

# Programming Languages and Compilers (CS 421)

---

Elsa L Gunter

2112 SC, UIUC

[http://www.cs.uiuc.edu/class  
/sp07/cs421/](http://www.cs.uiuc.edu/class/sp07/cs421/)

Based in part on slides by Mattox Beckman, as updated  
by Vikram Adve and Gul Agha

# Grammars

---

- Grammars are formal descriptions of which strings over a given character set are in a particular language
- Language designers write grammar
- Language implementers use grammar to know what programs to accept
- Language users use grammar to know how to write legitimate programs

# Types of Formal Language Descriptions

---

- Regular expressions, regular grammars
- Context-free grammars, BNF grammars, syntax diagrams
- Finite state automata
  
- Whole family more of grammars and automata – covered in automata theory

# Sample Grammar

---

- Language: Parenthesized sums of 0's and 1's
- $\langle \text{Sum} \rangle ::= 0$
- $\langle \text{Sum} \rangle ::= 1$
- $\langle \text{Sum} \rangle ::= \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$
- $\langle \text{Sum} \rangle ::= (\langle \text{Sum} \rangle)$

# BNF Grammars

---

- Start with a set of characters, **a,b,c,...**
  - We call these *terminals*
- Add a set of different characters, **X,Y,Z,...**
  - We call these *nonterminals*
- One special nonterminal **S** called *start symbol*

# BNF Grammars

---

- BNF rules (aka productions) have form

$$\mathbf{X} ::= y$$

where  $\mathbf{X}$  is any nonterminal and  $y$  is a string of terminals and nonterminals

- BNF grammar is a set of BNF rules such that every nonterminal appears on the left of some rule

# Sample Grammar

---

- Terminals: 0 1 + ( )
- Nonterminals:  $\langle \text{Sum} \rangle$
- Start symbol =  $\langle \text{Sum} \rangle$
  
- $\langle \text{Sum} \rangle ::= 0$
- $\langle \text{Sum} \rangle ::= 1$
- $\langle \text{Sum} \rangle ::= \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$
- $\langle \text{Sum} \rangle ::= (\langle \text{Sum} \rangle)$
- Can be abbreviated as  
 $\langle \text{Sum} \rangle ::= 0 \mid 1 \mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle \mid (\langle \text{Sum} \rangle)$

# BNF Derivations

---

- Given rules

$$\mathbf{X} ::= y\mathbf{Z}w \text{ and } \mathbf{Z} ::= v$$

we may replace  $\mathbf{Z}$  by  $v$  to say

$$\mathbf{X} \Rightarrow y\mathbf{Z}w \Rightarrow yvw$$

- Derivation called *right-most* if always replace the right-most non-terminal

# BNF Derivations

---

- Start with the start symbol:

$\langle \text{Sum} \rangle \Rightarrow$

# BNF Derivations

---

- Pick a non-terminal

$\langle \text{Sum} \rangle \Rightarrow$

# BNF Derivations

---

- Pick a rule and substitute:

–  $\langle \text{Sum} \rangle ::= \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

# BNF Derivations

---

- Pick a non-terminal:

$$\langle \text{Sum} \rangle \Rightarrow \boxed{\langle \text{Sum} \rangle} + \langle \text{Sum} \rangle$$

# BNF Derivations

---

- Pick a rule and substitute:

–  $\langle \text{Sum} \rangle ::= ( \langle \text{Sum} \rangle )$

$\langle \text{Sum} \rangle \Rightarrow \boxed{\langle \text{Sum} \rangle} + \langle \text{Sum} \rangle$

$\Rightarrow \boxed{( \langle \text{Sum} \rangle )} + \langle \text{Sum} \rangle$

# BNF Derivations

---

- Pick a non-terminal:

$$\begin{aligned} \langle \text{Sum} \rangle &\Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle \\ &\Rightarrow ( \boxed{\langle \text{Sum} \rangle} ) + \langle \text{Sum} \rangle \end{aligned}$$

# BNF Derivations

---

- Pick a rule and substitute:

