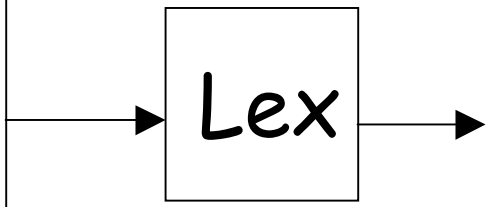
Lex

Lex is a lexical analyzer

Input

```
Var = 12 + 9;
if (test > 20)
   temp = 0;
else
   while (a < 20)
      temp++;
```



Output

```
Ident: Var
Integer: 12
Oper: +
Integer: 9
Semicolumn: ;
Keyword: if
Paren: (
Ident: test
Oper: >
....
```

For each kind of strings
there is a regular expression

Lex

Regular expressions

"+"
"-" /* operators */
"="

"if"
"then" /* keywords */

Lex

Regular expressions

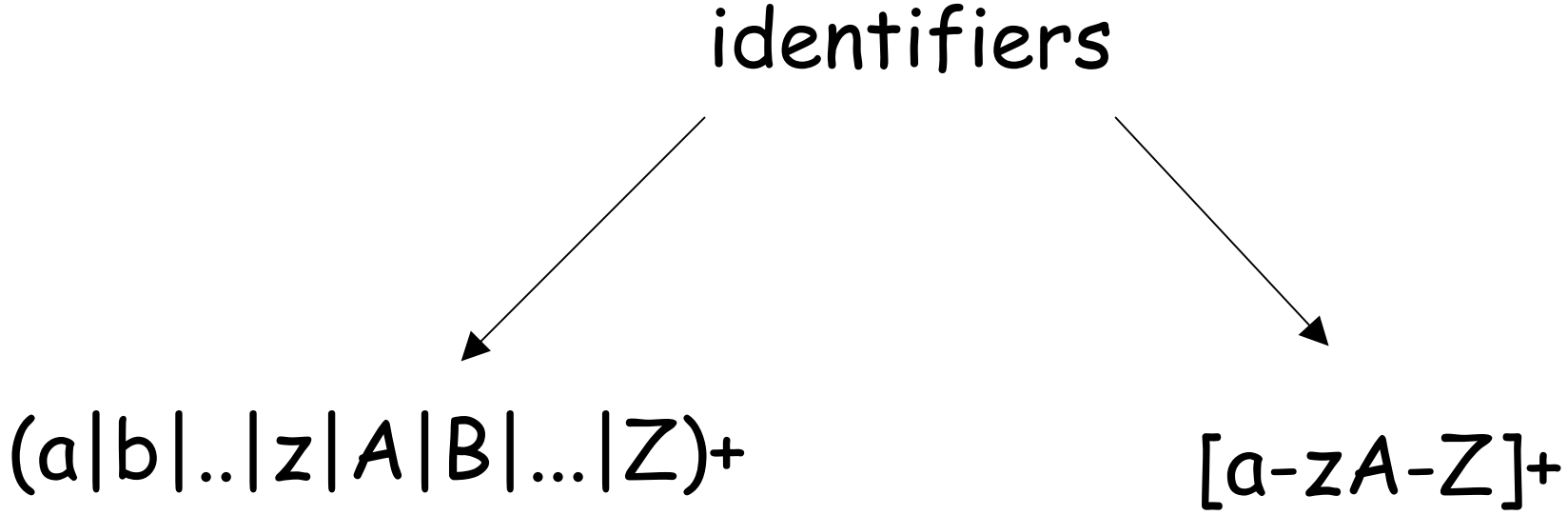
(0|1|2|3|4|5|6|7|8|9)+ /* integers */

(a|b|..|z|A|B|...|Z)+ /* identifiers */

integers

(0|1|2|3|4|5|6|7|8|9)+

[0-9]+


identifiers
(a|b|..|z|A|B|...|Z)+
[a-zA-Z]+

Each regular expression has an action:

Examples:

Regular expression	Action
\n	linenum++
[0-9]+	prinf("integer");
[a-zA-Z]+	printf("identifier");

Default action: ECHO;

Print the string identified to the output

A small program

```
%%
[ \t\n]                    ;    /*skip spaces*/

[0-9]+                     prinf("Integer\n");

[a-zA-Z]+                  printf("Identifier\n");
```

Input

```
1234   test
var 566    78
9800
```

Output

```
Integer
Identifier
Identifier
Integer
Integer
Integer
```

Another program

```
%{ int linenum = 1;
%}
%%
[ \t]                ;    /*skip spaces*/
\n                   linenum++;
[0-9]+               prinf("Integer\n");
[a-zA-Z]+            printf("Identifier\n");
.                    printf("Error in line: %d\n",
                             linenum);
```

