Menoufia University

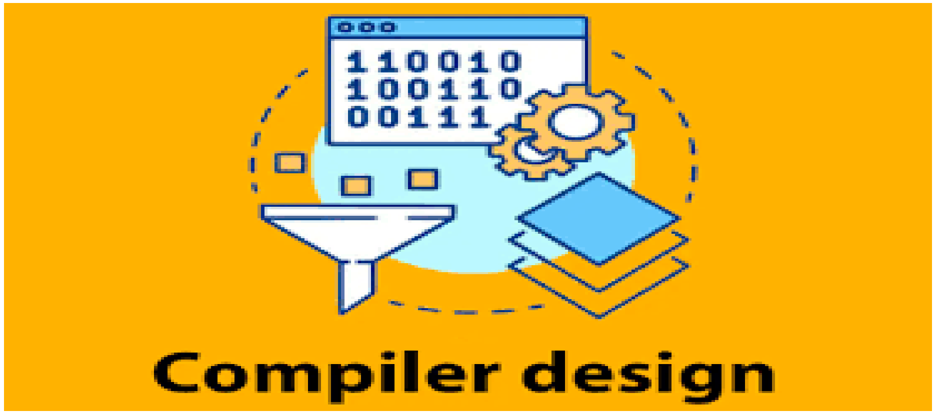
Faculty of computers & Information

Computer Science Department.

Compiler Design

4 Year – first Semester

Lecture 3



DR. Eman Meslhy Mohamed

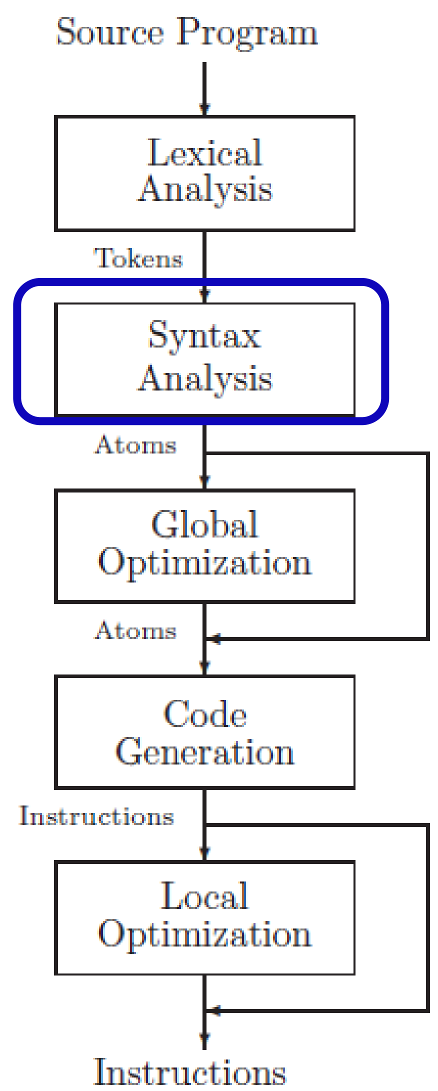
Lecturer at Computer Science department

2023-2024

Phases of Compilers

Phases of Compilers

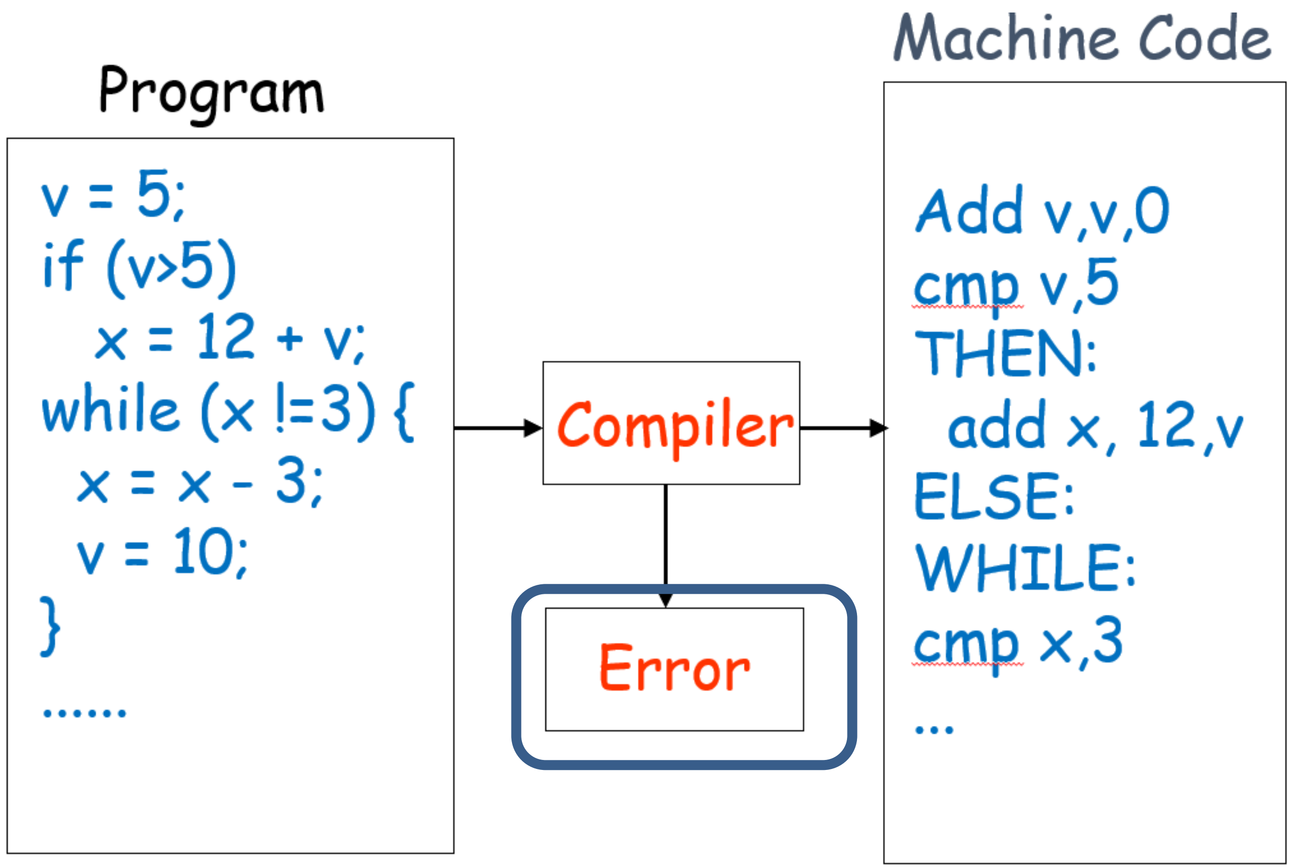
- Lexical Analysis (Scanner)
- Syntax Analysis Phase
- Global Optimization
- Code Generation
- Local Optimization