–  $\langle \text{Sum} \rangle ::= \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\Rightarrow ( \boxed{\langle \text{Sum} \rangle} ) + \langle \text{Sum} \rangle$

$\Rightarrow ( \boxed{\langle \text{Sum} \rangle + \langle \text{Sum} \rangle} ) + \langle \text{Sum} \rangle$

# BNF Derivations

---

- Pick a non-terminal:

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle ) + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle + \boxed{\langle \text{Sum} \rangle} ) + \langle \text{Sum} \rangle$

# BNF Derivations

---

- Pick a non-terminal:

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle ) + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle + \boxed{\langle \text{Sum} \rangle} ) + \langle \text{Sum} \rangle$

# BNF Derivations

---

- Pick a rule and substitute:

–  $\langle \text{Sum} \rangle ::= 1$

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle ) + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle + \boxed{\langle \text{Sum} \rangle} ) + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle + \boxed{1} ) + \langle \text{Sum} \rangle$

# BNF Derivations

---

- Pick a non-terminal:

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle ) + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle + \langle \text{Sum} \rangle ) + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle + 1 ) + \boxed{\langle \text{Sum} \rangle}$

# BNF Derivations

---

- Pick a rule and substitute:

–  $\langle \text{Sum} \rangle ::= 0$

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle ) + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle + \langle \text{Sum} \rangle ) + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle + 1 ) + \boxed{\langle \text{Sum} \rangle}$

$\Rightarrow ( \langle \text{Sum} \rangle + 1 ) + \boxed{0}$

# BNF Derivations

---

- Pick a non-terminal:

$$\begin{aligned} \langle \text{Sum} \rangle &\Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle \\ &\Rightarrow ( \langle \text{Sum} \rangle ) + \langle \text{Sum} \rangle \\ &\Rightarrow ( \langle \text{Sum} \rangle + \langle \text{Sum} \rangle ) + \langle \text{Sum} \rangle \\ &\Rightarrow ( \langle \text{Sum} \rangle + 1 ) + \langle \text{Sum} \rangle \\ &\Rightarrow ( \boxed{\langle \text{Sum} \rangle} + 1 ) + 0 \end{aligned}$$

# BNF Derivations

---

- Pick a rule and substitute

–  $\langle \text{Sum} \rangle ::= 0$

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle ) + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle + \langle \text{Sum} \rangle ) + \langle \text{Sum} \rangle$

$\Rightarrow ( \langle \text{Sum} \rangle + 1 ) + \langle \text{Sum} \rangle$

$\Rightarrow ( \boxed{\langle \text{Sum} \rangle} + 1 ) 0$

$\Rightarrow ( \boxed{0} + 1 ) + 0$

# BNF Derivations

---

- $(0 + 1) + 0$  is generated by grammar

$\langle \text{Sum} \rangle \Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$   
 $\Rightarrow ( \langle \text{Sum} \rangle ) + \langle \text{Sum} \rangle$   
 $\Rightarrow ( \langle \text{Sum} \rangle + \langle \text{Sum} \rangle ) + \langle \text{Sum} \rangle$   
 $\Rightarrow ( \langle \text{Sum} \rangle + 1 ) + \langle \text{Sum} \rangle$   
 $\Rightarrow ( \langle \text{Sum} \rangle + 1 ) + 0$   
 $\Rightarrow ( 0 + 1 ) + 0$

# Your Turn:

$\langle \text{Sum} \rangle ::= 0 \mid 1 \mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle \mid (\langle \text{Sum} \rangle)$

---

$\langle \text{Sum} \rangle \Rightarrow$

# BNF Semantics

---

- The meaning of a BNF grammar is the set of all strings consisting only of terminals that can be derived from the Start symbol

# Extended BNF Grammars

---

- Alternatives: allow rules of form  $X ::= y|z$ 
  - Abbreviates  $X ::= y, X ::= z$
- Options:  $X ::= y[v]z$ 
  - Abbreviates  $X ::= yvz, X ::= yz$
- Repetition:  $X ::= y\{v\}^*z$ 
  - Can be eliminated by adding new nonterminal  $V$  and rules  $X ::= yz, X ::= yVz, V ::= v, V ::= vV$