Input

```
1234   test
var 566    78
9800  +
temp
```

Output

```
Integer
Identifier
Identifier
Integer
Integer
Integer
Error in line 3
Identifier
```

Lex matches the longest input string

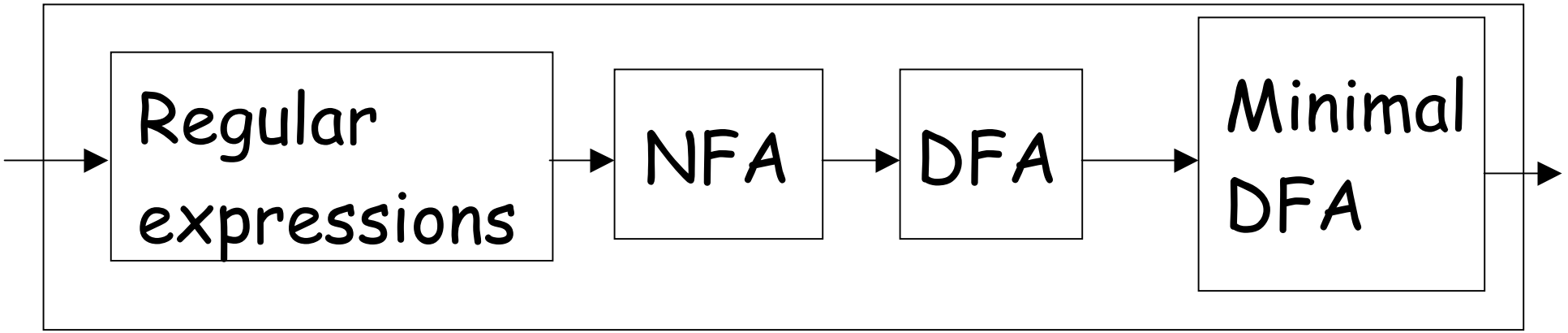
Regular Expressions

"if"

"ifend"

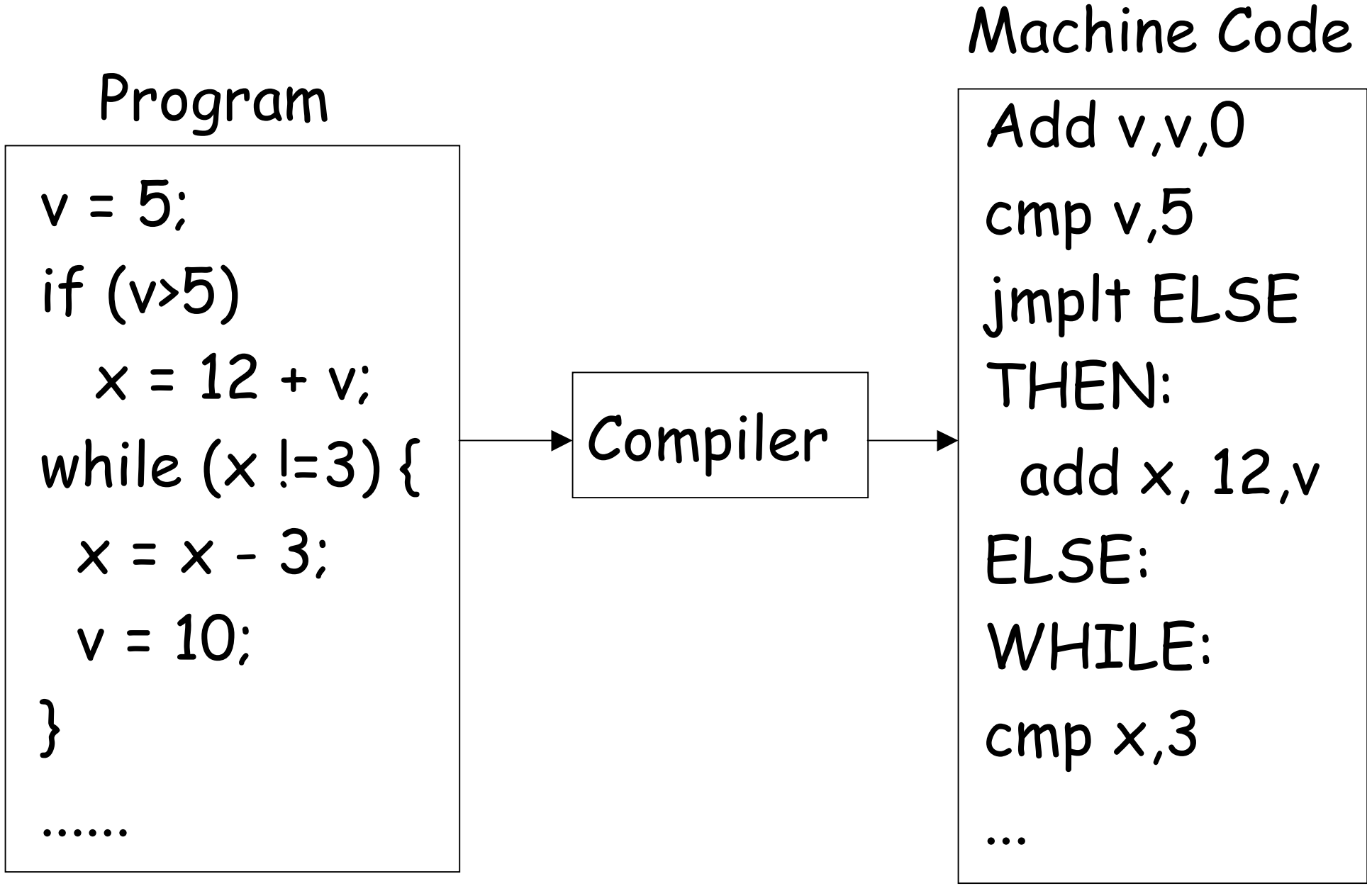
Input:	ifend	if	ifn
Matches:	"ifend"	"if"	nomatch

