

# Announcements

- **Check course web page often**  
<http://remus.rutgers.edu/cs314>
- **First homework will be on webpage soon...**
  - spot checked
  - important for exams
- **Additional office hours – email me**

1

# Grammars

**S** → NP VP

**NP** → Name | Det Noun

**VP** → Verb | Verb NP

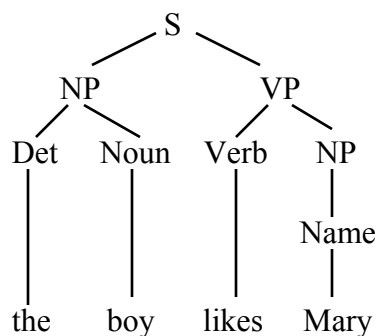
**Name** → john | mary

**Det** → a | the

**Det** → some | every

**Noun** → boy | girl

**Verb** → runs | likes



2

# Grammars

A grammar  $G$ , is a quadruple  $\langle T, N, P, S \rangle$ , where

- $T$  is a set of terminal symbols
  - e.g., john, mary, a, the, some, every, boy, girl, runs, likes
- $N$  is a set of nonterminal symbols
  - e.g., S, NP, VP, Name, Det, Noun, Verb
- $P$  is a set of productions, or rewrite rules; and
- $S$  is a special start symbol.

The language of  $G$ ,  $L(G)$ , is the set of all terminal sequences that can be produced by applying the rewrite rules, repeatedly, starting with  $S$ .

3

# Grammars

- For Programming Languages:

Stmt  $\rightarrow$  Identifier = 0

Identifier  $\rightarrow$  Letter | Identifier Letter | Identifier Digit

Letter  $\rightarrow$  a | b | c | ... | x | y | z

Digit  $\rightarrow$  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

- Backus-Naur Form (BNF):

$\langle \text{stmt} \rangle ::= \langle \text{id} \rangle = 0$

$\langle \text{id} \rangle ::= \langle \text{letter} \rangle | \langle \text{id} \rangle \langle \text{letter} \rangle | \langle \text{id} \rangle \langle \text{digit} \rangle$

$\langle \text{letter} \rangle ::= a | b | c | \dots | x | y | z$

$\langle \text{digit} \rangle ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$

4

## Backus-Naur form (BNF)

- Metasymbols < > ::= |
  - Terminal symbols of the PL
    - e.g., keywords, operators
  - Nonterminal symbols
- 

$\langle \textit{while\_stmt} \rangle ::= \textit{while} \langle \textit{expr} \rangle \textit{do} \langle \textit{stmt} \rangle$   
 $\langle \textit{identifier} \rangle ::= \langle \textit{letter} \rangle \mid \langle \textit{identifier} \rangle \langle \textit{digit} \rangle \mid$   
 $\quad \langle \textit{identifier} \rangle \langle \textit{letter} \rangle$

5

## BNF Examples

- *letter* (*letter* | *digit*)<sup>\*</sup>  
 $\langle \textit{id} \rangle ::= \langle \textit{letter} \rangle \mid \langle \textit{id} \rangle \langle \textit{letter} \rangle \mid \langle \textit{id} \rangle \langle \textit{digit} \rangle$
- *digit*<sup>\*</sup>  
 $\langle \textit{integer} \rangle ::= \langle \textit{integer} \rangle \langle \textit{digit} \rangle \mid \langle \textit{digit} \rangle \mid \varepsilon$
- Strings of 1's and 0's where all 1's come before all 0's, that is,  $1^*0^*$   
 $\langle \textit{str} \rangle ::= \langle \textit{one} \rangle \langle \textit{zero} \rangle$   
 $\langle \textit{one} \rangle ::= 1 \langle \textit{one} \rangle \mid 1 \mid \varepsilon$   
 $\langle \textit{zero} \rangle ::= 0 \langle \textit{zero} \rangle \mid 0 \mid \varepsilon$
- If statement  
 $\langle \textit{if\_stmt} \rangle ::= \textit{if} \langle \textit{expr} \rangle \textit{then} \langle \textit{statement} \rangle \textit{else} \langle \textit{statement} \rangle$

