Parsing

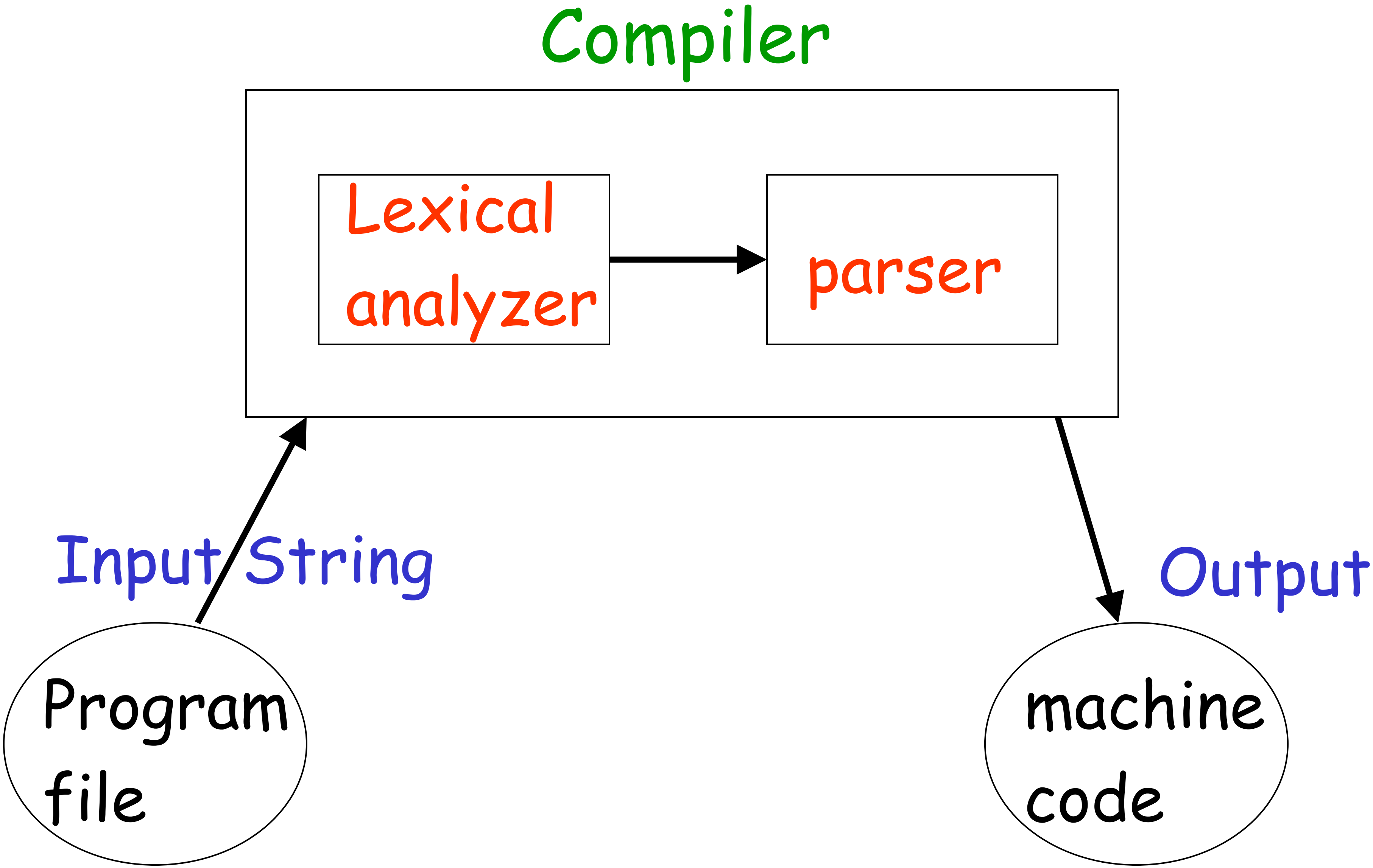
Program File

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

Compiler

Machine Code

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


Compiler
Lexical
analyzer
parser
Input String
Output
Program
file
machine
code

Lexical analyzer:

- Recognizes the lexemes of the input program file:

  Keywords (if, then, else, while,...),
  Integers,
  Identifiers (variables), etc

- It is built with DFAs (based on the theory of regular languages)

Parser:

- Knows the grammar of the programming language to be compiled

- Constructs derivation (and derivation tree) for input program file (input string)