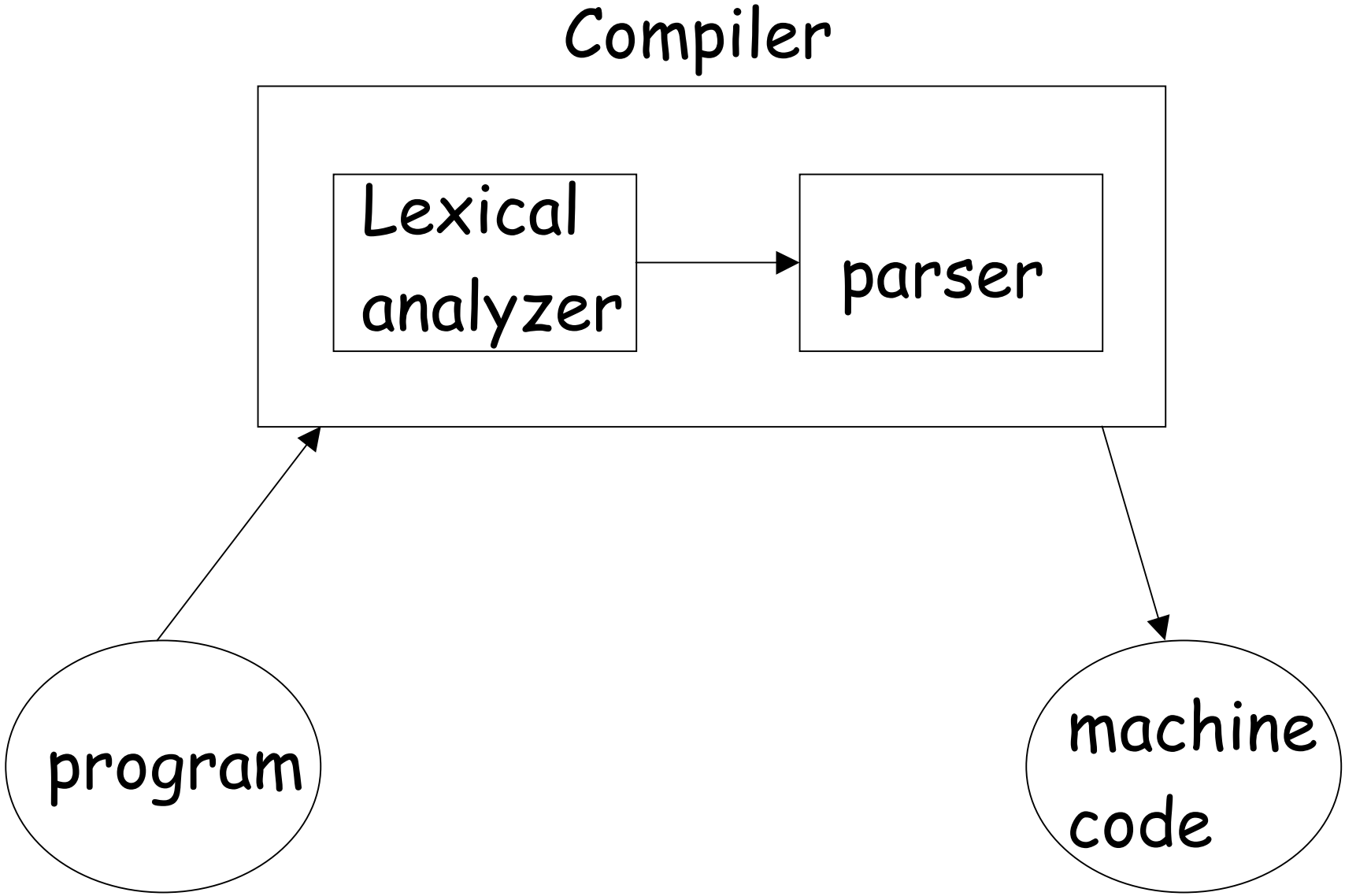
Internal Structure of Lex



The final states of the DFA are associated with actions