# Regular Grammars

---

- Subclass of BNF
- Only rules of form  
 $\langle \text{nonterminal} \rangle ::= \langle \text{terminal} \rangle \langle \text{nonterminal} \rangle$   
or  $\langle \text{nonterminal} \rangle ::= \langle \text{terminal} \rangle$
- Defines same class of languages as regular expressions
- Important for writing lexers (programs that convert strings of characters into strings of tokens)

# Example

---

- Regular grammar:
  - <Balanced> ::=  $\varepsilon$
  - <Balanced> ::= 0<OneAndMore>
  - <Balanced> ::= 1<ZeroAndMore>
  - <OneAndMore> ::= 1<Balanced>
  - <ZeroAndMore> ::= 0<Balanced>
- Generates even length strings where every initial substring of even length has same number of 0's as 1's

# Parse Trees

---

- Graphical representation of derivation
- Each node labeled with either non-terminal or terminal
- If node is labeled with a terminal, then it is a leaf (no sub-trees)
- If node is labeled with a non-terminal, then it has one branch for each character in the right-hand side of rule used to substitute for it

# Example

---

- Consider grammar:

$\langle \text{exp} \rangle ::= \langle \text{factor} \rangle$

$\quad \quad \quad | \langle \text{factor} \rangle + \langle \text{factor} \rangle$

$\langle \text{factor} \rangle ::= \langle \text{bin} \rangle$

$\quad \quad \quad | \langle \text{bin} \rangle * \langle \text{exp} \rangle$

$\langle \text{bin} \rangle ::= 0 \quad | \quad 1$

- Problem: Build parse tree for  $1 * 1 + 0$   
as an  $\langle \text{exp} \rangle$

# Example cont.

---

- $1 * 1 + 0$ :  $\langle \text{exp} \rangle$

$\langle \text{exp} \rangle$  is the start symbol for this parse tree

# Example cont.

---

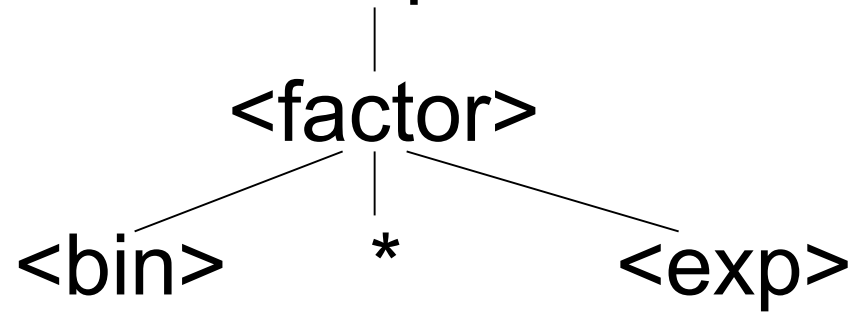
- $1 * 1 + 0$ :  $\langle \text{exp} \rangle$   
|  
 $\langle \text{factor} \rangle$

