COURSE: THEORY OF AUTOMATA COMPUTATION

TOPICS TO BE COVERED

- Normal Forms for Context-free Grammars
  - Chomsky Normal Form (CNF)
  - Griebach Normal Form (GNF)

CHOMSKY NORMAL FORM

Each productions has form:

$$A \to \underbrace{B}_{\text{variable}}\underbrace{C}_{\text{variable}} \quad \text{or} \quad A \to \underbrace{a}_{\text{terminal}}$$

Examples:

$$S \to AS$$
$$S \to a$$
$$A \to SA$$
$$A \to b$$

Chomsky
Normal Form

$$S \to AS$$
$$S \to AAS$$
$$A \to SA$$
$$A \to aa$$

Not Chomsky
Normal Form

CONVERTION TO CHOMSKY NORMAL FORM

- Example:

$$S \to ABa$$

$$A \to aab$$

$$B \to Ac$$

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

$$
\begin{aligned}
S &\to ABT_a \\
A &\to T_aT_aT_b \\
B &\to AT_c \\
T_a &\to a \\
T_b &\to b \\
T_c &\to c
\end{aligned}
$$

$$
\begin{aligned}
S &\to AV_1 \\
V_1 &\to BT_a \\
A &\to T_aT_aT_b \\
B &\to AT_c \\
T_a &\to a \\
T_b &\to b \\
T_c &\to c
\end{aligned}
$$

Introduce intermediate variable: V_2

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

$$\Rightarrow$$

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

Final grammar in Chomsky Normal Form:

$$S \to AV_1$$

$$V_1 \to BT_a$$

$$A \to T_a V_2$$

$$V_2 \to T_a T_b$$

$$B \to AT_c$$

$$T_a \to a$$

$$T_b \to b$$

$$T_c \to c$$

Initial grammar

$$S \to ABa$$

$$A \to aab$$

$$B \to Ac$$

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 $T_a \rightarrow a$

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

Theorem: For any context-free grammar (which doesn't produce λ) there is an equivalent grammar in Chomsky Normal Form

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:

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

symbol

variables

Observations

- Greinbach normal forms are very good for parsing
- It is hard to find the Greinbach normal form of any context-free grammar