6

## Extended BNF (EBNF)

- Nonterminals begin with capital letters or are shown in a different font
- {...} means repeat the enclosed 0 or more times
- [...] means the enclosed is optional
- (...) is used for grouping, usually with the alternation symbol |
- If { }, [ ], or ( ) are terminals in the PL being defined, then when they are used as terminals they must be underlined (Louden uses quotes)
  - { } metasyms
  - { } terminals “{” “}”

7

## EBNF Examples

**Identifier ::= Letter { LetterorDigit }**  
**LetterorDigit ::= Letter | Digit**  
**Expr ::= [ Expr - ] Subexpr**  
**IfStmt ::= if LogicExpr then Stmt [else Stmt]**  
**CompoundStmt ::= begin Stmt { ; Stmt } end**  
**WhileStmt ::= while ( LogicExpr ) Stmt { ; Stmt }**  
**ArrayElement ::= Identifier [ Identifier ]**

8

## Regular grammars

Productions take one of the following two forms:

- **Left-linear regular grammars:**
  - $A \rightarrow a$
  - $A \rightarrow A a$
- **Right-linear regular grammars**
  - $A \rightarrow a$
  - $A \rightarrow a A$
- **No mixing left/right -- not allowed:  $S \rightarrow a S b$**
- **Cannot generate the language**  
 $\{ a^n b^n \mid n = 1, 2, 3, \dots \}$

9

## Regular grammars

- **Describe the simple constructs in real PLs**
- **All strings describable by regular expressions can be written as regular grammars**
- **Also can be written in Backus-Naur Form,**  
e.g.,  $1 2^* | 0^+$ 
  - $N ::= X | Y$
  - $X ::= 1 | X 2$
  - $Y ::= 0 | Y 0$

10

## Formal Languages - 2

- Context-free PLs
- Grammars
  - Derivation
  - Ambiguity
  - Precedence and Associativity
- Parse trees

11

## Context-free grammars

- Every production has a single nonterminal on the left-hand side.
- Right-hand side is flexible
  - a mixture of terminals and nonterminals

i.e.  $A \rightarrow \varepsilon$   
 $A \rightarrow a A b$

- Not allowed:  $X A Y \rightarrow X a Y$
- Cannot generate the language  
 $\{ a^n b^n c^n \mid n = 1,2,3, \dots \}$

12

## Context-free PLs

- Describe most of the constructs in real PLs
- PLs describable by context-free grammars are recognized by push-down automata (analogous to an FSA with a stack)

13

## Context-free PLs

- **Derivation:** Can be used to generate correct sentences in the PL
- **Parse:** Can be used to recognize syntactically correct sentences in the PL
  - Can be automated efficiently in a compiler (using an algorithm called *LR parsing*)
  - insufficient to describe all constructs of a real PL
    - e.g., type checking from declaration

14

# Derivation

- G
- 1  $\langle \text{letter} \rangle ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z$
  - 2  $\langle \text{digit} \rangle ::= 0|1|2|3|4|5|6|7|8|9$
  - 3  $\langle \text{identifier} \rangle ::= \langle \text{letter} \rangle | \langle \text{identifier} \rangle \langle \text{letter} \rangle | \langle \text{identifier} \rangle \langle \text{digit} \rangle$
  - 4  $\langle \text{assign-stmt} \rangle ::= \langle \text{identifier} \rangle = 0$

- Can we generate  $x2 = 0$  from these rules?

$\langle \text{assign-stmt} \rangle \rightarrow 4 \quad \langle \text{identifier} \rangle = 0$   
 $\rightarrow 3c \quad \langle \text{identifier} \rangle \langle \text{digit} \rangle = 0$   
 $\rightarrow 3a \quad \langle \text{letter} \rangle \langle \text{digit} \rangle = 0$   
 $\rightarrow 1 \quad x \langle \text{digit} \rangle = 0$   
 $\rightarrow 2 \quad x 2 = 0$

YES!

This is a *derivation* of a sentence in the language described by the grammar above.

