


# Context-Free Languages

## Regular languages:

- keywords in a programming language
- names of identifiers
- integers
- all misc symbols: = ;

## Not Regular languages:

- $\{a^n cb^n | n > 0\}$
- expressions -  $((a + b) - c)$  
- block structures ( $\{\}$  in Java/C++ and begin ... end in pascal)

**Definition:** A grammar  $G=(V,T,S,P)$  is context-free if all productions are of the form

$$A \rightarrow x$$

Where  $A \in V$  and  $x \in (V \cup T)^*$ .

**Definition:**  $L$  is a context-free language (CFL) iff  $\exists$  context-free grammar (CFG)  $G$  s.t.  $L=L(G)$ .

**Example:**  $G = (\{S\}, \{a, b\}, S, P)$

$S \rightarrow aSb \mid ab$

CFG

**Derivation of aaabbb:**

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$

$L(G) = \{a^n b^n \mid n > 0\}$

Also linear grammar - at most one variable on r.h.s.

**Example:**  $G = (\{S\}, \{a, b\}, S, P)$

$S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$

**Derivation of ababa:**

$S \Rightarrow aSa \Rightarrow abSba \Rightarrow ababa$

$\Sigma = \{a, b\}, L(G) = \{w \in \Sigma^* \mid w = w^R\}$

Example:  $G = (\{S, A, B\}, \{a, b, c\}, S, P)$

$$\begin{aligned} S &\rightarrow AcB \\ A &\rightarrow aAa \mid \lambda \\ B &\rightarrow Bbb \mid \lambda \end{aligned}$$

CFG  
not Reg  
is Reg 2

$$L(G) = \{ a^{2n} c b^{2m} \mid n, m \geq 0 \}$$

not linear  
grammar

Derivations of aacbb:

1.  $S \Rightarrow \underline{A}cB \Rightarrow a\underline{A}acB \Rightarrow aac\underline{B} \Rightarrow aac\underline{B}bb \Rightarrow aacbb$

leftmost  
der

2.  $S \Rightarrow Ac\underline{B} \Rightarrow Ac\underline{B}bb \Rightarrow \underline{A}cbb \Rightarrow a\underline{A}acbb \Rightarrow aacbb$

rightmost  
der

Note: Next variable to be replaced is underlined.

**Definition: Leftmost derivation** - in each step of a derivation, replace the leftmost variable. (see derivation 1 above).

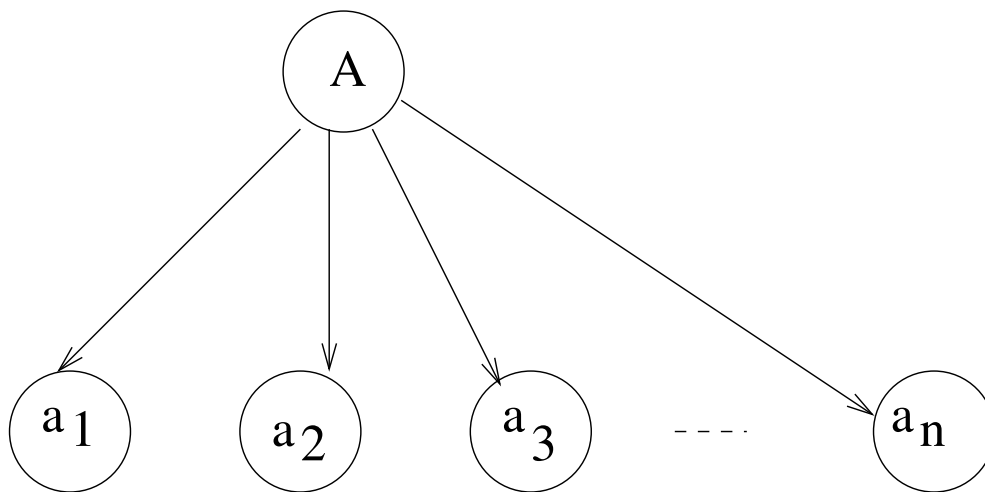
**Definition: Rightmost derivation** - in each step of a derivation, replace the rightmost variable. (see derivation 2 above).