COMPILERS

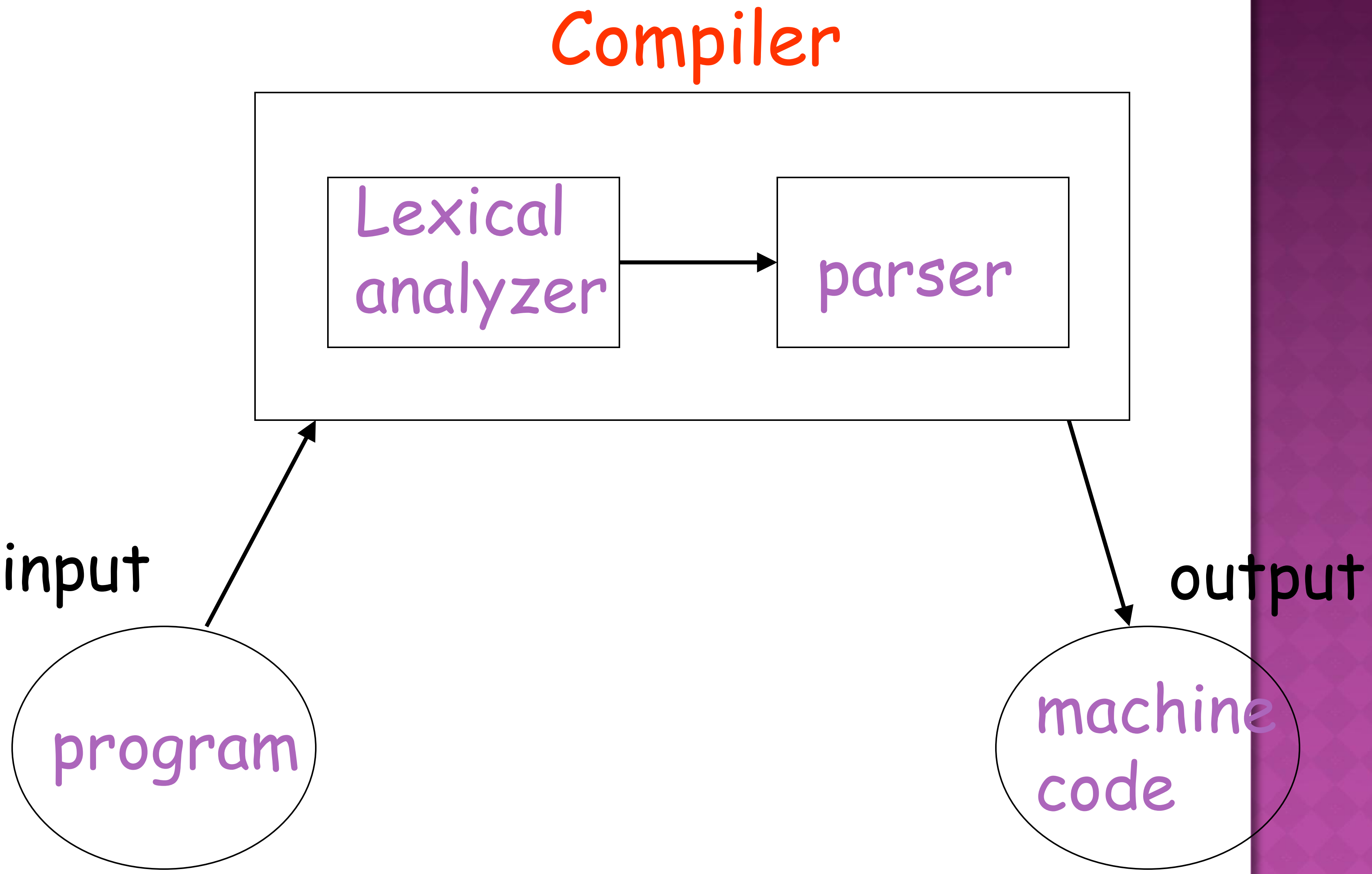
Program

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

→ **Compiler** →

Machine Code

```
Add v,v,0
cmp v,5
jmplt ELSE
THEN:
  add x, 12,v
ELSE:
WHILE:
cmp x,3
...
```


Compiler
Lexical analyzer
parser
input
program
output
machine code

A parser knows the grammar of the programming language

Parser

PROGRAM → STMT_LIST

STMT_LIST → STMT; STMT_LIST | STMT;