15

# Derivation

$\langle \text{assign-stmt} \rangle \rightarrow 4 \quad \langle \text{identifier} \rangle = 0$   
 $\rightarrow 3c \quad \langle \text{identifier} \rangle \langle \text{digit} \rangle = 0$   
 $\rightarrow 3a \quad \langle \text{letter} \rangle \langle \text{digit} \rangle = 0$   
 $\rightarrow 1 \quad x \langle \text{digit} \rangle = 0$   
 $\rightarrow 2 \quad x 2 = 0$

- Sentences derivable from the start symbol are called *sentential forms* of G (each sequence in this derivation is one)
- This is a *leftmost* or *canonical derivation*.
  - At each step, the rule indicated is used to substitute the rhs of the rule for the leftmost nonterminal in the sentential form.

16

# Parse

- 1  $\langle \text{letter} \rangle ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z$
- 2  $\langle \text{digit} \rangle ::= 0|1|2|3|4|5|6|7|8|9$
- 3  $\langle \text{identifier} \rangle ::= \langle \text{letter} \rangle | \langle \text{identifier} \rangle \langle \text{letter} \rangle | \langle \text{identifier} \rangle \langle \text{digit} \rangle$
- 4  $\langle \text{assign-stmt} \rangle ::= \langle \text{identifier} \rangle = 0$

Can we recognize  $x2 = 0$  as belonging to this PL?

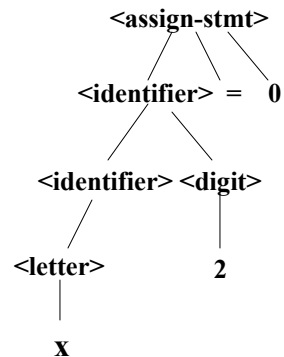
$x2 = 0$	$\rightarrow$	$\langle \text{letter} \rangle 2 = 0$	rule 1
	$\rightarrow$	$\langle \text{identifier} \rangle 2 = 0$	rule 3a
	$\rightarrow$	$\langle \text{identifier} \rangle \langle \text{digit} \rangle = 0$	rule 2
	$\rightarrow$	$\langle \text{identifier} \rangle = 0$	rule 3c
	$\rightarrow$	$\langle \text{assign-stmt} \rangle$	rule 4

This is a *parse* of the sentence  $x2 = 0$ .

17

# Parse Tree

$x2 = 0$	$\rightarrow$	$\langle \text{letter} \rangle 2 = 0$	rule 1
	$\rightarrow$	$\langle \text{identifier} \rangle 2 = 0$	rule 3a
	$\rightarrow$	$\langle \text{identifier} \rangle \langle \text{digit} \rangle = 0$	rule 2
	$\rightarrow$	$\langle \text{identifier} \rangle = 0$	rule 3c
	$\rightarrow$	$\langle \text{assign-stmt} \rangle$	rule 4



In parse tree, each internal node is a nonterminal; its children are the rhs of a rule for that nonterminal.

18

# Grammars are not Unique

- Consider a grammar  $G'$ :

$\langle \text{stmt} \rangle ::= \langle \text{id} \rangle = 0$

$\langle \text{id} \rangle ::= \langle \text{letter} \rangle | \langle \text{id} \rangle \langle \text{char} \rangle$

$\langle \text{char} \rangle ::= \langle \text{letter} \rangle | \langle \text{digit} \rangle$

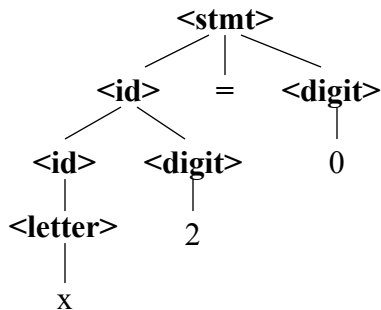
$\langle \text{letter} \rangle ::= a | b | c | \dots | x | y | z$

$\langle \text{digit} \rangle ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$

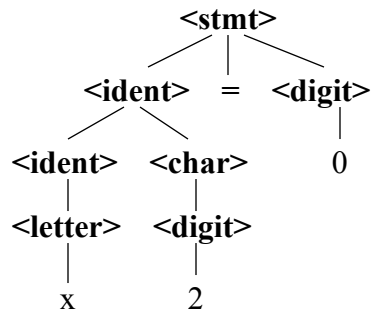
- The grammar  $G'$  generates the same language as  $G$ , but it has different parse trees.