Use rule:  $\langle \text{exp} \rangle ::= \langle \text{factor} \rangle$

# Example cont.

---

- $1 * 1 + 0$ :  $\langle \text{exp} \rangle$

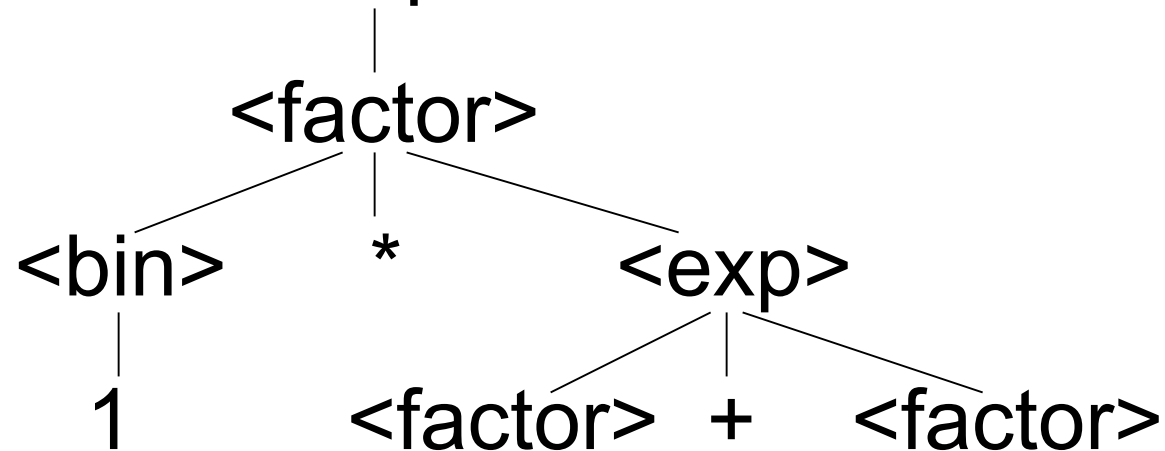


Use rule:  $\langle \text{factor} \rangle ::= \langle \text{bin} \rangle * \langle \text{exp} \rangle$

# Example cont.

---

- $1 * 1 + 0$ :  $\langle \text{exp} \rangle$

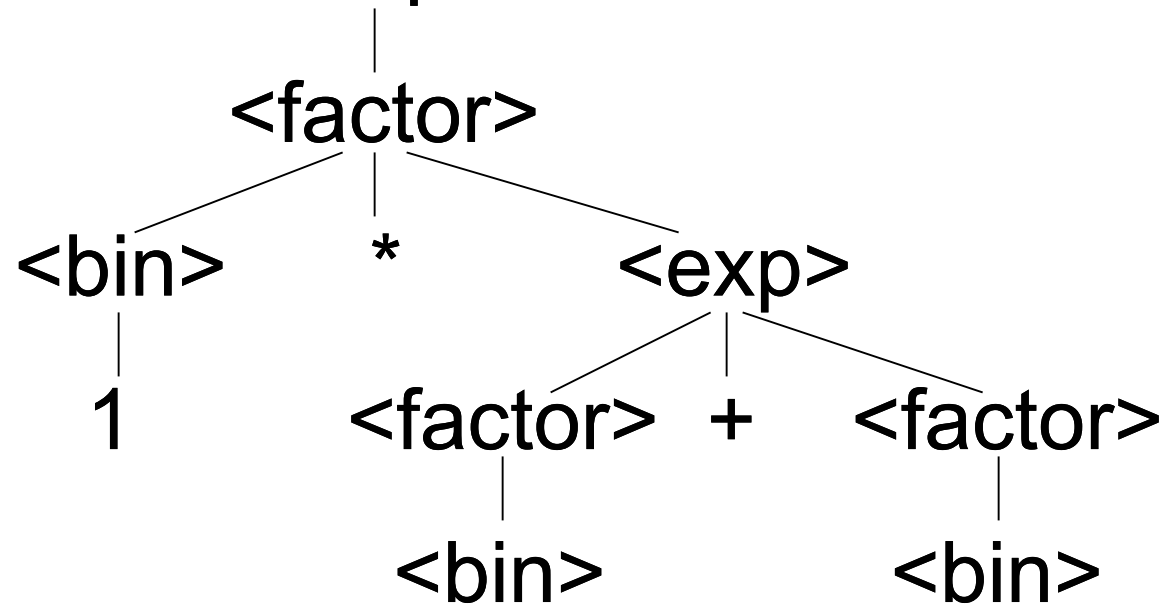


Use rules:  $\langle \text{bin} \rangle ::= 1$  and  
 $\langle \text{exp} \rangle ::= \langle \text{factor} \rangle + \langle \text{factor} \rangle$