STMT → EXPR | IF_STMT | WHILE_STMT
| { STMT_LIST }

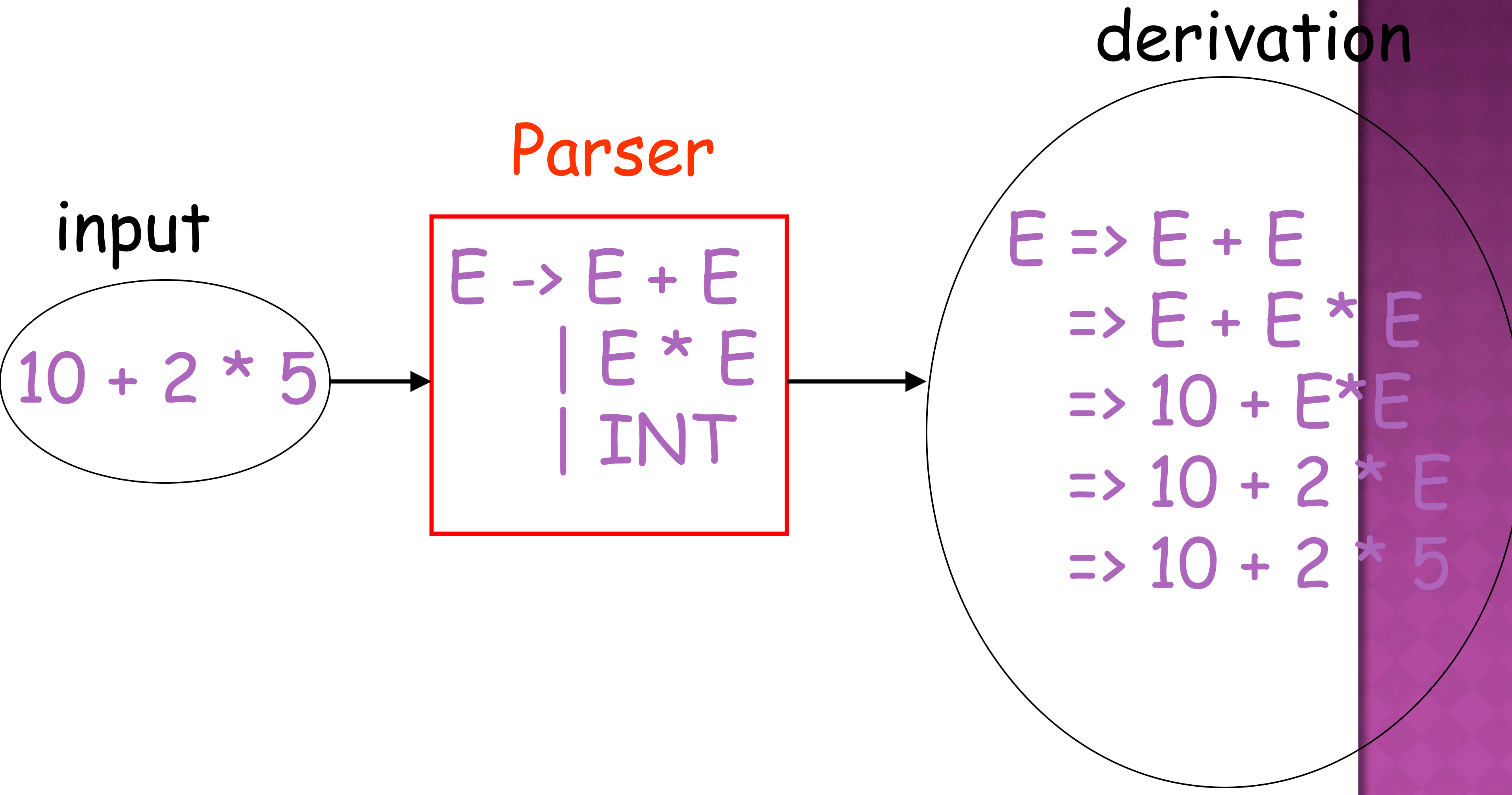
EXPR EXPR + EXPR | EXPR - EXPR | ID

IF_STMT → if (EXPR) then STMT
| → if (EXPR) then STMT else STMT

WHILE_STMT while (EXPR) do STMT

→

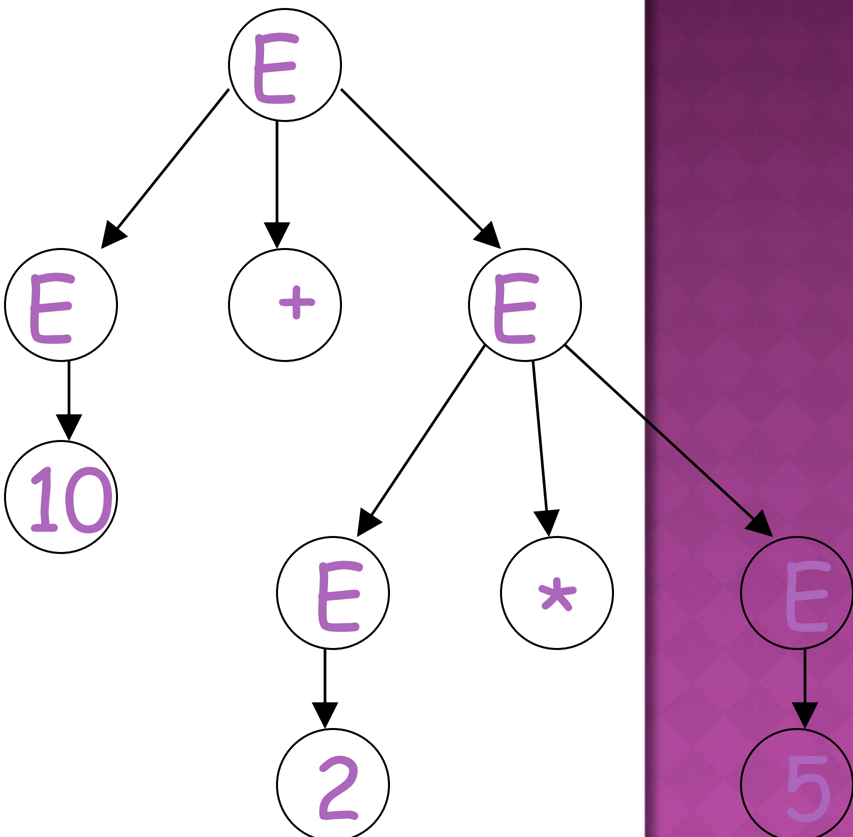