19

# Grammars are not Unique



Parse Tree for  $G$



Parse Tree for  $G'$

20

## Definitions - Review

- ***Grammar***
  - a formalism that describes which sequences of terminals are meaningful in a PL
  - <finite set of terminals, nonterminals, production rules, special symbol>
- ***Context-free grammar***
  - corresponds to PLs whose rules have only 1 nonterminal on the lhs

21

## Definitions

- ***Sentence***
  - a finite sequence of terminals, constructed according to the rules of the grammar for that PL
- ***Sentential form***
  - a finite sequence of terminals and nonterminals, constructed according to the rules of the grammar for that PL
- ***Derivation***
- ***Parse***

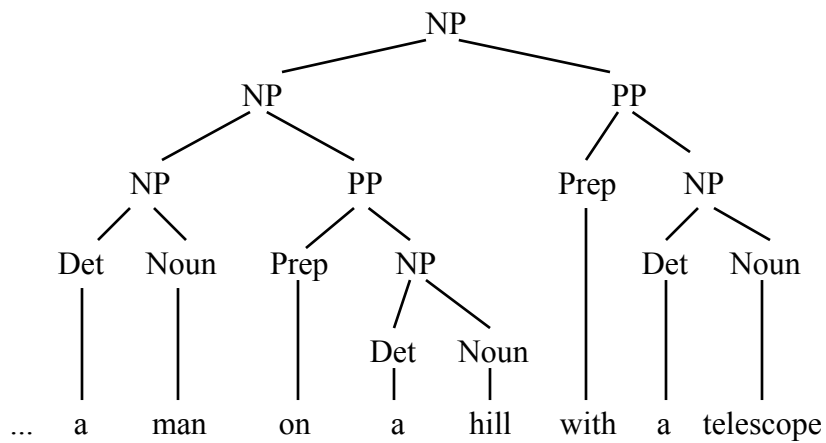
22

# Ambiguity

**I saw a man on a hill with a telescope**

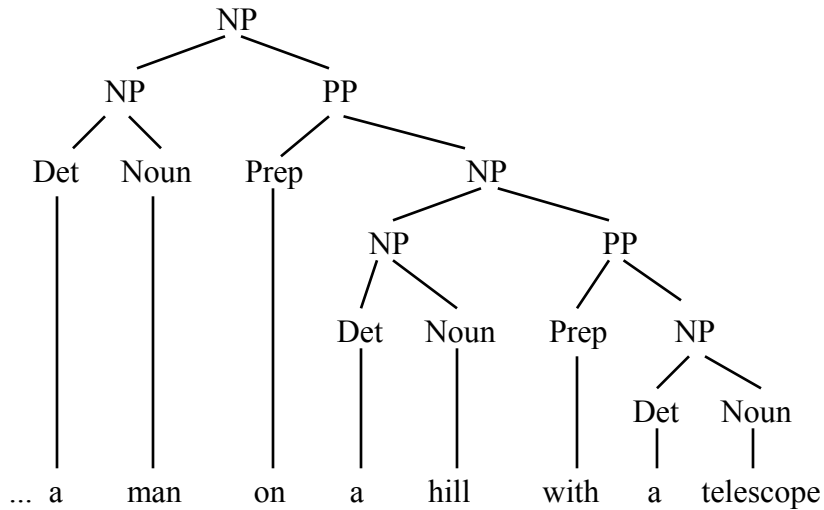
23

# Ambiguity



24

# Ambiguity



25

# Arithmetic Expressions

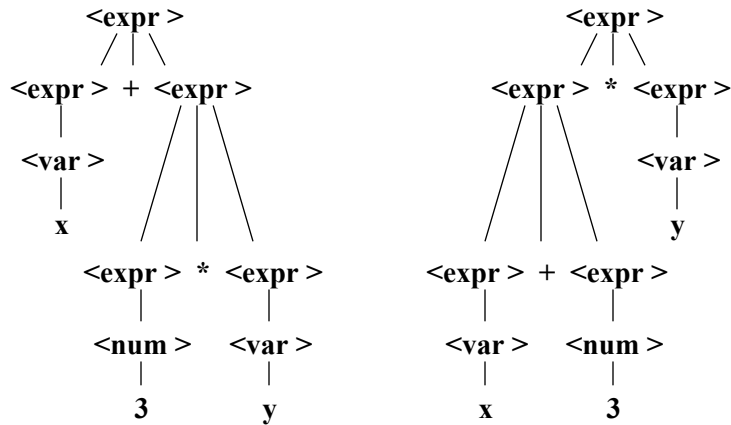
**Here is a grammar for arithmetic expressions:**

```
<expr> ::= <expr> + <expr> | <expr> - <expr> |
          <expr> * <expr> | <expr> / <expr> |
          <var> | <num>
<var> ::= a | b | c | ... | x | y | z
<num> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

**Using this grammar, how would we parse:  $x + 3 * y$  ?**

26

## Two Parse Trees



27

## Ambiguity

- **Ambiguity**
  - If there are 2 different canonical derivations (or alternatively, 2 parse trees) for the same sentence then the grammar is *ambiguous*
  - There is no algorithm which can tell if an arbitrary context-free grammar is ambiguous
  - **Solution**
    - Change grammar to reflect operator precedences  
 $X * Y - Z$  means  $((X * Y) - Z)$

28

# Precedence

Modify the grammar to add precedence:

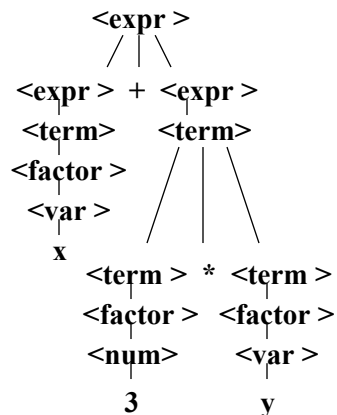