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
program
machine code

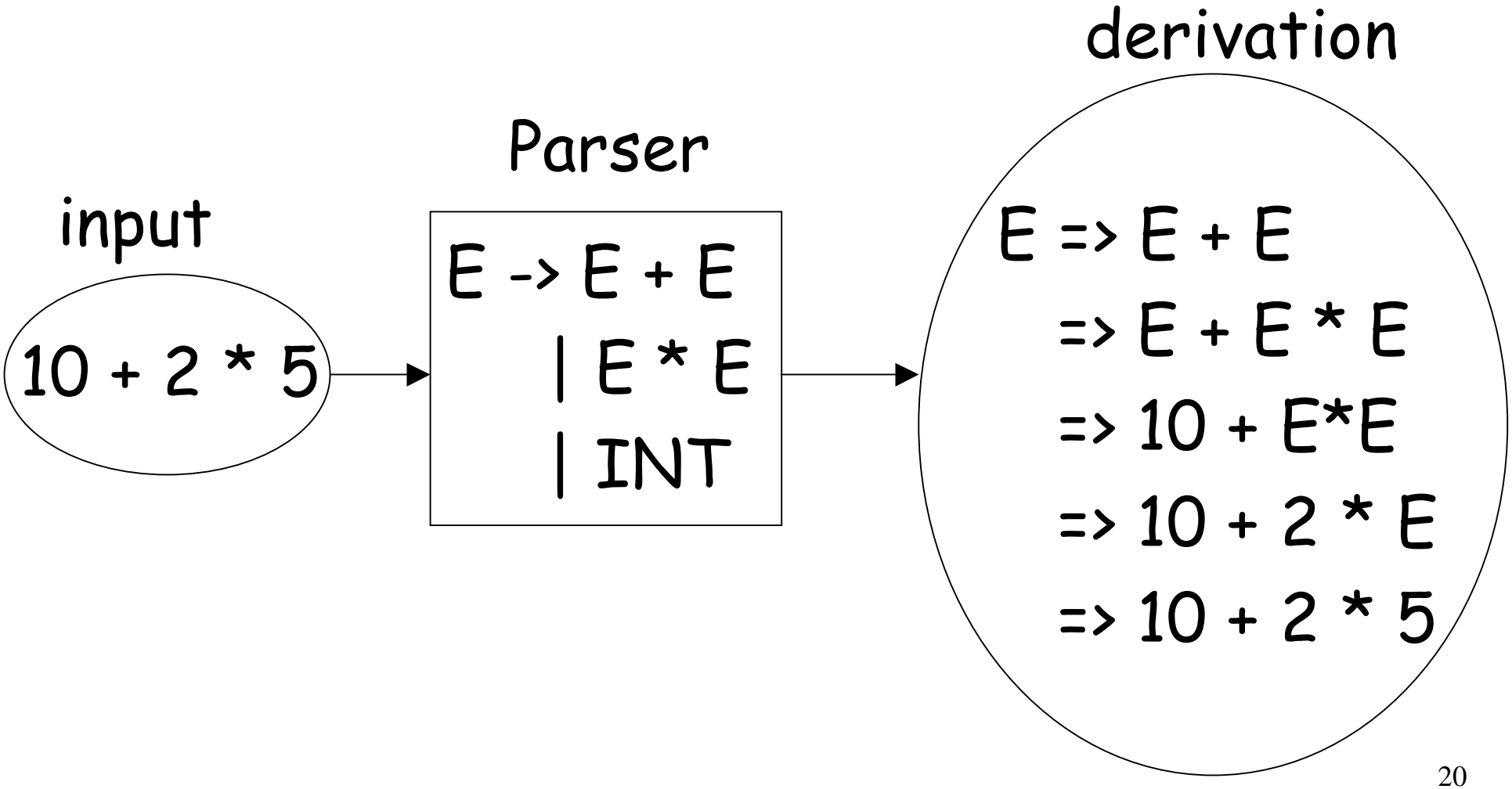
Parser knows the grammar
of the programming language