The parser finds the derivation of a particular input



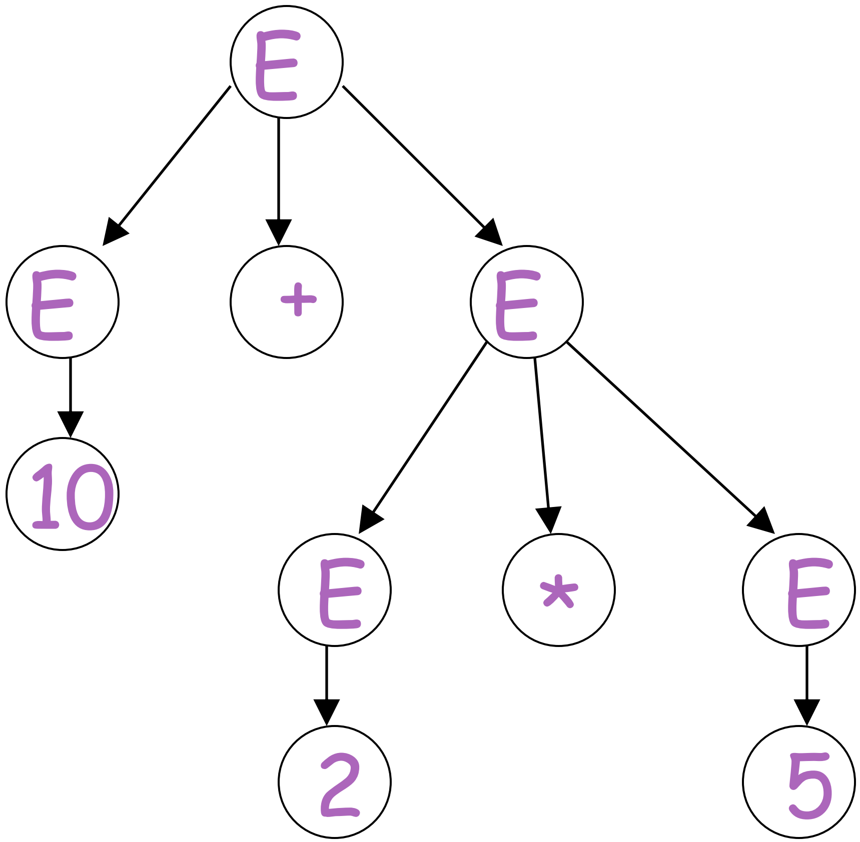
derivation

$$
\begin{aligned}
E &\Rightarrow E + E \\
&\Rightarrow E + E * E \\
&\Rightarrow 10 + E * E \\
&\Rightarrow 10 + 2 * E \\
&\Rightarrow 10 + 2 * 5
\end{aligned}
$$