Outline

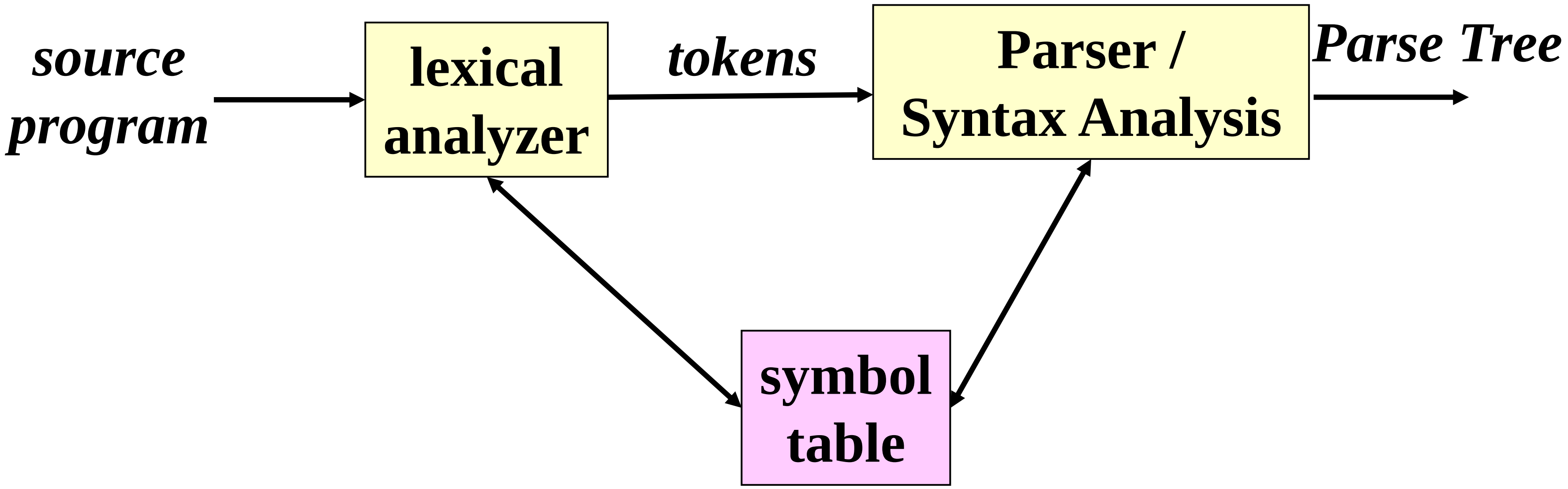
- **Syntax Analysis.**
- Grammar.
- Pushdown Machine.
- Parser.

The Role of a Syntax Analyzer



Syntax Analyzer is used to check for the **prober syntax** and generate the **syntax tree**.

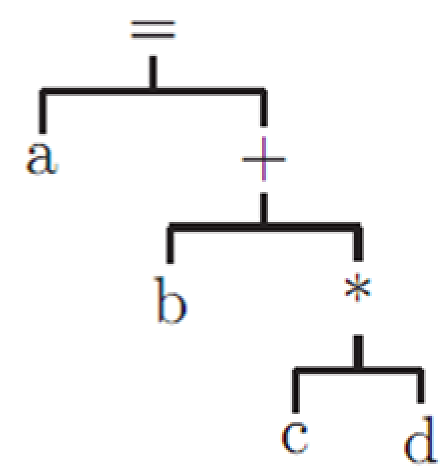
The Role of a Syntax Analyzer



Parse (Syntax) Tree

Syntax Tree is a data structure in which the interior nodes represents operations and leaves represent operands.

$$a = b + c * d$$



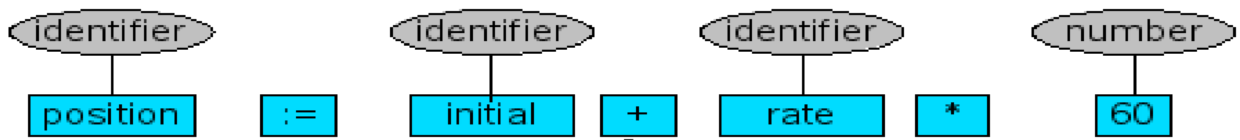
Example

source program

position = initial + rate * 60

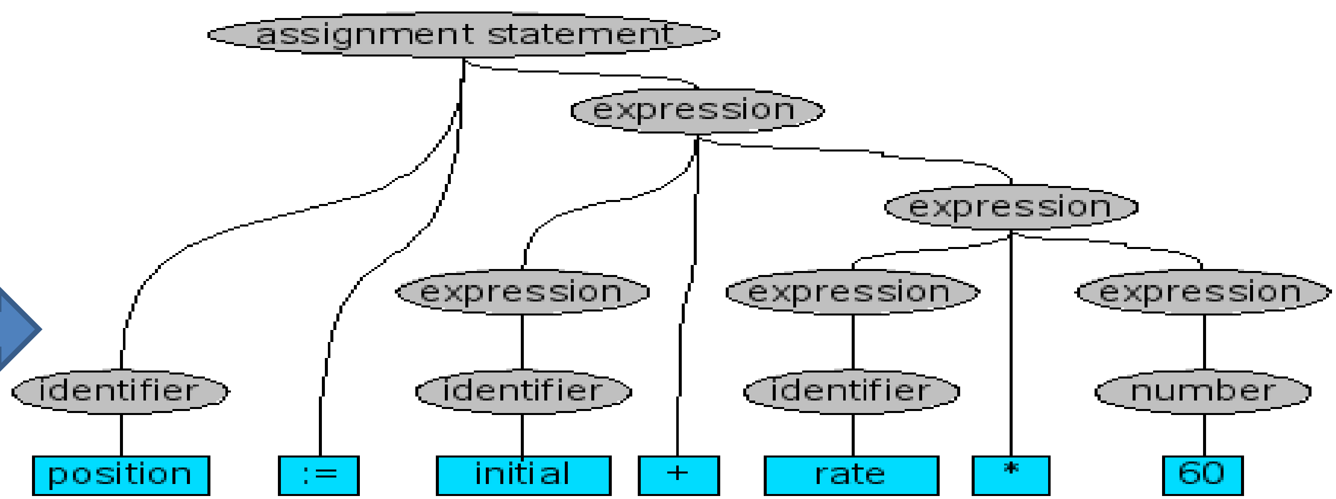
Lexical Analysis

Tokens



Syntax Analysis

Parse Tress



Syntax Analysis

Before getting into syntax analysis, we need to cover the concepts of formal **grammar** and **pushdown machine** which are critical to the design of the lexical analyzer.

Outline

- Syntax Analysis.
- **Grammar.**
- Pushdown Machine.
- Parser.

Grammar

Grammar is a list of rules which can be used to describe the structure or syntax of a language. (i.e., The grammar of a language defines the correct form for sentences in that language.)

Example: English language:

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

$$\langle noun \rangle \to \langle article \rangle \ \langle noun \rangle$$

$$\langle article \rangle \to a$$

$$\langle article \rangle \to the$$

$$\langle noun \rangle \to cat$$

$$\langle noun \rangle \to dog$$

$$\langle verb \rangle \to runs$$

$$\langle verb \rangle \to walks$$

✓ Derivation of "**the dog walks**"

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

Grammar

A **Grammar** is a list of rules which can be used to describes the structure or syntax of a language. (i.e., <u>The grammar of a language defines the correct form for sentences in that language</u>.)

Example: English language:

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

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

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

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

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

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

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

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

✓ Derivation of "**a cat runs**"

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

Grammar

A **Grammar** is a list of rules which can be used to describes the structure or syntax of a language. (i.e., <u>The grammar of a language defines the correct form for sentences in that language</u>.)

Example: English language:

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

$$\langle noun\rangle \to \langle article\rangle\ \langle noun\rangle$$

$$\langle article\rangle \to a$$

$$\langle article\rangle \to the$$

$$\langle noun\rangle \to cat$$

$$\langle noun\rangle \to dog$$

$$\langle verb\rangle \to runs$$

$$\langle verb\rangle \to walks$$

✓ Derivation of "**dog the runs**"

Error

Grammar Components

A **Grammar** is denoted by G and is defined as a **4–tuple** i.e., **G (V, T, S, P)**

Where

V is non empty set of symbols called as **Variables**

T is non empty set of symbols called as **Terminals**

S $\in$**V** is a **Start Variable**

P is set of **productions** or **production rules.**

Grammar

Notation:

- **Variables** are denoted by only UPPER CASE letters and some special Greek letters etc. Variables are also called a Non-Terminals.
- **Terminals** are denoted by lower case letters i.e., **a to z** and digits 0 **to 9** and some special operators like arithmetic operators, relational operators etc.