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
```

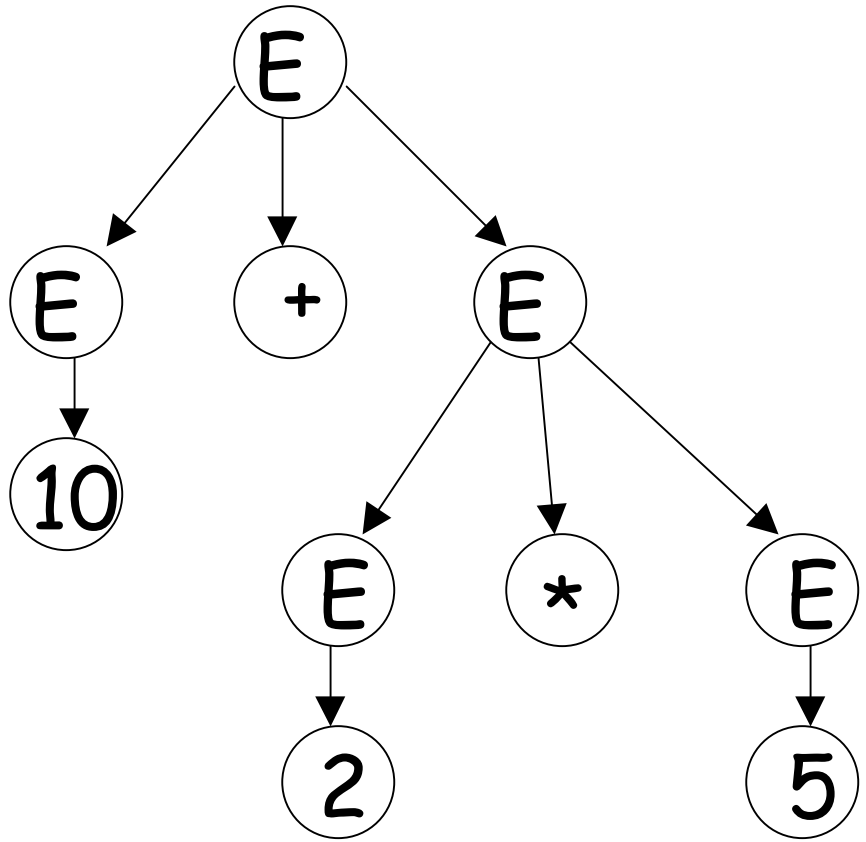
The parser constructs the derivation for the particular input program



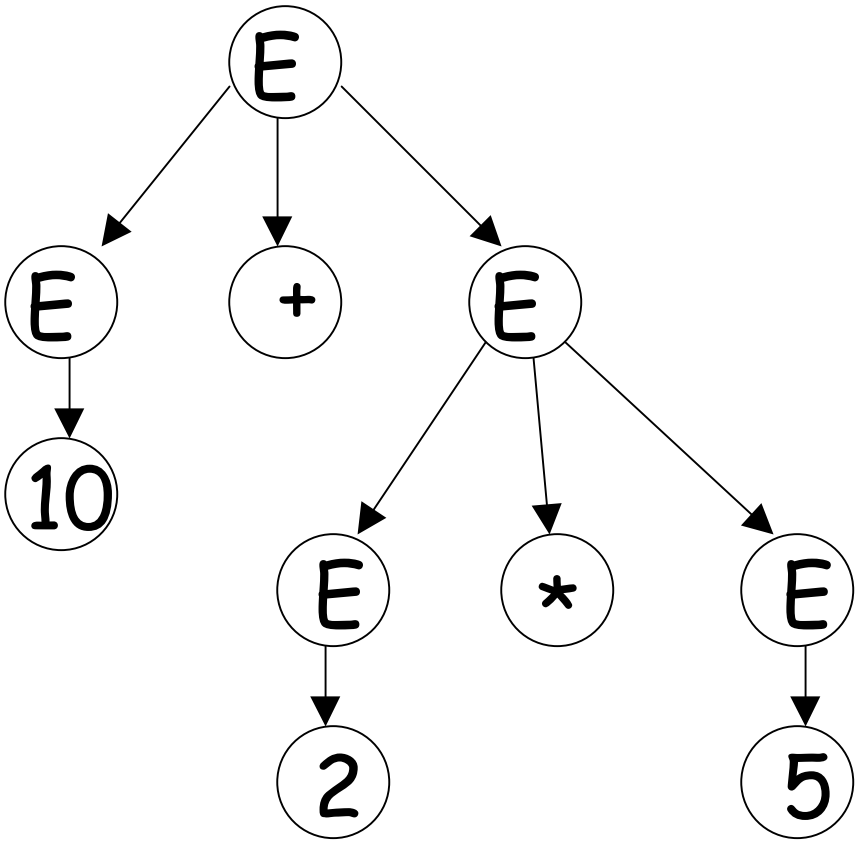