# Example cont.

---

- $1 * 1 + 0$ :  $\langle \text{exp} \rangle$

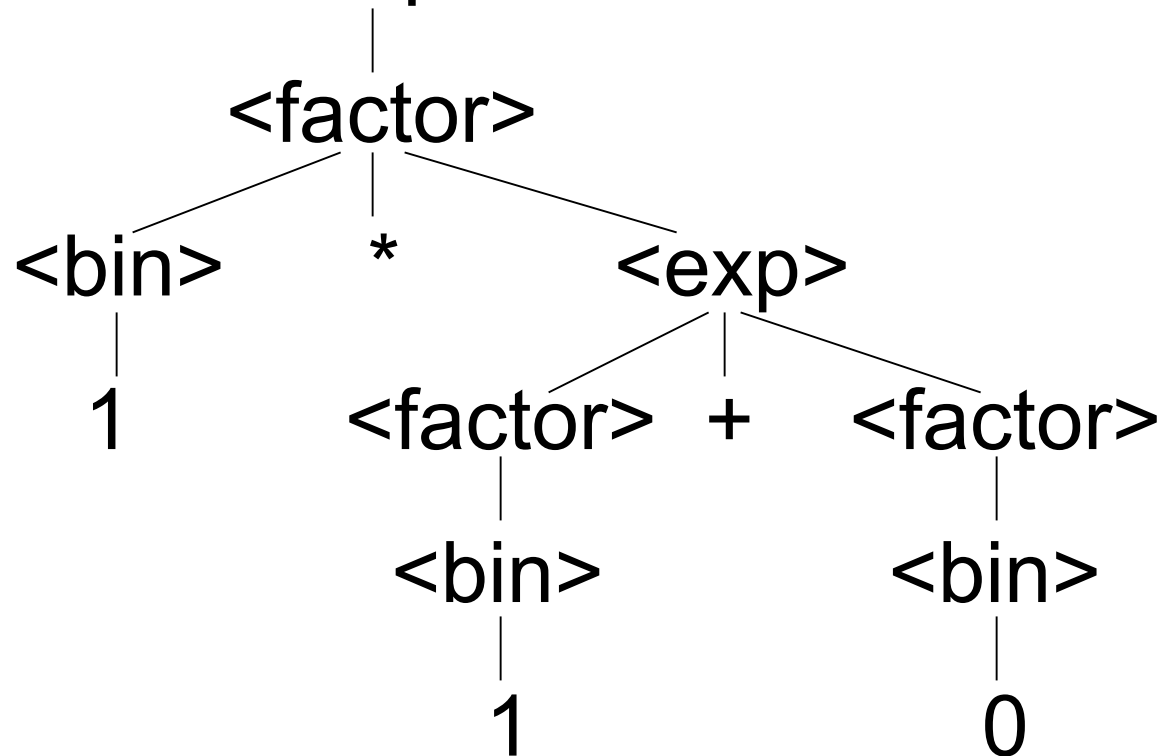


Use rule:  $\langle \text{factor} \rangle ::= \langle \text{bin} \rangle$

# Example cont.

---

- $1 * 1 + 0$ :  $\langle \text{exp} \rangle$

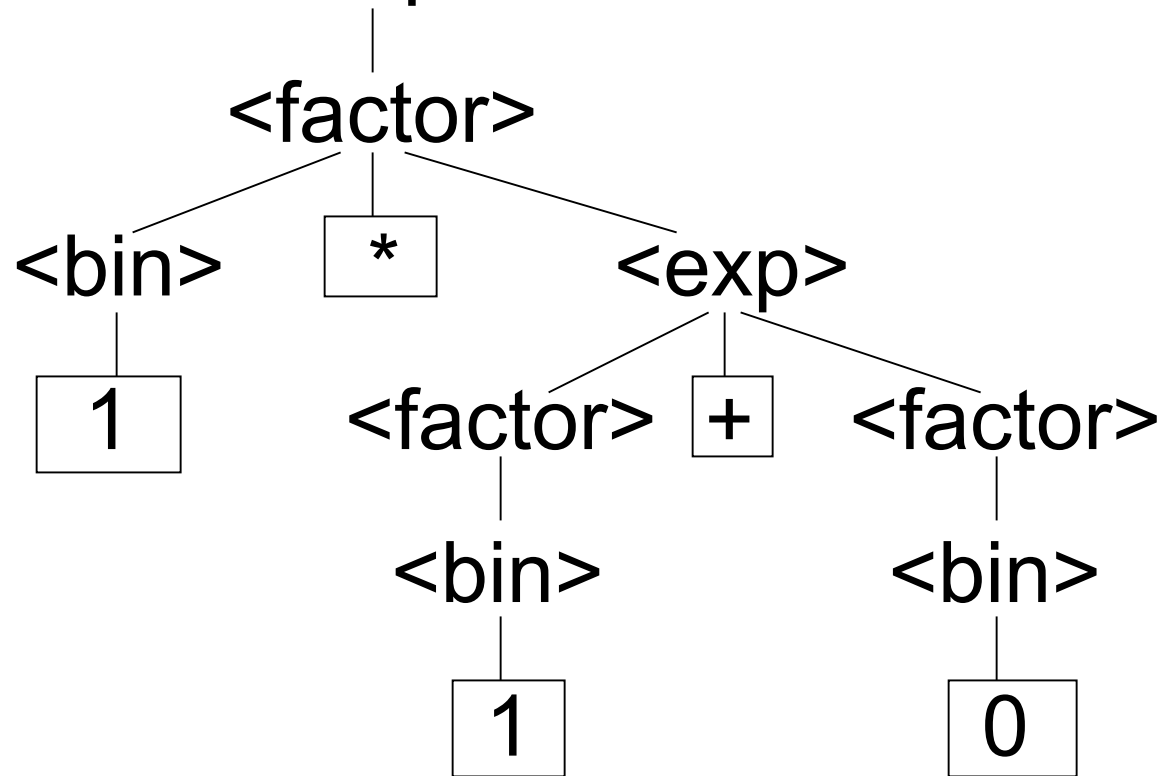


Use rules:  $\langle \text{bin} \rangle ::= 1 \mid 0$

# Example cont.

---

- $1 * 1 + 0$ :  $\langle \text{exp} \rangle$



Fringe of tree is string generated by grammar

Your Turn:  $1 * 0 + 0 * 1$

---

# Parse Tree Data Structures

---

- Parse trees may be represented by SML datatypes
- One datatype for each nonterminal
- One constructor for each rule
- Defined as mutually recursive collection of datatype declarations