Production Form

Productions is defined as **mapping function.**

General form of Productions:

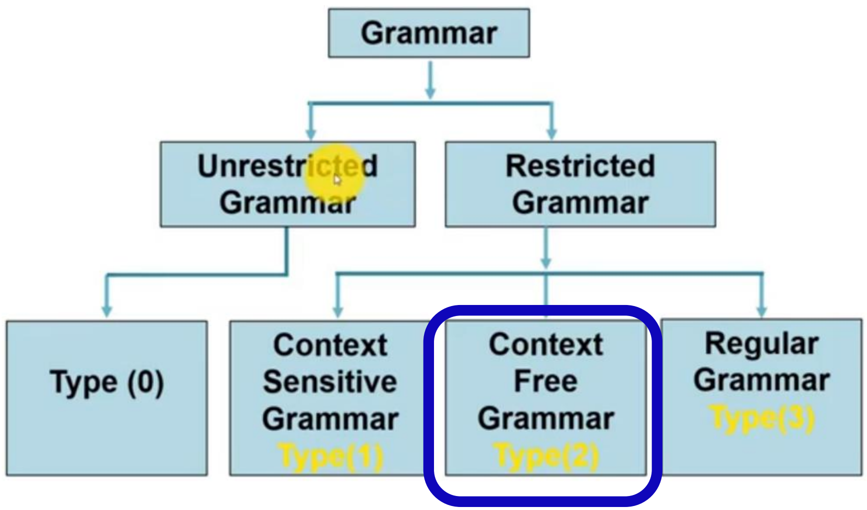
$$P : \alpha \rightarrow \beta$$

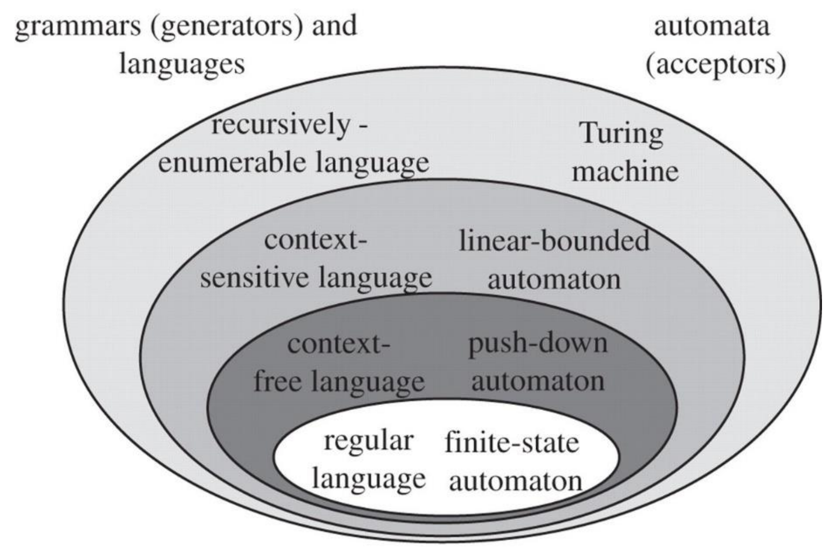
$$\alpha \in (V \cup T)^{+}$$

$$\beta \in (V \cup T)^{*}$$

Types of Grammar

Chomsky Hierarchy





the traditional Chomsky hierarchy

Examples

Example-1:

Let G = (V, T, S, P) is a grammar.

Where

V = {S, A, B}

T = {a, b}

S is a Start Variable

And Productions P are given below:

$$S \rightarrow ASB$$

$$A \rightarrow aSb \mid \varepsilon$$

$$B \rightarrow bSa \mid \varepsilon$$

ε or λ is Null String

Examples

Example-2:

Let $G = (V, T, S, P)$ is a grammar.

Where

$V = \{S, A, B\}$

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

S is a Start Variable

And Productions P are given below:

$$\boxed{\begin{aligned} Sa &\rightarrow ASB \\ Sb &\rightarrow aSb \mid \varepsilon \\ BA &\rightarrow bSB \mid A+B \end{aligned}}$$

ε or λ is Null String

Derivation of grammar

The grammar specifies a language in the following way:

Beginning with the starting nonterminal, any of the rewriting rules are applied repeatedly to produce a sentential form, which may contain a mix of terminals and nonterminals.

A **derivation** is a sequence of rewriting rules, applied to the starting nonterminal, ending with a string of terminals.

Examples

Example-3:

$$S \rightarrow aSb$$

$$S \rightarrow \lambda$$

- Derivation of string **ab**:

$$S \underset{1}{\Rightarrow} aSb \underset{2}{\Rightarrow} ab$$

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

Examples

Example-3:

$$S \to aSb$$

$$S \to \lambda$$

- Derivation of string **aabb**:

$$S \underset{1}{\Rightarrow} aSb \underset{1}{\Rightarrow} aaSbb \underset{2}{\Rightarrow} aabb$$

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

Grammar

Examples: Describe the language of this grammar

Example-3:

$$S \to aSb$$

$$S \to \lambda$$

- Derivation of string **aaabbb**:

$$S \underset{1}{\Rightarrow} aSb \underset{1}{\Rightarrow} aaSbb \underset{1}{\Rightarrow} aaaSbbb \underset{2}{\Rightarrow} aaabbb$$

- Derivation of string **aaaabbbb**:

$$S \overset{1}{\Rightarrow} aSb \overset{1}{\Rightarrow} aaSbb \overset{1}{\Rightarrow} aaaSbbb$$

$$\overset{1}{\Rightarrow} aaaaSbbbb \overset{2}{\Rightarrow} aaaabbbb$$

Language of the grammar

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

Grammar

Examples: Describe the language of this grammar

Example-4:

$$S \to Ab$$

$$A \to aAb$$

$$A \to \lambda$$

Derivations:

$$S \to Ab \to b$$

$$S \to Ab \to aAbb \to abb$$

$$S \to Ab \to aAbb \to aaAbbb \to aabbb$$

Language of the grammar

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

Examples: Describe the language of this grammar

Example-5:

$$S \to 0S0$$

$$S \to 1S1$$

$$S \to 0$$

$$S \to 1$$

Derivations:

$$S \to 0S0 \to 000$$

$$S \to 0S0 \to 01S10 \to 01010$$

Language of the grammar

Palindromes of **odd** length over the alphabet $\{0,1\}$

Examples: Describe the language of this grammar

Example-6:

$$S \rightarrow (S)$$
$$S \rightarrow \lambda$$

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

Describes parentheses:

((( )))

$$S \rightarrow (S)$$
$$S \rightarrow SS$$
$$S \rightarrow \lambda$$

Describes parentheses:

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

Derivation Order

There are two type of derivations which are:

- **Leftmost derivation** is one in which the **left-most nonterminal** is always the one to which a rule is applied.
- **Rightmost derivation** is one in which the **right-most nonterminal** is always the one to which a rule is applied.

Derivation Order

Example:

$$S \rightarrow aAB$$

$$A \rightarrow bBb$$

$$B \rightarrow A \mid \lambda$$

Leftmost derivation:

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

$$\Rightarrow abbbbB \Rightarrow abbbb$$

Rightmost derivation:

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

$$\Rightarrow abbBbb \Rightarrow abbbb$$

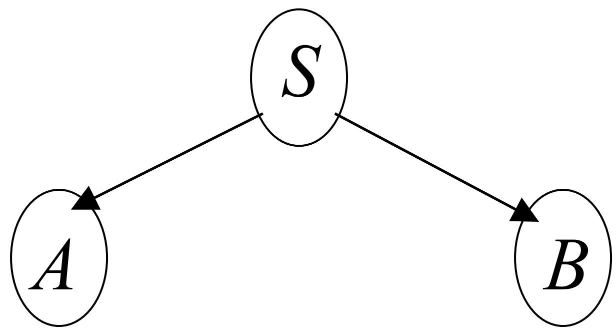
Derivation Tree

Derivation Tree is a tree in which each **interior node** corresponds to a **nonterminal** in a sentential form and each **leaf node** corresponds to a **terminal** symbol in the derived string.