derivation

E => E + E
=> E + E * E
=> 10 + E*E
=> 10 + 2 * E
=> 10 + 2 * 5

derivation tree



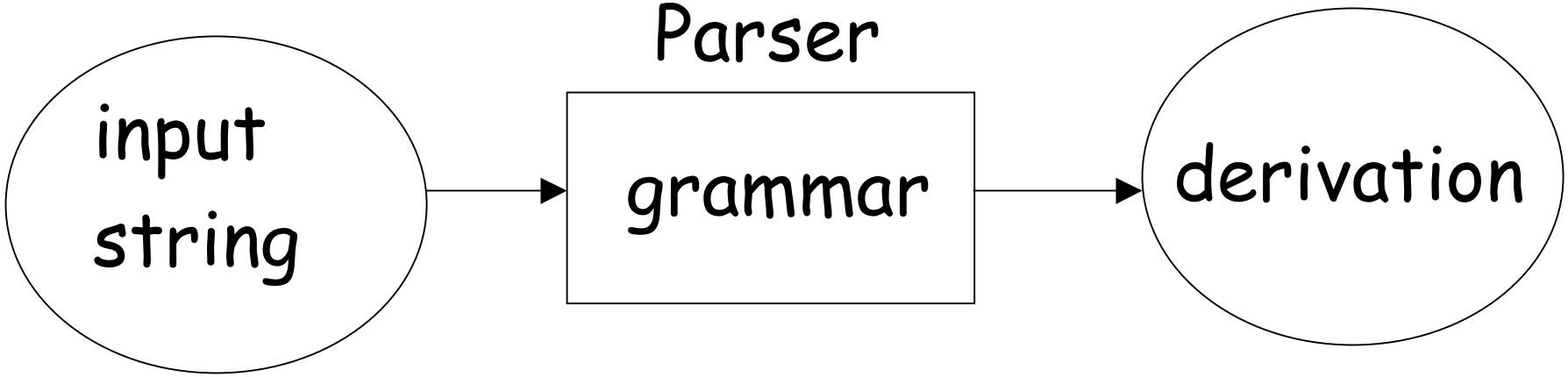
derivation tree



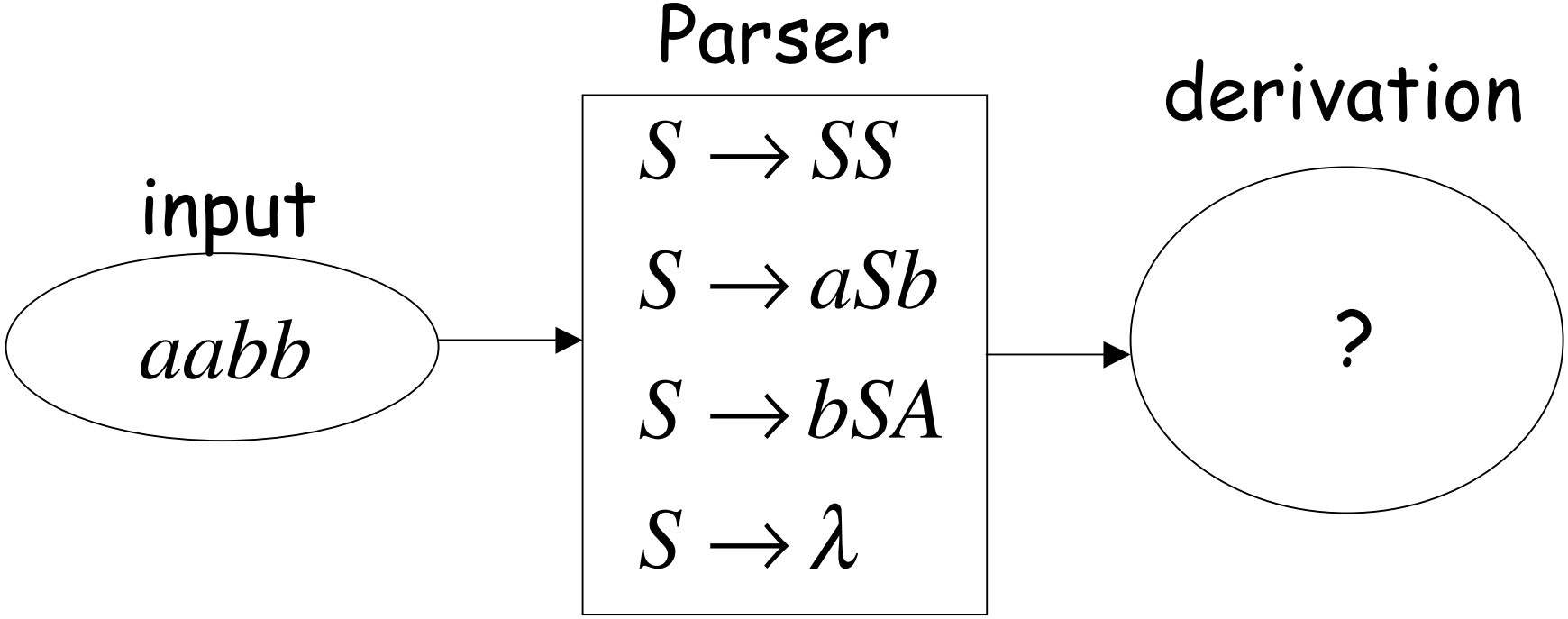
machine code