**Derivation Trees** (also known as “parse trees”)

A derivation tree represents a derivation but does not show the order productions were applied.

A derivation tree for  $G=(V,T,S,P)$ :

- root is labeled  $S$
- leaves labeled  $x$ , where  $x \in T \cup \{\lambda\}$
- nonleaf vertices labeled  $A$ ,  $A \in V$
- For rule  $A \rightarrow a_1 a_2 a_3 \dots a_n$ , where  $A \in V$ ,  $a_i \in (T \cup V \cup \{\lambda\})$ ,



Example:  $G = (\{S, A, B\}, \{a, b, c\}, S, P)$

$$S \rightarrow AcB$$

$$A \rightarrow aAa \mid \lambda$$

$$B \rightarrow Bbb \mid \lambda$$

The screenshot shows the JFLAP interface with the 'Brute Parser' tab selected. The input string is 'aaaacbbb', and the message 'String accepted! 19 nodes generated.' is displayed. Below the input field, there are two sliders for 'Table Text Size'. The main derivation table is shown below the sliders.

LHS		RHS
S	→	AcB
A	→	aAa
A	→	λ
B	→	Bbb
B	→	λ

S	→	AcB	S
A	→	aAa	AcB
B	→	Bbb	aAacB
A	→	aAa	aAacBbb
B	→	Bbb	aaAaacBbb
A	→	λ	aaAaacBbbbb
B	→	λ	aaaacBbbbb
			aaaacbbb

Derived λ from B. Derivations complete.



With parse tree

JFLAP : (cfgExample1.jff)

File Input Test Convert Help

Editor Brute Parser

Start Pause Step Noninverted Tree

Input **aaaacbbb**

String accepted! 19 nodes generated.

Input Field Text Size (For optimization, move one of the window size adjusters around this window after resizing the text field)

Table Text Size

LHS		RHS
S	→	AcB
A	→	aAa
A	→	λ
B	→	Bbb
B	→	λ

Derived λ from B. Derivations complete.

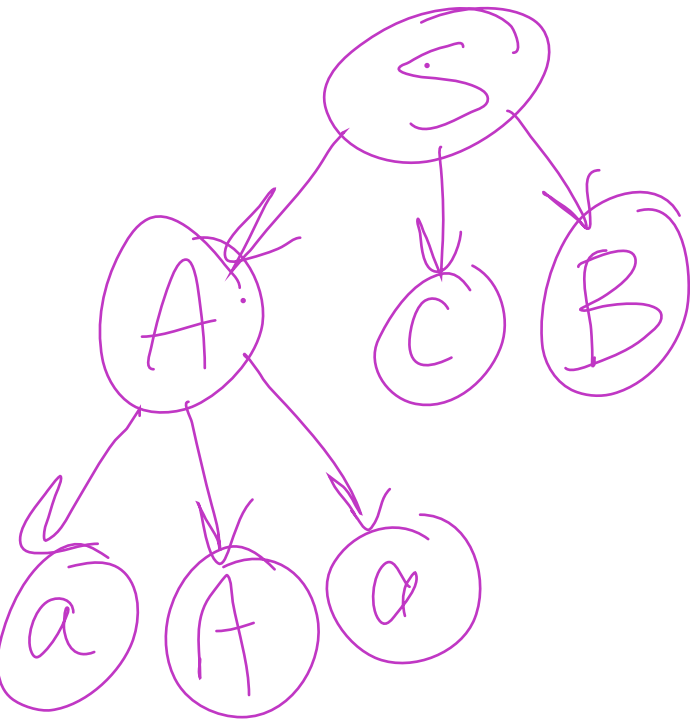
Definitions Partial derivation tree - subtree of derivation tree.

If partial derivation tree has root S then it represents a sentential form.

Leaves from left to right in a derivation tree form the *yield* of the tree.

Yield ( $w$ ) of derivation tree is such that  $w \in L(G)$ .