Derivation Tree for aab

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

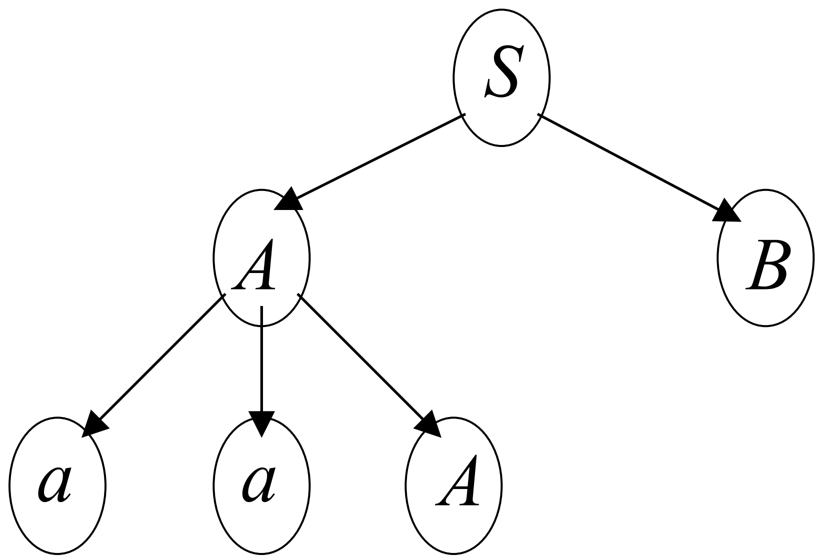
$$S \Rightarrow AB$$



Derivation Tree for aab

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

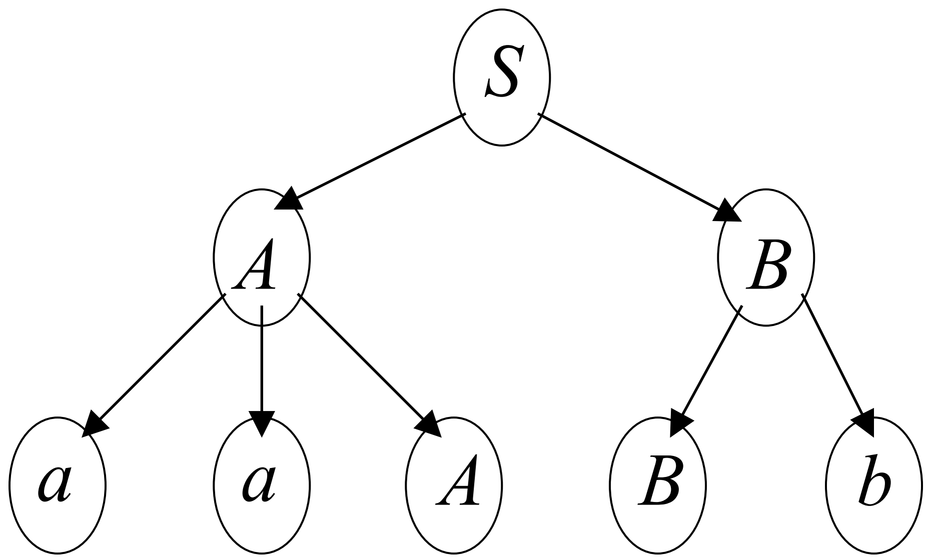
$$S \Rightarrow AB \Rightarrow aaAB$$



Derivation Tree for aab

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

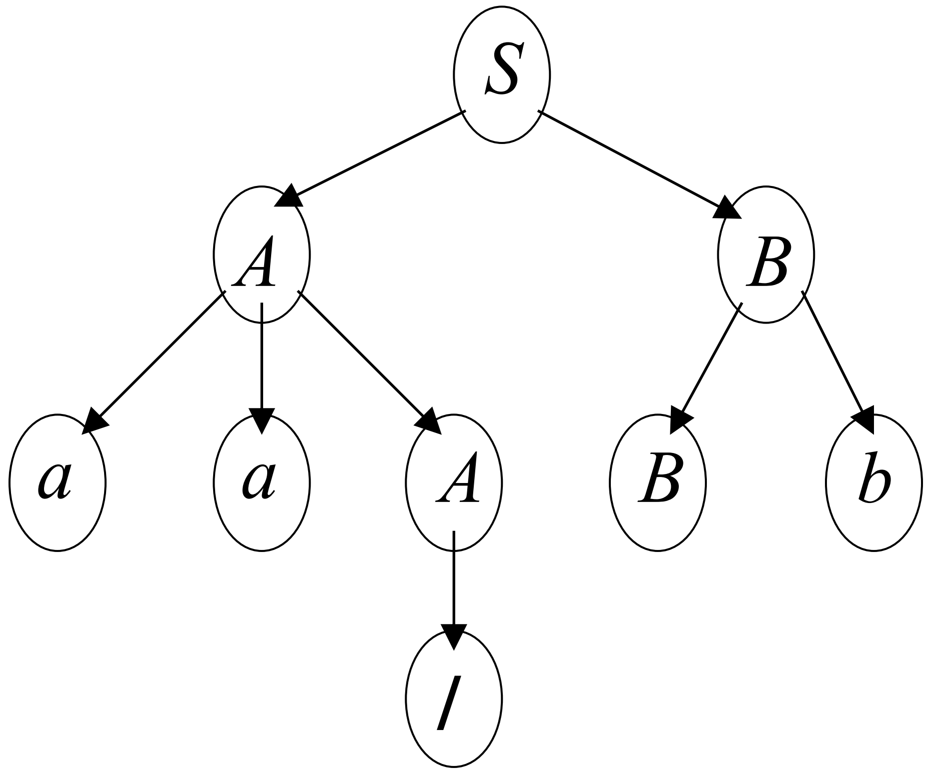
$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb$$