- Converts derivation to machine code

Example 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 file

Input string

$10 + 2 * 5$

Example Parser

$$
\begin{aligned}
E &\rightarrow E + E \\
&\mid E * E \\
&\mid \mathrm{INT}
\end{aligned}
$$

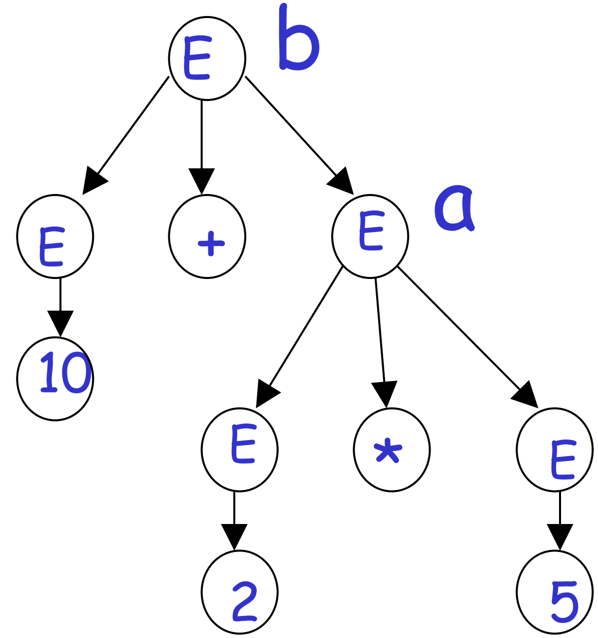
derivation

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

derivation

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

derivation tree



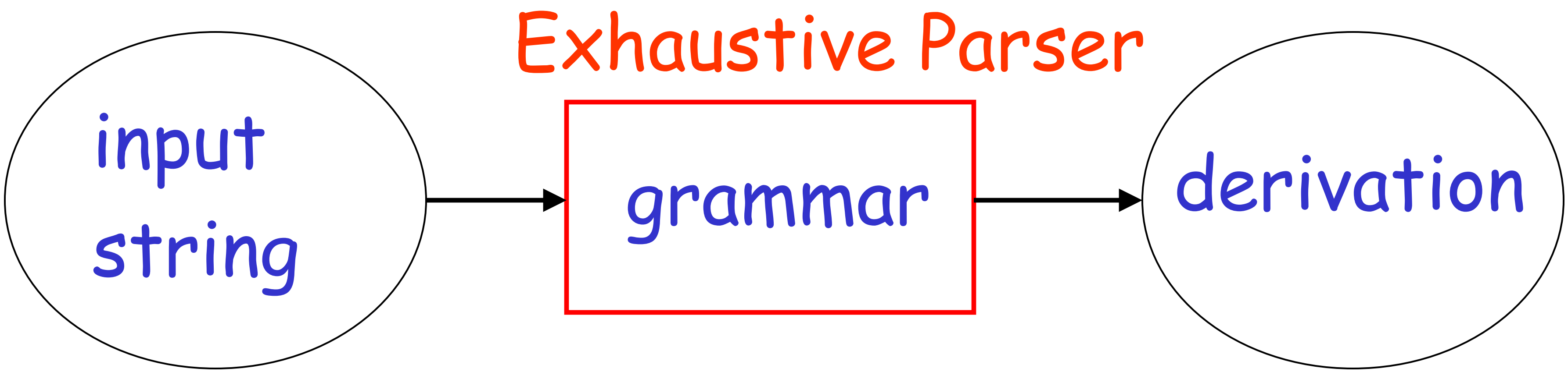
machine code

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

Derivation trees are used to build Machine code

A simple (exhaustive) parser

We will build an exhaustive search parser that examines all possible derivations



Example: Find derivation of string $aabb$

Exhaustive Parser

Input string

$aabb$

$S \to SS$

$S \to aSb$

$S \to bSa$

$S \to \varepsilon$

derivation

?