The yield for the example ~~above~~ <sup>below</sup> is



below

$aAaB$

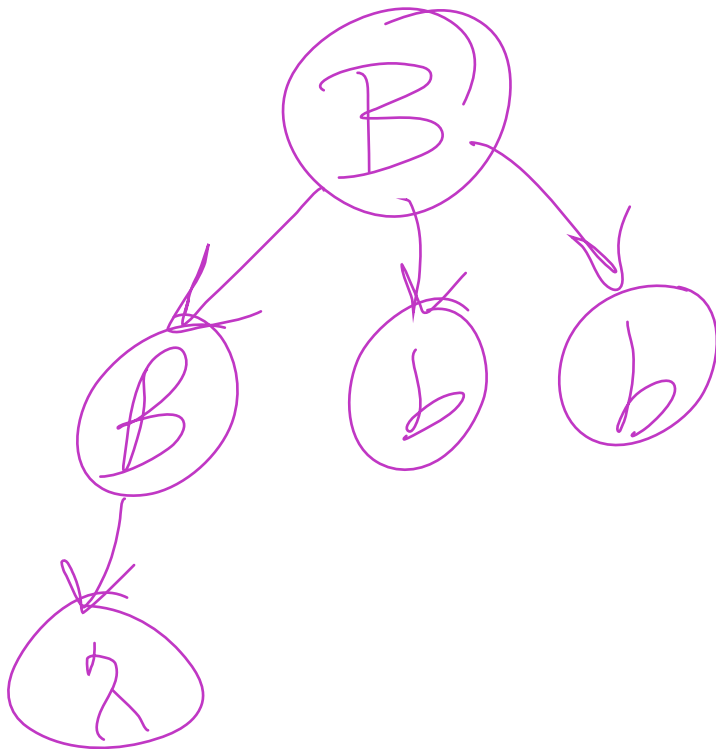
Sentential form  
because  
partial tree  
starts w/ S

Example of partial derivation tree  
that has root S:

See above

The yield of this example is  
\_\_\_\_\_ which is a sentential  
form.

**Example of partial derivation tree that does not have root S:**



yield is bb

$S \rightarrow A \cup B \rightarrow aAacB \rightarrow$   
 $aacB \rightarrow aacBbb \rightarrow$

all sentential forms  $aacbib$

Membership Given CFG  $G$  and string  $w \in \Sigma^*$ , is  $w \in L(G)$ ?

If we can find a derivation of  $w$ , then we would know that  $w$  is in  $L(G)$ .

Motivation

$G$  is grammar for Java  
 $w$  is Java program.  
Is  $w$  syntactically correct?

## Example

$G = (\{S\}, \{a, b\}, S, P), P =$

$S \rightarrow SS \mid aSa \mid b \mid \lambda$

$L_1 = L(G) = \{w \in \Sigma^* \mid w \text{ has an even number of } a\text{'s}\}$

Is  $abbab \in L(G)$ ?

# Exhaustive Search Algorithm

For all  $i=1,2,3,\dots$

Examine all sentential forms yielded  
by  $i$  substitutions

Example: Is  $abbab \in L(G)$ ?