Derivation Tree for aab

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

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

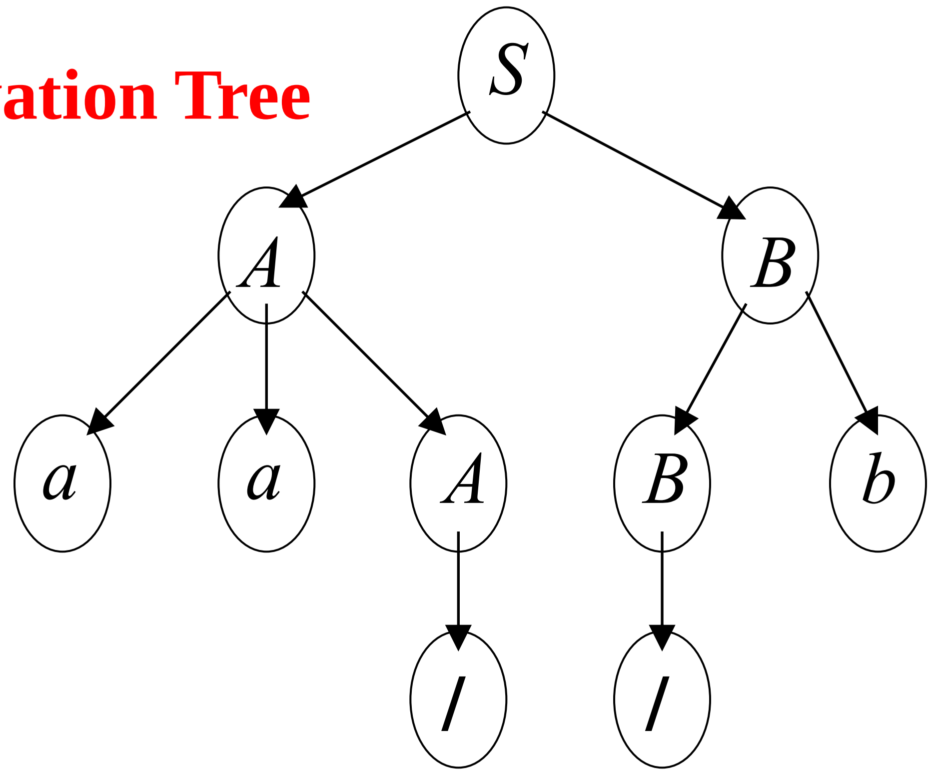


Derivation Tree for aab

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

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

Derivation Tree

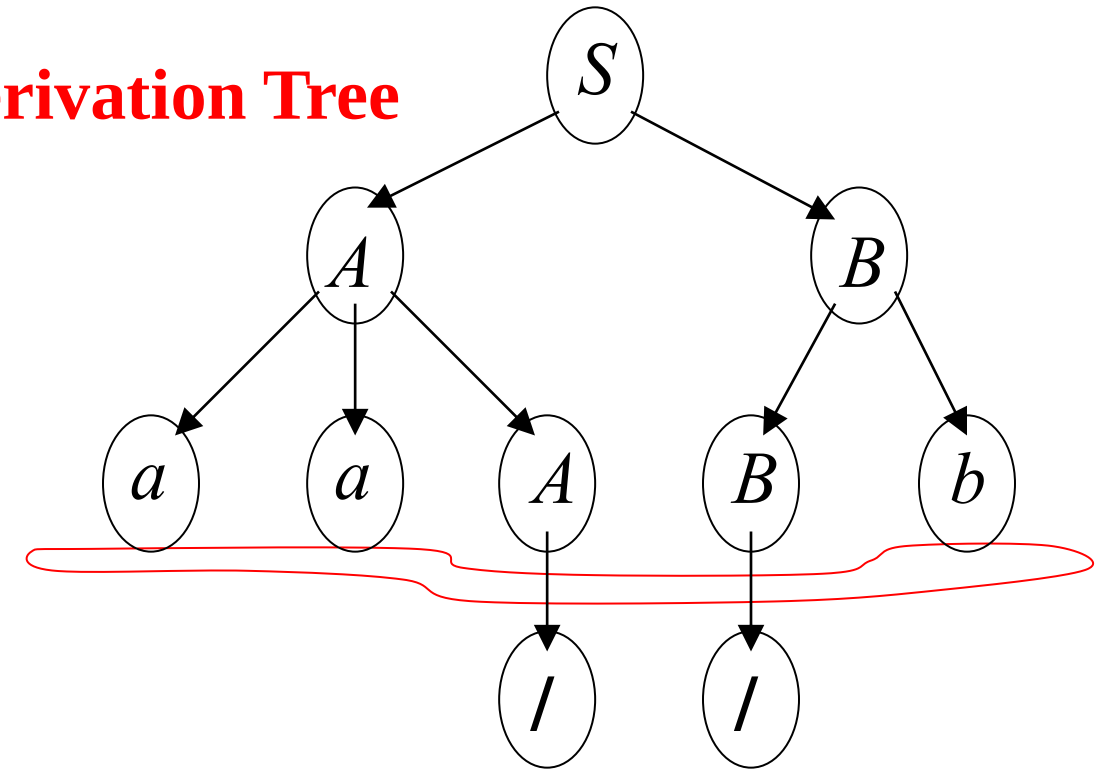


Derivation Tree for aab

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

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

Derivation Tree



yield

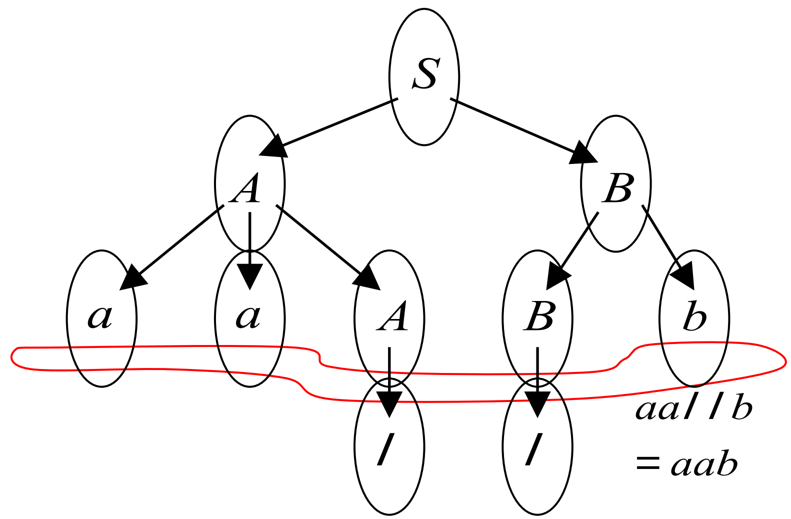
$aa\lambda\lambda b$

$= aab$

Derivation Tree for aab

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

Derivation Tree



Leftmost:

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

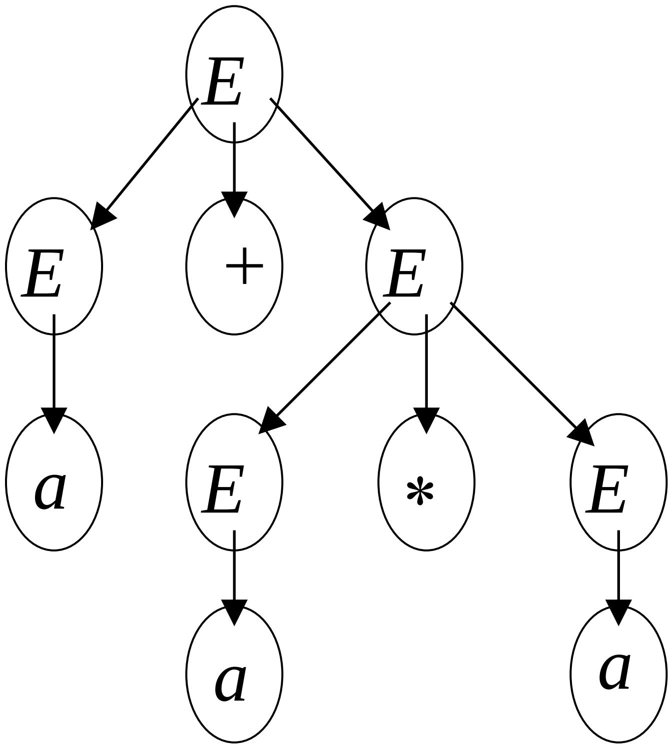
Rightmost:

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

Ambiguity

A context-free grammar is said to be **ambiguous** if there is **more than one derivation tree for a particular string**.

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



A leftmost derivation for

$$\boxed{a + a * a}$$

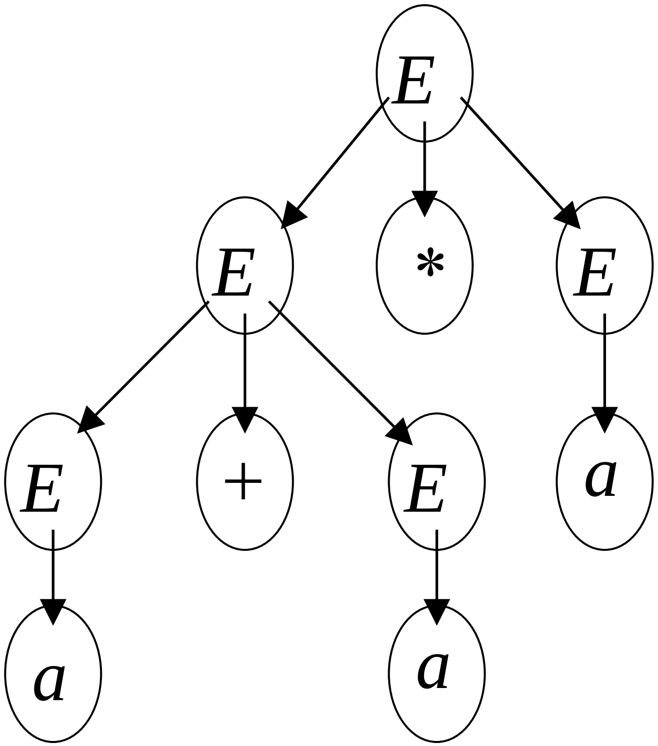
$$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E$$

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

Ambiguity

A context-free grammar is said to be **ambiguous** if there is **more than one derivation tree for a particular string**.