# Example

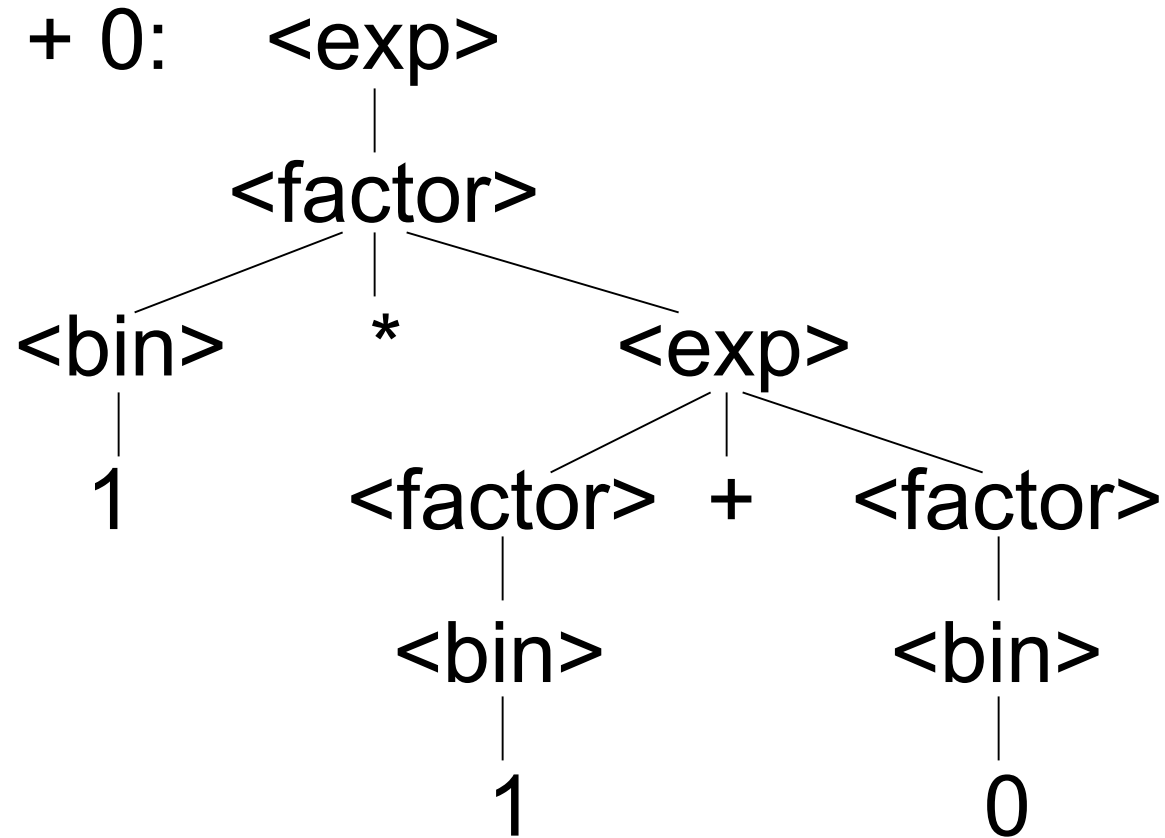
---

- Recall grammar:  
     $\langle \text{exp} \rangle ::= \langle \text{factor} \rangle \mid \langle \text{factor} \rangle + \langle \text{factor} \rangle$   
     $\langle \text{factor} \rangle ::= \langle \text{bin} \rangle \mid \langle \text{bin} \rangle * \langle \text{exp} \rangle$   
     $\langle \text{bin} \rangle ::= 0 \mid 1$
- datatype exp = Factor2Exp of factor  
                  | Plus of factor \* factor  
and factor = Bin2Factor of bin  
                  | Mult of bin \* exp  
and bin = Zero | One

# Example cont.

---

- $1 * 1 + 0$ :



# Example cont.

---

- Can be represented as

Factor2Exp

(Mult(One,

Plus(Bin2Factor One,

Bin2Factor Zero)))