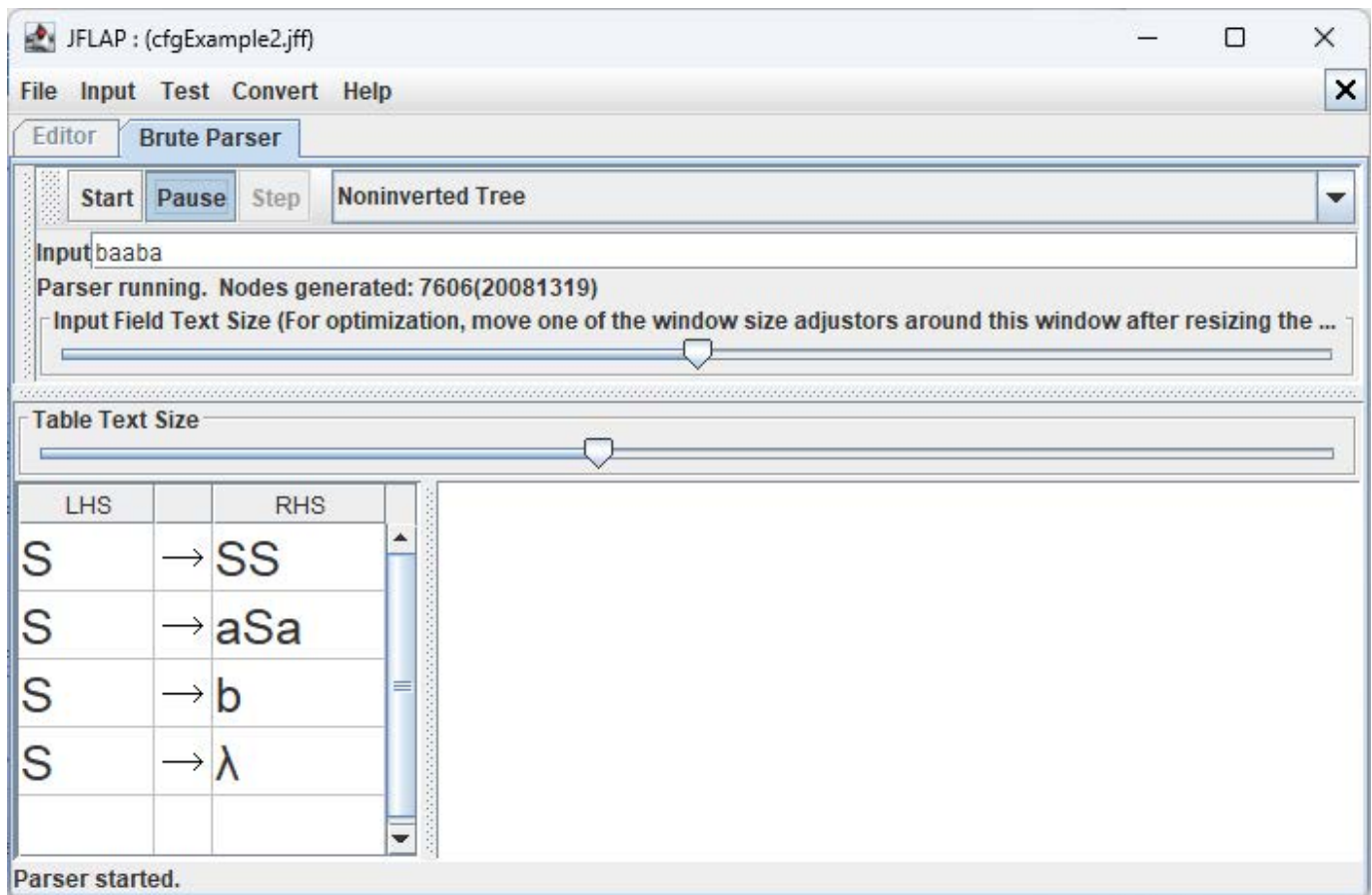
$i=1$

$S \rightarrow SS$   
 $S \rightarrow aSa$   
 $S \rightarrow b$   
 $S \rightarrow \lambda$

$i=2$

$S \rightarrow SS \rightarrow SSS$   
 $S \rightarrow SS \rightarrow aSaS$   
 $S \rightarrow SS \rightarrow bS$   
 $S \rightarrow SS \rightarrow S$   
 $S \rightarrow SS \rightarrow \lambda b$   
 $\vdots$   
 $S \rightarrow aSa \rightarrow aSSa$

The brute force parser is trying to derive baaba but it is taking too long for it to realize that string should be rejected.





Theorem If CFG  $G$  does not contain rules of the form

unit prod

$$\begin{aligned} A &\rightarrow \lambda \\ A &\rightarrow B \end{aligned}$$

shrinks  $\rightarrow$   
same size

where  $A, B \in V$ , then we can determine if  $w \in L(G)$  or if  $w \notin L(G)$ .

- Proof: Consider

1. length of sentential forms
2. number of terminal symbols in a sentential form

Either 1 or 2 increases with each derivation

Derivation of string  $w$  in  $L(G)$   
takes  $\leq 2|w|$

Example: Let  $L_2 = L_1 - \{\lambda\}$ .  $L_2 = L(G)$   
 where  $G$  is:

$$S \rightarrow SS \mid aa \mid aSa \mid b$$

Show  $baaba \notin L(G)$ .