```
mult t1, 10, 5
add t2, 10, t1
```

Parsing


Parser
input
string
grammar
derivation

Example:


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

Exhaustive Search

$$S \rightarrow SS \,|\, aSb \,|\, bSA \,|\, \lambda$$

Phase 1:

$$S \Rightarrow SS$$

$$S \Rightarrow aSb$$

$$S \Rightarrow bSa$$

$$S \Rightarrow \lambda$$

$$aabb$$

$S \Rightarrow SS$

$S \Rightarrow aSb$

~~$S \Rightarrow bSa$~~

~~$S \Rightarrow \lambda$~~

$aabb$

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

Phase 2

$$S \Rightarrow SS \Rightarrow SSS$$

$$S \Rightarrow SS \Rightarrow aSbS$$

$$S \Rightarrow SS \Rightarrow bSaS$$

$$S \Rightarrow SS \Rightarrow S$$

Phase 1

$$S \Rightarrow SS$$

$$S \Rightarrow aSb$$

$$S \Rightarrow aSb \Rightarrow aSSb$$

$$S \Rightarrow aSb \Rightarrow aaSbb$$

$$S \Rightarrow aSb \Rightarrow bSaS$$

$$S \Rightarrow aSb \Rightarrow a$$

$$aabb$$

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

Phase 2

$S \Rightarrow SS \Rightarrow SSS$

$S \Rightarrow SS \Rightarrow aSbS$

~~$S \Rightarrow SS \Rightarrow bSaS$~~

$S \Rightarrow SS \Rightarrow S$

Phase 1

$S \Rightarrow SS$

$S \Rightarrow aSb$

$S \Rightarrow aSb \Rightarrow aSSb$

$S \Rightarrow aSb \Rightarrow aaSbb$

~~$S \Rightarrow aSb \Rightarrow bSaS$~~

