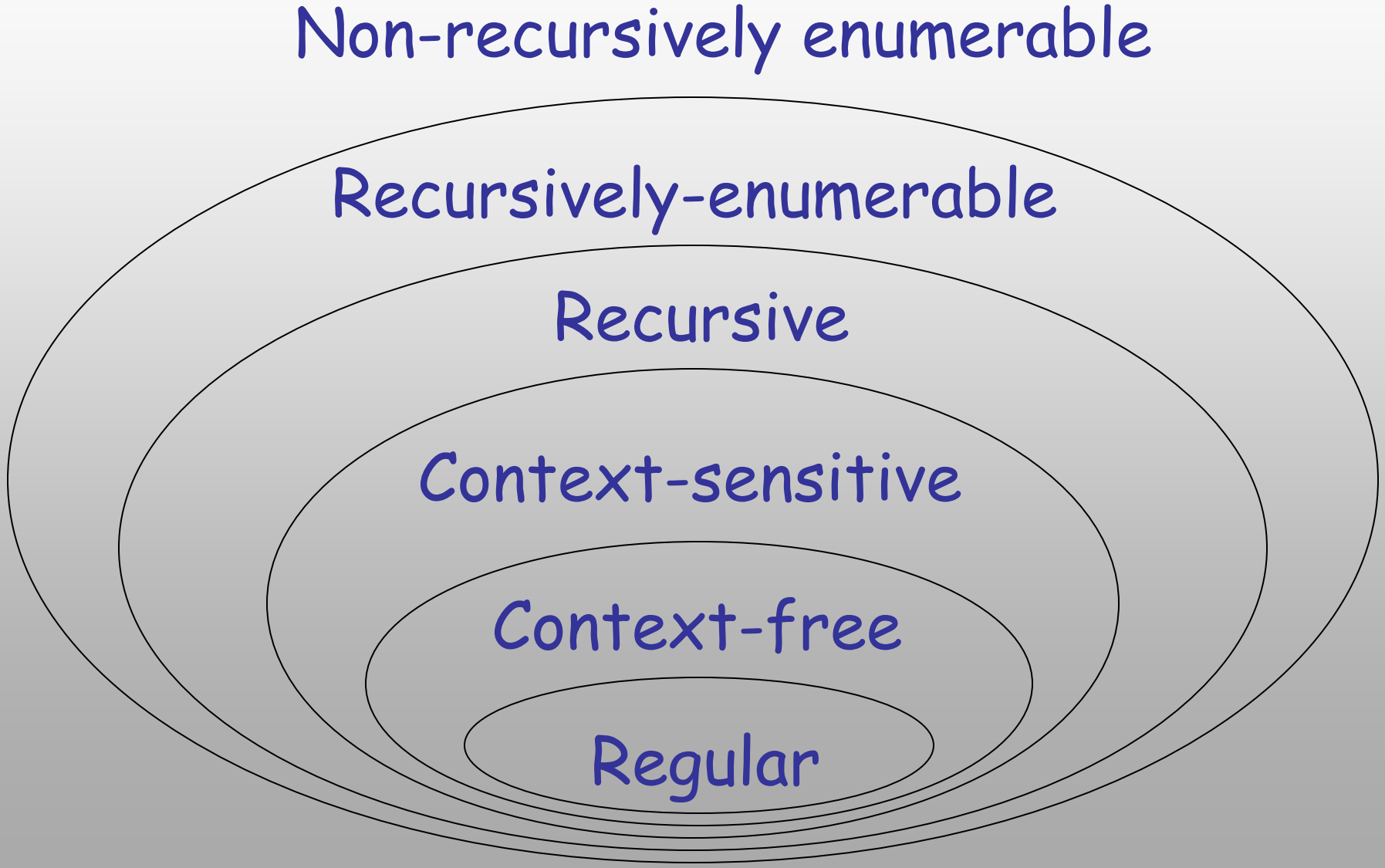
a regular language

is a context-free language



Summary

The Chomsky Hierarchy



Who is Noam Chomsky Anyway?

- Philosopher of Languages
- Professor of Linguistics at MIT
- Constructed the idea that language was not a learned “behavior”, but that it was cognitive and innate; versus stimulus-response driven
- In an effort to explain these theories, he developed the Chomsky Hierarchy

Chomsky Hierarchy

Language	Grammar	Machine	Example
Regular Language	Regular Grammar • Right-linear grammar • Left-linear grammar	Deterministic or Nondeterministic Finite-state acceptor	a^*
Context-free Language	Context-free grammar	Nondeterministic Pushdown automaton	$a^n b^n$
Context-sensitive	Context-sensitive grammar	Linear-bounded automaton	$a^n b^n c^n$
Recursively enumerable	Unrestricted grammar	Turing machine	Any computable function

Chomsky Hierarchy

- Comprises four types of languages and their associated grammars and machines.
- Type 3: Regular Languages
- Type 2: Context-Free Languages
- Type 1: Context-Sensitive Languages
- Type 0: Recursively Enumerable Languages
- These languages form a strict hierarchy