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



Another leftmost derivation for

$$\boxed{a + a * a}$$

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

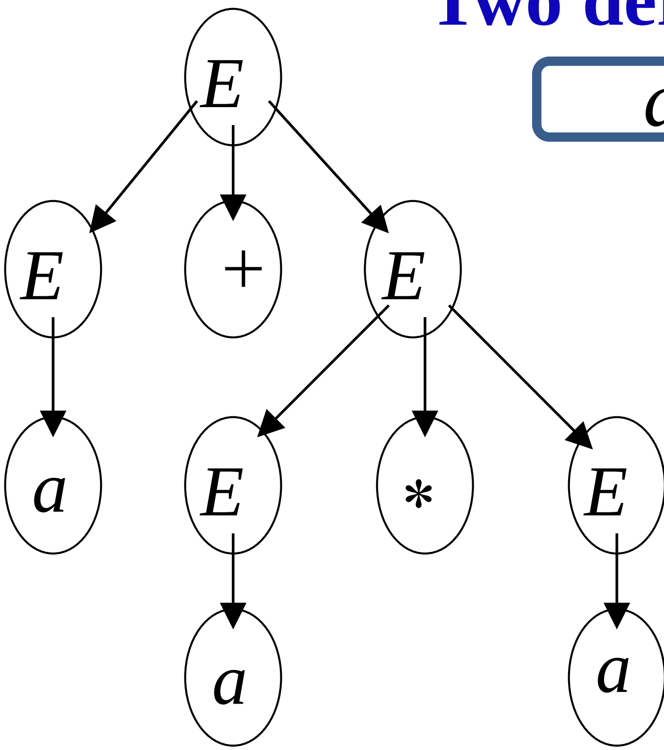
$$\Rightarrow a + a * E \Rightarrow a + a * a$$