~~$S \Rightarrow aSb \Rightarrow a$~~

$aabb$

Phase 2

$S \Rightarrow SS \Rightarrow SSS$

$S \Rightarrow SS \Rightarrow aSbS$

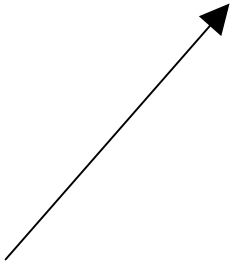
$S \Rightarrow SS \Rightarrow S$

Phase 3

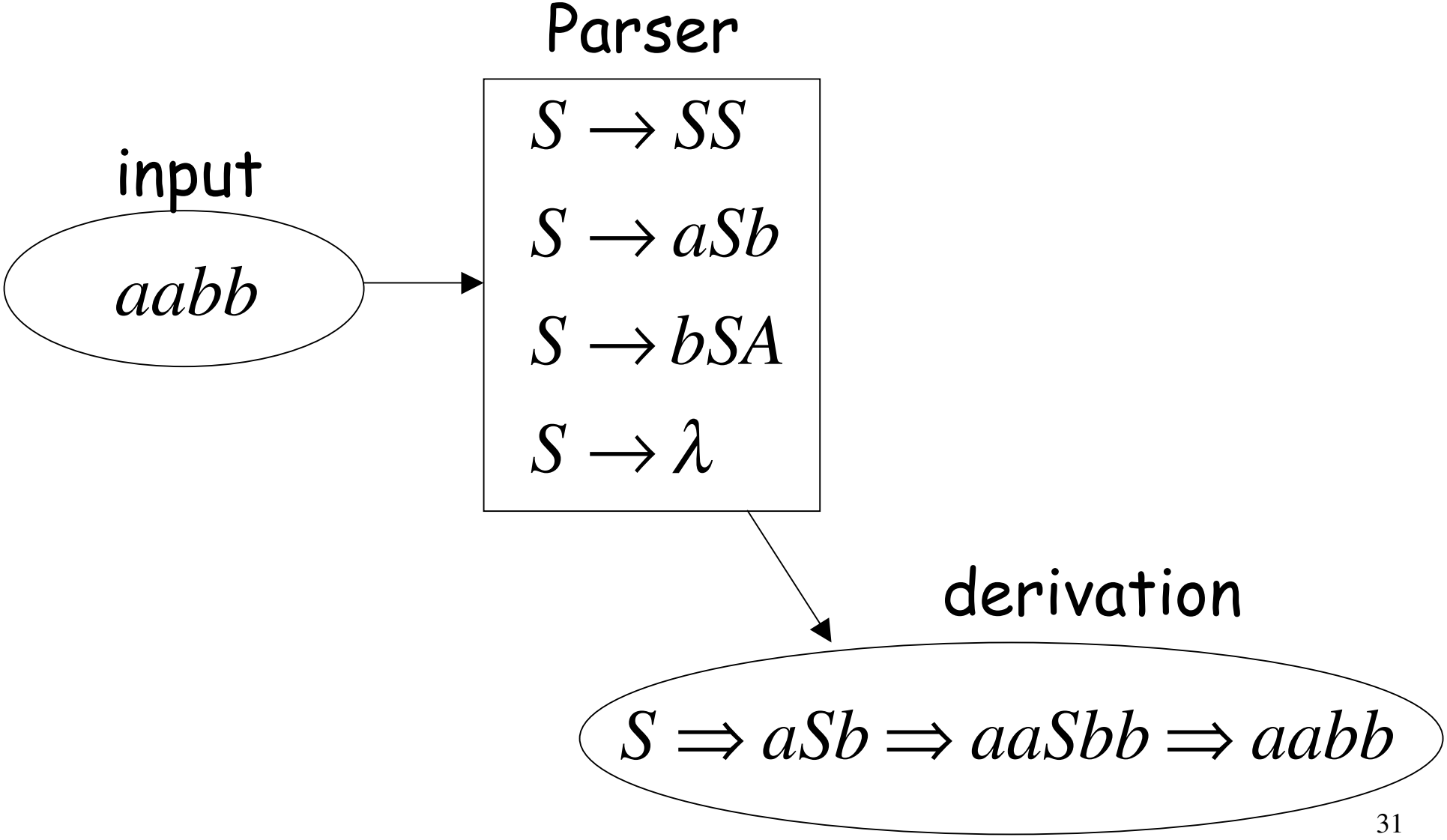
$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

$S \Rightarrow aSb \Rightarrow aSSb$

$S \Rightarrow aSb \Rightarrow aaSbb$



Final result of exhaustive search (Top-down parsing)



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 : $2|w|$

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 $2|w|$: $k^{2|w|}$

$k^{2|w|}$ possible derivations

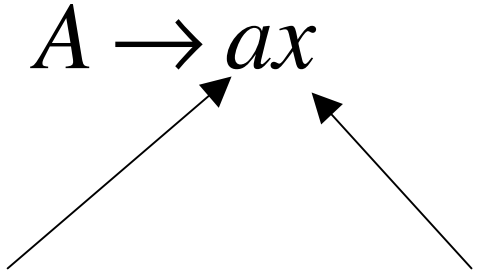
Total time needed for string w:

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

Extremely bad!!!

There exist faster algorithms
for specialized grammars

S-grammar:



symbol

string
of variables

(A, a) appears once

S-grammar example:

$$S \to aS$$
$$S \to bSS$$
$$S \to c$$

Each string has a unique derivation

$$S \Rightarrow aS \Rightarrow abSS \Rightarrow abcS \Rightarrow abcc$$

For S-grammars:

In the exhaustive search parsing
there is only one choice in each phase

Time for a phase: 1

Total time for parsing string w: $|w|$

For general context-free grammars:

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