Exhaustive Search

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

Find derivation of $aabb$

Phase 1:

$$S \Rightarrow SS$$

$$S \Rightarrow aSb$$

$$S \Rightarrow bSa$$

$$S \Rightarrow \varepsilon$$

All possible derivations of length 1

$$\boxed{S \to SS \mid aSb \mid bSa \mid \varepsilon}$$

Find derivation of $aabb$

Phase 1:

$$S \Rightarrow SS$$

$$S \Rightarrow aSb$$

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

$$\cancel{S \Rightarrow \varepsilon}$$

Cannot possibly produce $aabb$

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

Phase 1

$$S \Rightarrow SS$$

$$S \Rightarrow aSb$$

In Phase 2,
explore the next step
of each derivation
from Phase 1

Phase 2

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

$$S \Rightarrow SS \Rightarrow SSS$$

$$S \Rightarrow SS \Rightarrow aSbS$$

Phase 1

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

$$S \Rightarrow SS \qquad S \Rightarrow SS \Rightarrow S$$

Find derivation of $aabb$

$$S \Rightarrow aSb \qquad S \Rightarrow aSb \Rightarrow aSSb$$

$$S \Rightarrow aSb \Rightarrow aaSbb$$

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

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

Phase 2

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

$$S \Rightarrow SS \Rightarrow SSS$$

$$S \Rightarrow SS \Rightarrow aSbS$$

$$S \Rightarrow SS \Rightarrow S$$

Find derivation of $aabb$

$$S \Rightarrow aSb \Rightarrow aSSb$$

$$S \Rightarrow aSb \Rightarrow aaSbb$$

In Phase 3 explore all possible derivations

Phase 2

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

$$S \Rightarrow SS \Rightarrow SSS$$

$$S \Rightarrow SS \Rightarrow aSbS$$

Find derivation of $aabb$

$$S \Rightarrow SS \Rightarrow S$$

$$S \Rightarrow aSb \Rightarrow aSSb$$

$$S \Rightarrow aSb \Rightarrow aaSbb$$

A possible derivation of Phase 3

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

Final result of exhaustive search

Exhaustive Parser

Input string

$aabb$

$S \to SS$

$S \to aSb$

$S \to bSa$

$S \to \varepsilon$

derivation

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

Time Complexity

Suppose that the grammar does not have productions of the form

$A \rightarrow \varepsilon$ (ε-productions)

$A \rightarrow B$ (unit productions)

Since the are no ε-productions

For any derivation of a string of terminals $w \in L(G)$

$$S \Rightarrow x_1 \Rightarrow x_2 \Rightarrow \Lambda \Rightarrow x_k \Rightarrow w$$

it holds that $|x_i| \leq |w|$ for all i

Since the are no unit productions

1. At most $|w|$ derivation steps are needed to produce a string x_j with at most $|w|$ variables

2. At most $|w|$ derivation steps are needed to convert the variables of x_j to the string of terminals w

Therefore, at most $2|w|$ derivation steps are required to produce w

The exhaustive search requires at most

$2|w|$ phases

Suppose the grammar has k productions

Possible derivation choices
to be examined in phase 1: at most k

Choices for phase 2: at most $k \times k = k^2$

Choices of phase 1

Number of Productions

In General

Choices for phase i: at most $k^{(i-1)} \times k = k^i$

Choices of phase i-1

Number of Productions

Total exploration choices for string w:

$$k + k^2 + \Lambda + k^{2|w|} = O(k^{2|w|})$$

phase 1

phase 2

phase 2|w|

Exponential to the string length

Extremely bad!!!

Faster Parsers

There exist faster parsing algorithms for specialized grammars

S-grammar: $A \to av$

Symbol

String of variables

Each pair of variable, terminal (X, σ) appears once in a production $X \to \sigma w$

(a restricted version of Greinbach Normal form)

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

Steps for a phase: 1

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

For general context-free grammars:

Next, we give a parsing algorithm that parses a string w in time $O(|w|^3)$

(this time is very close to the worst case optimal since parsing can be used to solve the matrix multiplication problem)

The CYK Parsing Algorithm

Input:
- Arbitrary Grammar G in Chomsky Normal Form
- String w

Output: Determine if $w \in L(G)$

Number of Steps: $O(|w|^3)$

Can be easily converted to a Parser

Basic Idea

Consider a grammar G
In Chomsky Normal Form

Denote by $F(w)$ the set of variables
that generate a string w

$$X \in F(w) \quad \text{if} \quad X \overset{*}{\Rightarrow} w$$

Suppose that we have computed $F(w)$

Check if $S \in F(w)$:

YES $\Rightarrow$ $w \in L(G)$ $\quad (S \overset{*}{\Rightarrow} w)$

NO $\Rightarrow$ $w \notin L(G)$

$F(w)$ can be computed recursively:

prefix suffix

Write $w = uv$

If $X \in F(u)$ and $Y \in F(v)$

$(X \overset{*}{\Rightarrow} u)$ $(Y \overset{*}{\Rightarrow} v)$

and there is production $H \rightarrow XY$

Then $H \in F(w)$

$$(H \Rightarrow XY \overset{*}{\Rightarrow} uY \overset{*}{\Rightarrow} uv = w)$$