derivation tree



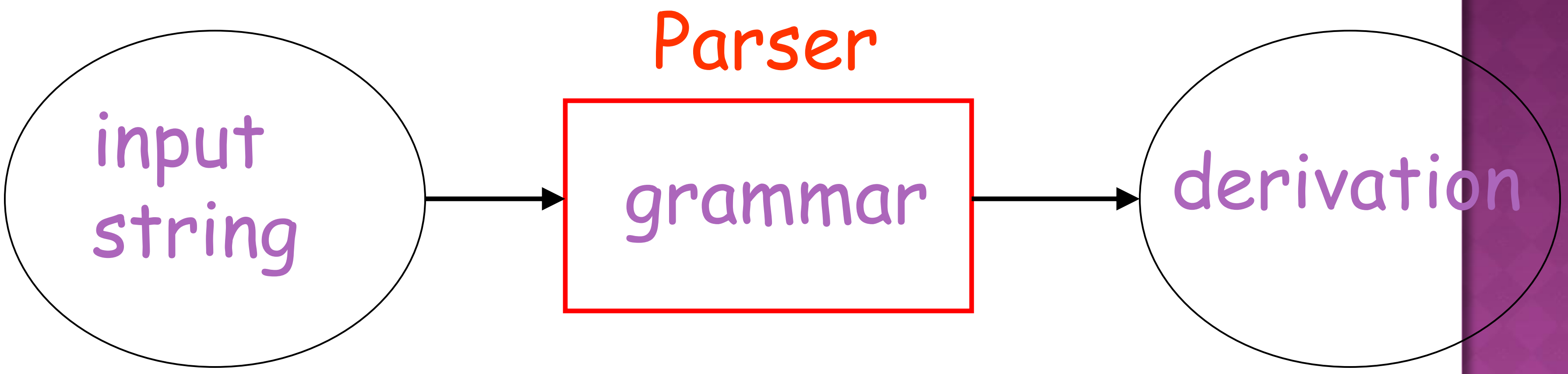
derivation tree



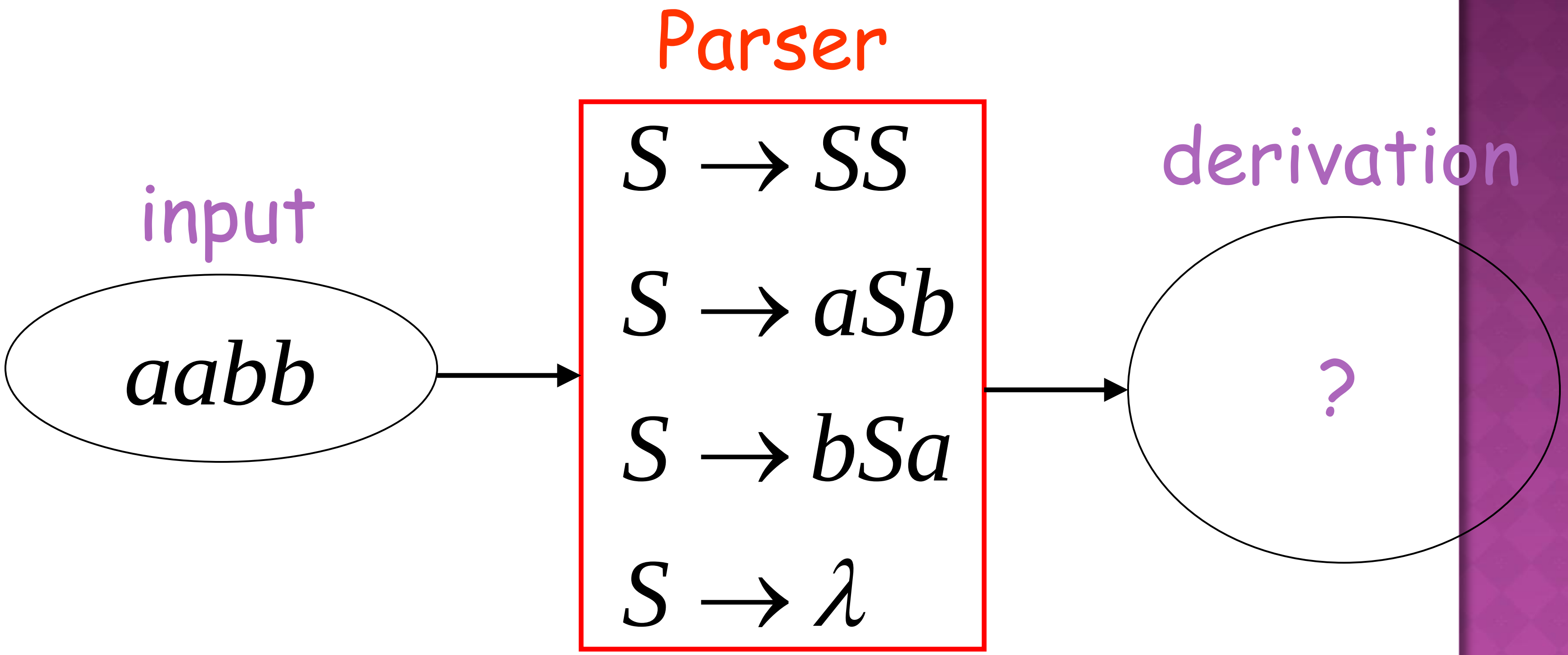
machine code

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

PARSING


Parser
input string
grammar
derivation

Example:


Parser
input
$aabb$
$S \to SS$
$S \to aSb$
$S \to bSa$
$S \to \lambda$
derivation
?

Exhaustive Search

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

Phase 1:

$$S \Rightarrow SS$$

$$S \Rightarrow aSb$$

$$S \Rightarrow bSa$$

$$S \Rightarrow \lambda$$

Find derivation of

$$aabb$$

All possible derivations of length 1

$$S \Rightarrow SS$$

$$aabb$$

$$S \Rightarrow aSb$$

$$\cancel{S \Rightarrow bSa}$$

$$\cancel{S \Rightarrow \lambda}$$

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

$S \Rightarrow SS \Rightarrow SSS$

$S \Rightarrow SS \Rightarrow aSbS$

$aabb$

Phase 1

$\cancel{S \Rightarrow SS \Rightarrow bSaS}$

$S \Rightarrow SS$

$S \Rightarrow SS \Rightarrow S$

$S \Rightarrow aSb$

$S \Rightarrow aSb \Rightarrow aSSb$

$S \Rightarrow aSb \Rightarrow aaSbb$

$\cancel{S \Rightarrow aSb \Rightarrow abSab}$

$\cancel{S \Rightarrow aSb \Rightarrow ab}$

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

Phase 2

$$S \Rightarrow SS \Rightarrow SSS$$

$$S \Rightarrow SS \Rightarrow aSbS$$

$$aabb$$

$$S \Rightarrow SS \Rightarrow S$$

$$S \Rightarrow aSb \Rightarrow aSSb$$

$$S \Rightarrow aSb \Rightarrow aaSbb$$

Phase 3

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

Final result of exhaustive search (top-down parsing)

input

$$aabb$$

Parser

$$S \to SS$$

$$S \to aSb$$

$$S \to bSa$$

$$S \to \lambda$$

derivation

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

Time complexity of exhaustive search

Suppose there are no productions of the form

$$A \to \lambda$$

$$A \to B$$

Number of phases for string w : approx. |w|

For grammar with k rules

Time for phase 1: k

k possible derivations

Time for phase 2: k^2

k^2 possible derivations

Time for phase $|w|$ is $k^{|w|}$:

A total of $k^{|w|}$ possible derivations

Total time needed for string w:

$$k + k^2 + \cdots + k^{|w|}$$

phase 1

phase 2

phase |w|

Extremely bad!!!

For general context-free grammars:

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

The CYK parser