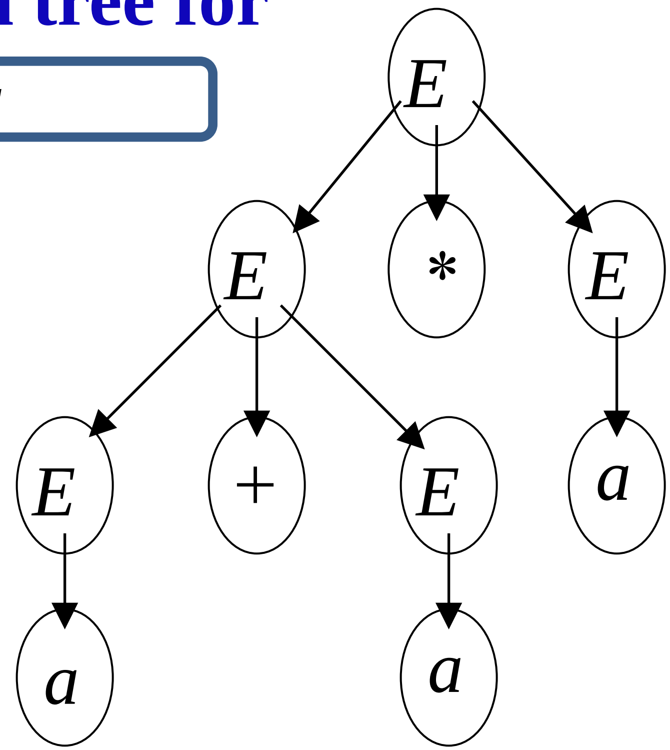
Ambiguity

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

Two derivation tree for

$$\boxed{a + a * a}$$

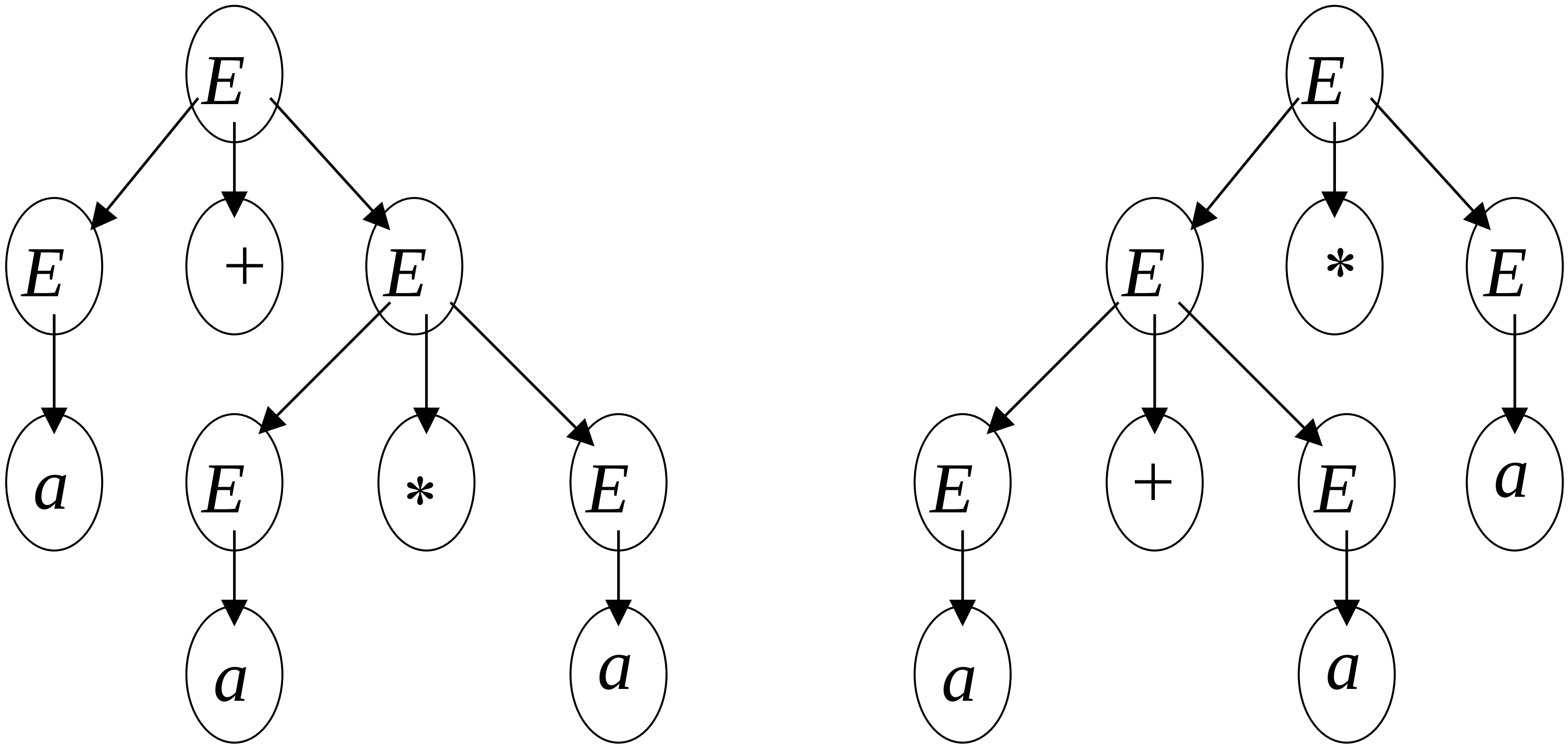




Ambiguity

take $a = 2$

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

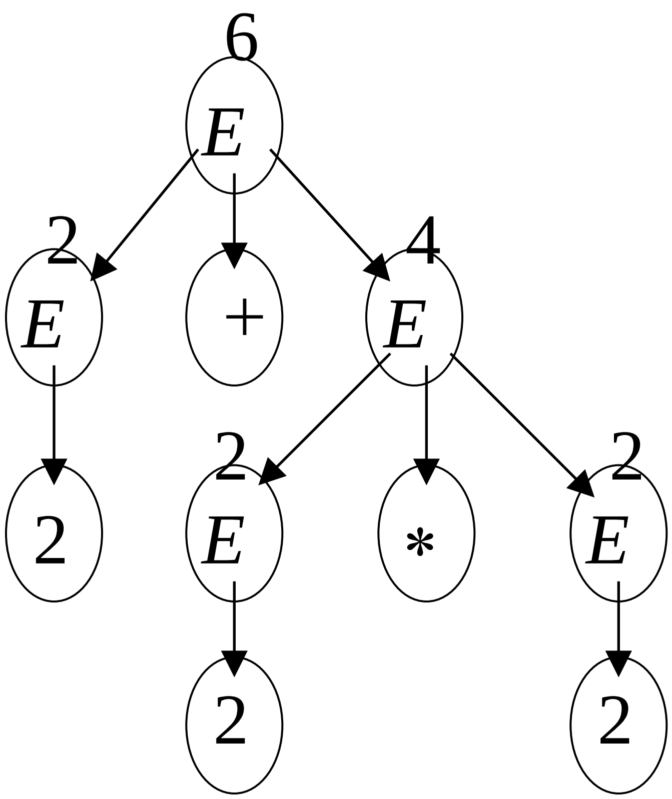


Ambiguity

Compute expression result using the tree

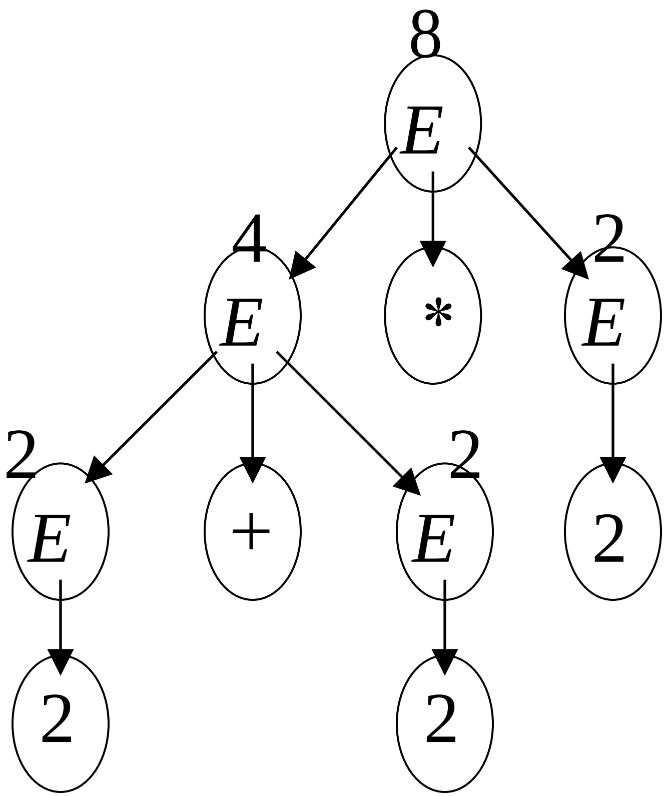
Good Tree

$2+2*2=6$



Bad Tree

$2+2*2=8$



Ambiguity

A successful example:

Ambiguous Grammar

$$
\begin{aligned}
E &\to E + E \\
E &\to E * E \\
E &\to (E) \\
E &\to a
\end{aligned}
$$

Equivalent

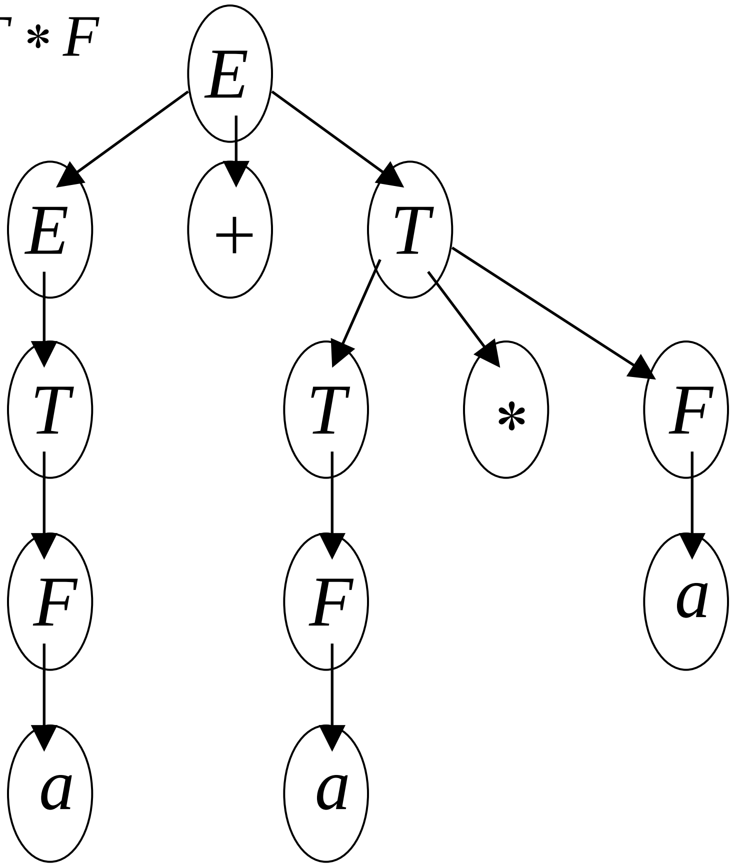
Non-Ambiguous Grammar

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

Ambiguity

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

$$\boxed{\begin{array}{l} E \rightarrow E+T \mid T \\ T \rightarrow T*F \mid F \\ F \rightarrow (E) \mid a \end{array}}$$



Unique derivation tree for

$$\boxed{a + a * a}$$

Grammar

Example

Determine whether the following grammar is ambiguous. If so, show two different derivation trees for the same string of terminals, and show a left-most derivation corresponding to each tree.

$$S \to aSbS$$

$$S \to aS$$

$$S \to c$$

$$S \Rightarrow a\ S\ b\ S \Rightarrow a\ a\ S\ b\ S \Rightarrow a\ a\ c\ b\ S \Rightarrow a\ a\ c\ b\ c$$

$$S \Rightarrow a\ S \Rightarrow a\ a\ S\ b\ S \Rightarrow a\ a\ c\ b\ S \Rightarrow a\ a\ c\ b\ c$$

Outline

- Syntax Analysis
- Grammar
- **Pushdown Machine**
- Parser

Pushdown Machine

Pushdown Machine

A **Pushdown machines** can be used for syntax analysis, just as finite state machines are used for lexical analysis.

A pushdown machine consists of:

- ✓ **A finite set of states**
- ✓ **A finite set of input symbols**
- ✓ $\boxed{\textbf{An infinite stack}}$
- ✓ **A state transition function**

Pushdown Machine

Pushdown Machine

- **Rows** are labeled by **stack symbols** and the **columns** are labeled by **input symbols**.
- ← character is used as an **endmarker**, indicating the end of the input string,
- ∇ symbol is a stack symbol which we are using to mark the bottom of the stack so that we can test for the **empty stack** condition.
- **Each cell** of those tables shows a **stack operation** (**push() or pop**), an input pointer **function** (**advance or retain**), and the **next state**. **Accept and Reject** are exits from the machine.