- $i=1$
1.  $S \Rightarrow SS$
  2.  $S \Rightarrow aSa$
  3.  $S \Rightarrow aa$
  4.  $S \Rightarrow b$

- $i=2$
1.  $S \Rightarrow SS \Rightarrow SSS$
  2.  $S \Rightarrow SS \Rightarrow aSaS$
  3.  $S \Rightarrow SS \Rightarrow aaS$
  4.  $S \Rightarrow SS \Rightarrow bS$
  5.  $S \Rightarrow aSa \Rightarrow aSSa$
  6.  $S \Rightarrow aSa \Rightarrow aaSaa$
  7.  $S \Rightarrow aSa \Rightarrow aaaa$
  8.  $S \Rightarrow aSa \Rightarrow aba$

⋮

*stop at 10 steps if not found it*

With the modified grammar, the brute force parser rejects the string quickly

The screenshot shows the JFLAP application window titled "JFLAP : (cfgExample2.jff)". The menu bar includes "File", "Input", "Test", "Convert", and "Help". The "Brute Parser" tab is active, showing a "Start", "Pause", and "Step" button, and a dropdown menu set to "Noninverted Tree". The input field contains the string "baaba". Below the input field, a message states "String rejected. 47 nodes generated." There are two sliders: "Input Field Text Size" and "Table Text Size". A table with grammar rules is visible at the bottom left.

LHS		RHS
S	→	SS
S	→	aSa
S	→	b
S	→	aa

Try another string.

You can use the User Control Parse to see why a string cannot be derived. If there is more than one variable in the sentential form, you will need to click on which variable you want to expand in the very bottom window (not in the parse tree)

JFLAP : (cfgExample2.jff)

File Input Test Convert Help

Editor Brute Parser **User Control Parser**

Table Text Size

Start Previous Step **Noninverted Tree**

Input **baaba**

No Additional Production is Possible

Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the ...)