# Ambiguous Grammars and Languages

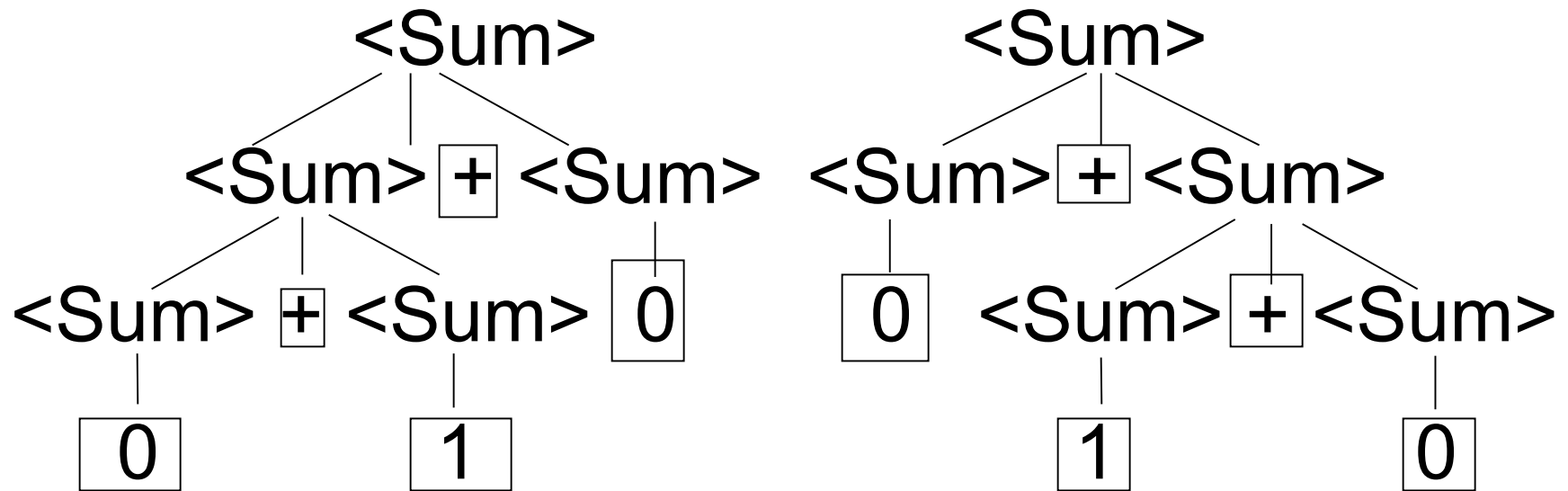
---

- A BNF grammar is *ambiguous* if its language contains strings for which there is more than one parse tree
- If all BNF's for a language are ambiguous then the language is *inherently ambiguous*

# Example: Ambiguous Grammar

---

- $0 + 1 + 0$



# Example

---

- What is the result for:

$$3 + 4 * 5 + 6$$

# Example

---

- What is the result for:

$$3 + 4 * 5 + 6$$

- Possible answers:

- $41 = ((3 + 4) * 5) + 6$

- $47 = 3 + (4 * (5 + 6))$

- $29 = (3 + (4 * 5)) + 6 = 3 + ((4 * 5) + 6)$

- $77 = (3 + 4) * (5 + 6)$

# Example

---

- What is the value of:

$$7 - 5 - 2$$

# Example

---

- What is the value of:

$$7 - 5 - 2$$

- Possible answers:

- In Pascal, C++, SML assoc. left

$$7 - 5 - 2 = (7 - 5) - 2 = 0$$

- In APL, associate to right

$$7 - 5 - 2 = 7 - (5 - 2) = 4$$

# Two Major Sources of Ambiguity

---

- Lack of determination of operator precedence
- Lack of determination of operator associativity
- Not the only sources of ambiguity

# How to Enforce Associativity

---

- Have at most one recursive call per production
- When two or more recursive calls would be natural leave right-most one for right associativity, left-most one for left associativity

# Example

---

- $\langle \text{Sum} \rangle ::= 0 \mid 1 \mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$   
 $\mid (\langle \text{Sum} \rangle)$
- **Becomes**
  - $\langle \text{Sum} \rangle ::= \langle \text{Num} \rangle \mid \langle \text{Num} \rangle + \langle \text{Sum} \rangle$
  - $\langle \text{Num} \rangle ::= 0 \mid 1 \mid (\langle \text{Sum} \rangle)$

# Operator Precedence

---

- Operators of highest precedence evaluated first (bind more tightly).
- Precedence for infix binary operators given in following table
- Needs to be reflected in grammar

# Precedence Table - Sample

	Fortran	Pascal	C/C++	Ada	SML
highest	**	*, /, div, mod	++, --	**	div, mod, /, *
	*, /	+, -	*, /, %	*, /, mod	+, -, ^
	+, -		+, -	+, -	::

# First Example Again

---

- In any above language,  $3 + 4 * 5 + 6 = 29$
- In APL, all infix operators have same precedence
  - Thus we still don't know what the value is (handled by associativity)
- How do we handle precedence in grammar?

# Precedence in Grammar

---

- Higher precedence translates to longer derivation chain
- Example:

$$\langle \text{exp} \rangle ::= \langle \text{id} \rangle \mid \langle \text{exp} \rangle + \langle \text{exp} \rangle \\ \mid \langle \text{exp} \rangle * \langle \text{exp} \rangle$$

- Becomes

$$\langle \text{exp} \rangle ::= \langle \text{mult\_exp} \rangle \\ \mid \langle \text{exp} \rangle + \langle \text{mult\_exp} \rangle$$
$$\langle \text{mult\_exp} \rangle ::= \langle \text{id} \rangle \mid \langle \text{mult\_exp} \rangle * \langle \text{id} \rangle$$