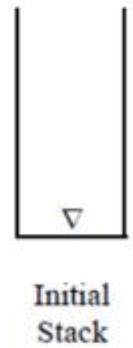
- A state transition function which takes as arguments the **current state**, the **current input symbol**, and the **symbol currently on top of the stack**; its result is the **new state of the machine**.
- On each state transition the machine may perform one of the stack operations, **push(X)** or **pop**, where **X** is one of the **stack symbols**.
- A state transition may include an exit from the machine labeled either **Accept** or **Reject**

Pushdown Machine

Example

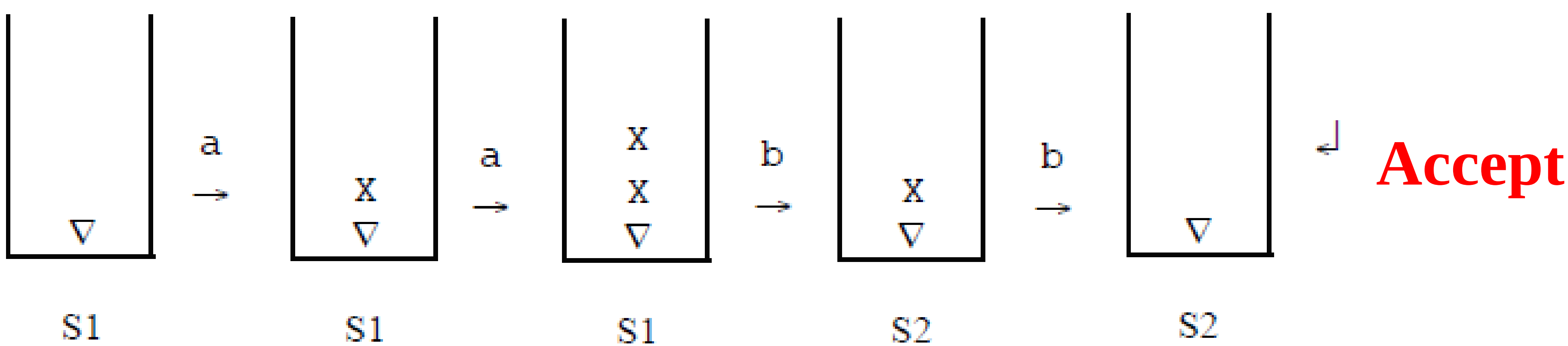
S1	a	b	↲
X	Push (X) Advance S1	Pop Advance S2	Reject
▽	Push (X) Advance S1	Reject	Accept

S2	a	b	↲
X	Reject	Pop Advance S2	Reject
▽	Reject	Reject	Accept



$$S \rightarrow ASB$$
$$S \rightarrow \epsilon$$
$$A \rightarrow a$$
$$B \rightarrow b$$

Show the sequence of stacks for the input string aabb ←

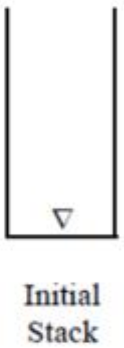


Pushdown Machine

Example

S1	a	b	↵
X	Push (X) Advance S1	Pop Advance S2	Reject
∇	Push (X) Advance S1	Reject	Accept

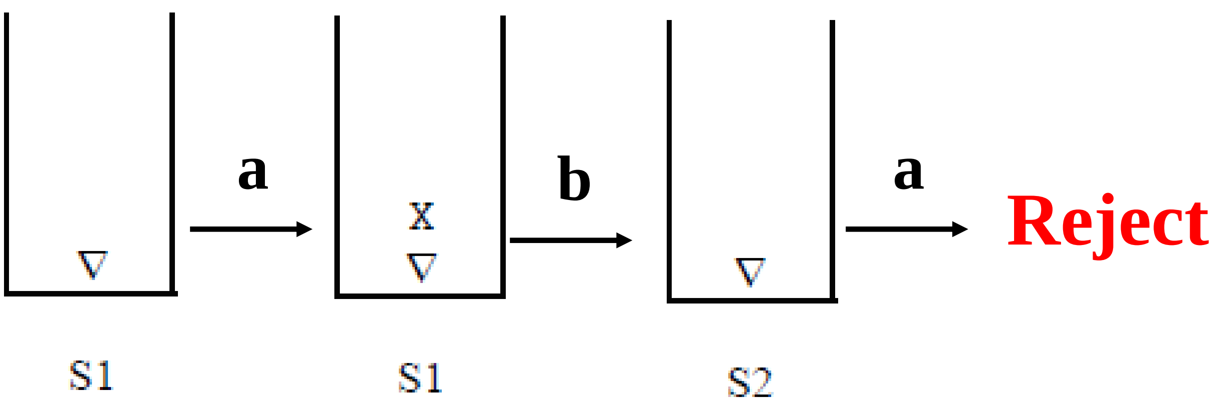
S2	a	b	↵
X	Reject	Pop Advance S2	Reject
∇	Reject	Reject	Accept



=

$$S \rightarrow ASB$$
$$S \rightarrow \epsilon$$
$$A \rightarrow a$$
$$B \rightarrow b$$

Show the sequence of stacks for the input string aba ←



Outline

- Syntax Analysis
- Grammar
- Pushdown Machine
- **Parser**

Parser

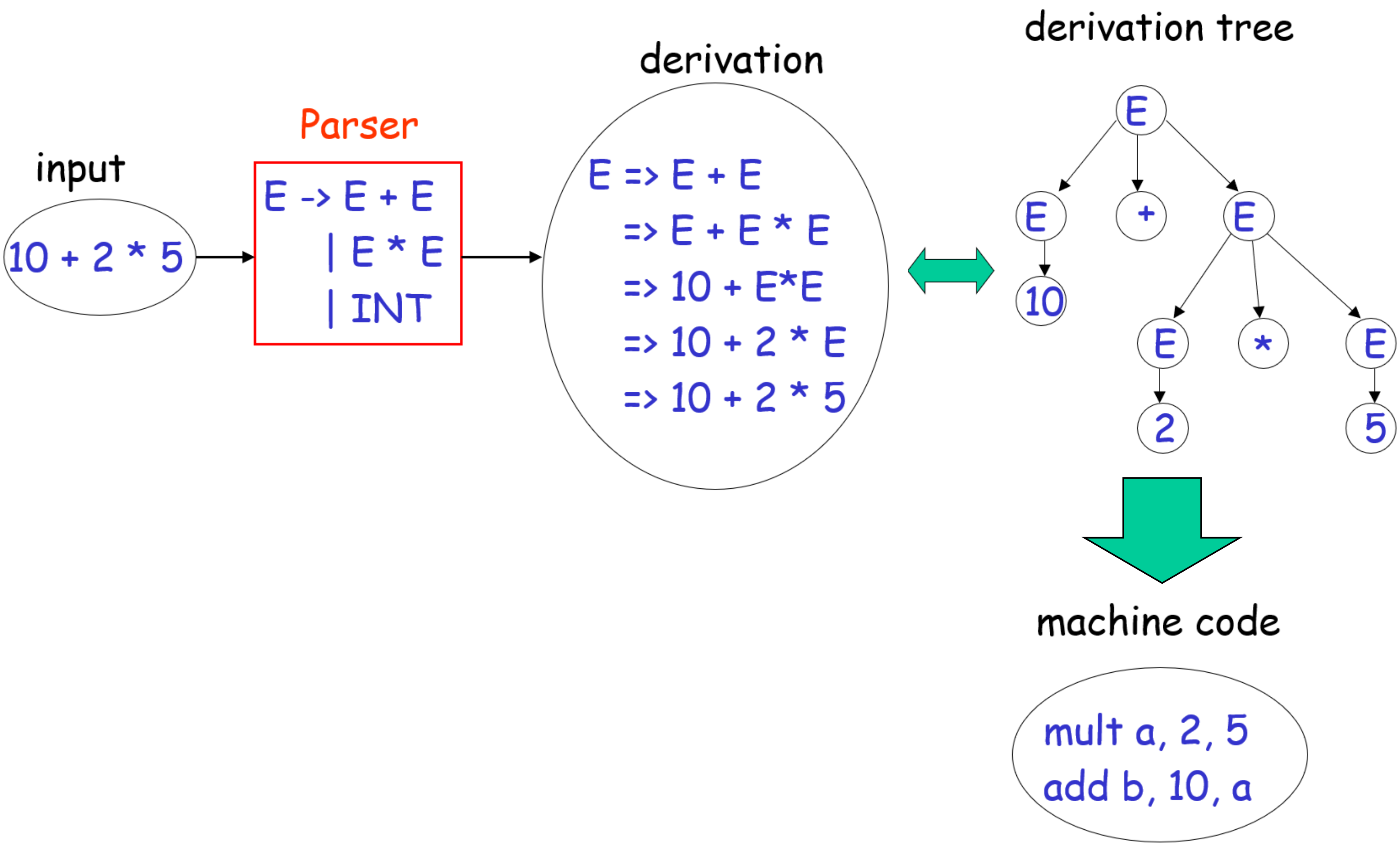
A parser knows the grammar of the programming language

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

Parser



THANKS

for your attention