LHS		RHS
S	→	SS
S	→	aSa
S	→	b
S	→	aa

**baab**

Derived current Strings using S→b production

**Definition Simple grammar (or s-grammar) has all productions of the form:**

$$A \rightarrow ax$$

**where  $A \in V$ ,  $a \in T$ , and  $x \in V^*$  AND any pair  $(A,a)$  can occur in at most one rule.**

## Ambiguity

**Definition:** A CFG  $G$  is ambiguous if  $\exists$  some  $w \in L(G)$  which has two distinct derivation trees.

# Example Expression grammar

$G = (\{E, I\}, \{a, b, +, *, (\, )\}, E, P), P =$

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

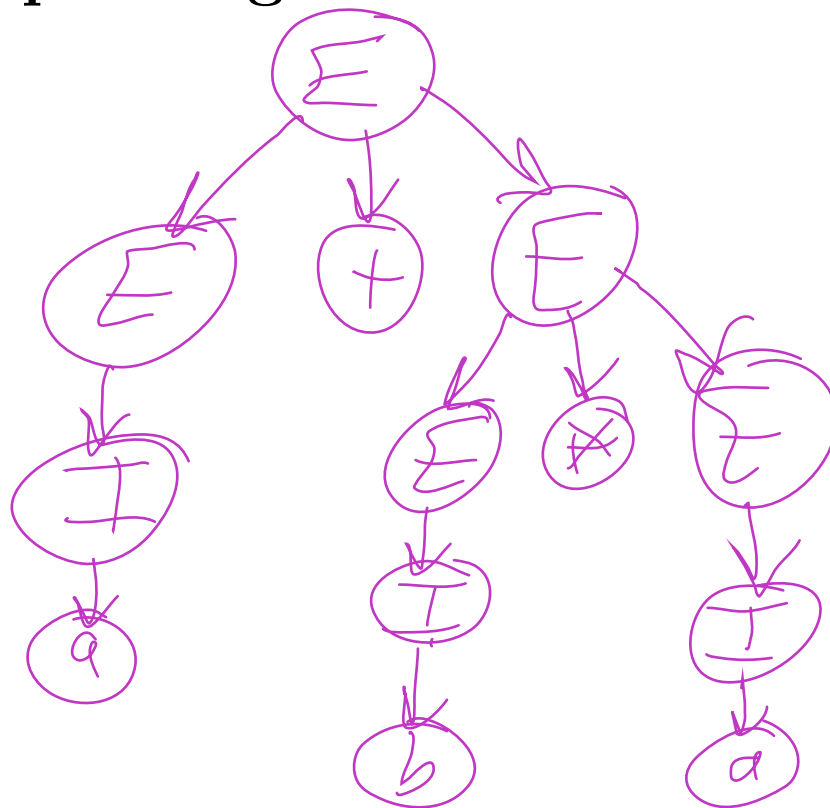
$$I \rightarrow a \mid b$$

*Ambiguous  
grammar  
not  
good*

Derivation of  $a + b * a$  is:

$$E \Rightarrow E + E \Rightarrow I + E \Rightarrow a + E \Rightarrow a + E * E \Rightarrow a + I * E \Rightarrow a + b * E \Rightarrow a + b * I \Rightarrow a + b * a$$

Corresponding derivation tree is:

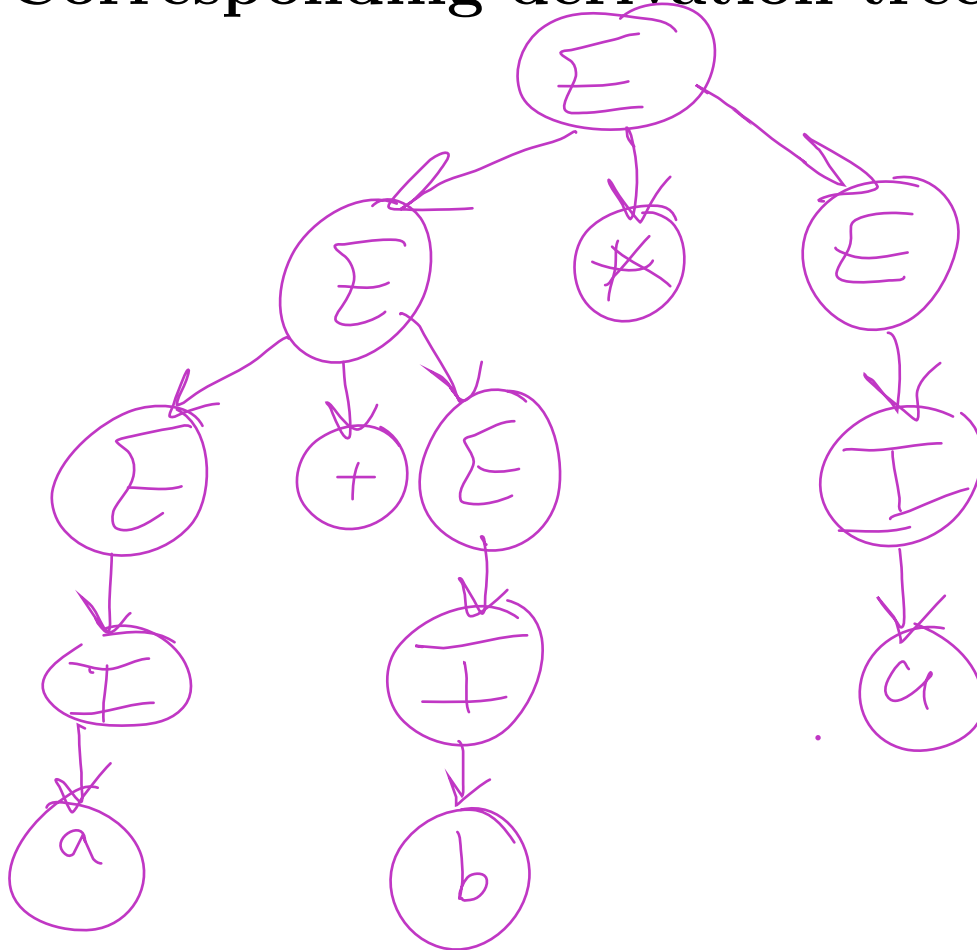


*Probably  
the  
tree  
you  
want  
mult  
higher  
prec*

Another derivation of  $a+b*a$  is:

$E \Rightarrow \underline{E}*E \Rightarrow \underline{E}+\underline{E}*E \Rightarrow \underline{I}+\underline{E}*E \Rightarrow$   
 $a+\underline{E}*E \Rightarrow a+\underline{I}*E \Rightarrow a+b*\underline{E} \Rightarrow a+b*\underline{I} \Rightarrow$   
 $a+b*a$

Corresponding derivation tree is:



Stopped here



Rewrite the grammar as an unambiguous grammar. (with meaning that multiplication has higher precedence than addition)

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

There is only one derivation tree for  $a+b*a$ :

**Definition** If L is CFL and G is an unambiguous CFG s.t.  $L=L(G)$ , then L is unambiguous.

**Backus-Naur Form of a grammar:**

- Nonterminals are enclosed in brackets  $\langle \rangle$
- For “ $\rightarrow$ ” use instead “ $::=$ ”

**Sample C++ Program:**

```
main ()
{
    int a;      int b;      int sum;
    a = 40;     b = 6;      sum = a + b;
    cout << "sum is " << sum << endl;
}
```

“Attempt” to write a CFG for C++ in BNF (Note:  $\langle \text{program} \rangle$  is start symbol of grammar.)

$\langle \text{program} \rangle ::= \text{main} () \langle \text{block} \rangle$   
 $\langle \text{block} \rangle ::= \{ \langle \text{stmt-list} \rangle \}$   
 $\langle \text{stmt-list} \rangle ::= \langle \text{stmt} \rangle \mid \langle \text{stmt} \rangle \langle \text{stmt-list} \rangle$   
 $\langle \text{decl} \rangle ::= \langle \text{decl} \rangle \mid \langle \text{decl} \rangle \langle \text{stmt-list} \rangle$   
 $\langle \text{decl} \rangle ::= \text{int} \langle \text{id} \rangle ; \mid \text{double} \langle \text{id} \rangle ;$   
 $\langle \text{stmt} \rangle ::= \langle \text{asgn-stmt} \rangle \mid \langle \text{cout-stmt} \rangle$   
 $\langle \text{asgn-stmt} \rangle ::= \langle \text{id} \rangle = \langle \text{expr} \rangle ;$   
 $\langle \text{expr} \rangle ::= \langle \text{expr} \rangle + \langle \text{expr} \rangle$   
 $\mid \langle \text{expr} \rangle * \langle \text{expr} \rangle$   
 $\mid ( \langle \text{expr} \rangle ) \mid \langle \text{id} \rangle$   
 $\langle \text{cout-stmt} \rangle ::= \text{cout} \langle \text{out-list} \rangle ;$

etc., Must expand all nonterminals!

So a derivation of the program test would look like:

$$\begin{aligned} \langle \text{program} \rangle &\Rightarrow \text{main } () \langle \text{block} \rangle \\ &\Rightarrow \text{main } () \{ \langle \text{stmt-list} \rangle \} \\ &\Rightarrow \text{main } () \{ \langle \text{decl} \rangle \langle \text{stmt-list} \rangle \} \\ &\Rightarrow \text{main } () \{ \text{int } \langle \text{id} \rangle ; \langle \text{stmt-list} \rangle \} \\ &\Rightarrow \text{main } () \{ \text{int } a ; \langle \text{stmt-list} \rangle \} \\ &\stackrel{*}{\Rightarrow} \textit{complete C++ program} \end{aligned}$$

More on CFG for C++

We can write a CFG  $G$  s.t.

$L(G) = \{\text{syntactically correct C++ programs}\}$ .

But note that  $\{\text{semantically correct C++ programs}\} \subset L(G)$ .

Can't recognize redeclared variables:

```
int x;  
double x;
```

Can't recognize if formal parameters match actual parameters in number and types:

```
declar: int Sum(int a, int b, int c) ...  
call:  newsum = Sum(x,y);
```