Examine all prefix-suffix decompositions of w

	Length	Set of Variables that generate w
$w = u_1 v_{\|w\|-1}$	1	H_1
$w = u_2 v_{\|w\|-2}$	2	H_2
$\vdots$		$\vdots$
$w = u_{\|w\|-1} v_1$	|w|-1	$H_{\|w\|-1}$

Result: $F(w) = H_1 \cup H_2 \cup \Lambda \cup H_{|w|-1}$

At the basis of the recursion
we have strings of length 1

$$F(\sigma) = \{\text{Variables that generate symbol } \sigma\}$$

symbol

$$X \rightarrow \sigma$$

Very easy to find

Remark:

The whole algorithm can be implemented with dynamic programming:

First compute $F(w')$ for smaller substrings w' and then use this to compute the result for larger substrings of w

Example:

- Grammar G: $S \to AB$

  $A \to BB \mid a$

  $B \to AB \mid b$

- Determine if $w = aabbb \in L(G)$

Decompose the string $aabbb$ to all possible substrings

Length					
1	a	a	b	b	b
2	aa	ab	bb	bb	
3	aab	abb	bbb		
4	aabb	abbb			
5	aabbb				

$$S \to AB, \qquad A \to BB \mid a, \qquad B \to AB \mid b$$

$F(\sigma)$

a	a	b	b	b
{A}	{A}	{B}	{B}	{B}
aa	ab	bb	bb	
aab	abb	bbb		
aabb	abbb			
aabbb				

$$\boxed{S \to AB, \quad A \to BB \mid a, \quad B \to AB \mid b}$$

$F(\sigma) =$

a	a	b	b	b
{A}	{A}	{B}	{B}	{B}

$F(\cdot) =$

aa	ab	bb	bb
{}	{S,B}	{A}	{A}

aab abb bbb

aabb abbb

aabbb

$$S \to AB, \qquad A \to BB \mid a, \qquad B \to AB \mid b$$

$F(aa)$

prefix $a|a$ suffix

$$F(a) = \{A\} \quad \Big| \quad F(a) = \{A\}$$

There is no production of form $X \to AA$

Thus, $F(aa) = \{\}$

$F(ab)$

prefix $a|b$ suffix

$$F(a) = \{A\} \quad \Big| \quad F(b) = \{B\}$$

There are two productions of form $X \to AB$

$$S \to AB, \quad B \to AB$$

Thus, $F(ab) = \{S, B\}$

$$S \to AB, \qquad A \to BB \mid a, \qquad B \to AB \mid b$$

a	a	b	b	b
{A}	{A}	{B}	{B}	{B}
aa	ab	bb	bb	
{}	{S,B}	{A}	{A}	
aab	abb	bbb		
{S,B}	{A}	{S,B}		
aabb	abbb			
aabbb				

$$S \to AB, \quad A \to BB \mid a, \quad B \to AB \mid b$$

$F(aab)$

Decomposition 1

prefix $a \mid ab$ suffix

$F(a) = \{A\} \quad F(ab) = \{S, B\}$

There is no production of form $X \to AS$

There are 2 productions of form $X \to AB$

$$S \to AB, \quad B \to AB$$

$$H_1 = \{S, B\}$$

$$S \to AB, \qquad A \to BB \mid a, \qquad B \to AB \mid b$$

$F(aab)$

Decomposition 2

prefix $aa|b$ suffix

$F(aa) = \{\}$ $\quad$ $F(b) = \{B\}$

There is no production of form $X \to B$

$$H_2 = \{\ \}$$

$$F(aab) = H_1 \cup H_2 = \{S, B\} \cup \{\ \} = \{S, B\}$$

$$S \to AB, \quad A \to BB \mid a, \quad B \to AB \mid b$$

a {A}	a {A}	b {B}	b {B}	b {B}
aa {}	ab {S,B}	bb {A}	bb {A}	
aab {S,B}	abb {A}	bbb {S,B}		
aabb {A}	abbb {S,B}			
aabbb {S,B}				

Since

$$S \in F(w)$$

$$\Downarrow$$

$$aabbb \in L(G)$$

$$F(aabbb) = \{S,B\}$$

Approximate time complexity:

$$O(|w|^2 \cdot |w|) = O(|w|^3)$$

Number of substrings

Number of Prefix-suffix decompositions for a string