$\langle \text{expr} \rangle ::= \langle \text{expr} \rangle + \langle \text{expr} \rangle \mid \langle \text{expr} \rangle - \langle \text{expr} \rangle \mid \langle \text{term} \rangle$   
 $\langle \text{term} \rangle ::= \langle \text{term} \rangle * \langle \text{term} \rangle \mid \langle \text{term} \rangle / \langle \text{term} \rangle \mid \langle \text{factor} \rangle$   
 $\langle \text{factor} \rangle ::= \langle \text{var} \rangle \mid \langle \text{num} \rangle \mid ( \langle \text{expr} \rangle )$   
 $\langle \text{var} \rangle ::= a \mid b \mid c \mid \dots \mid x \mid y \mid z$   
 $\langle \text{num} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Using this grammar, how would we parse:  $x + 3 * y$  ?

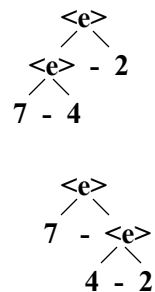
Using this grammar, how would we parse:  $7 - 4 - 2$  ?

29

# Only One Parse Tree



But there are two parse trees for the second example:



30

# Associativity

Modify the grammar to add associativity:

$\langle \text{expr} \rangle ::= \langle \text{expr} \rangle + \langle \text{term} \rangle \mid \langle \text{expr} \rangle - \langle \text{term} \rangle \mid$   
 $\langle \text{term} \rangle$

$\langle \text{term} \rangle ::= \langle \text{term} \rangle * \langle \text{factor} \rangle \mid \langle \text{term} \rangle / \langle \text{factor} \rangle \mid$   
 $\langle \text{factor} \rangle$

$\langle \text{factor} \rangle ::= \langle \text{var} \rangle \mid \langle \text{num} \rangle \mid ( \langle \text{expr} \rangle )$

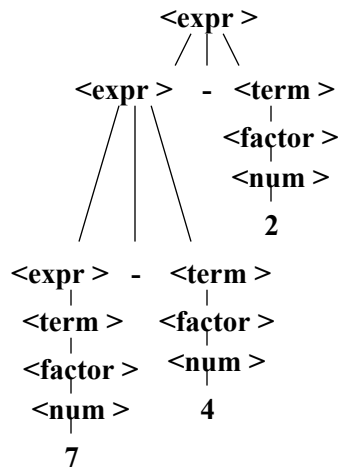
$\langle \text{var} \rangle ::= a \mid b \mid c \mid \dots \mid x \mid y \mid z$

$\langle \text{num} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Using this grammar, how would we parse: 7 - 4 - 2 ?

31

# Only One Parse Tree

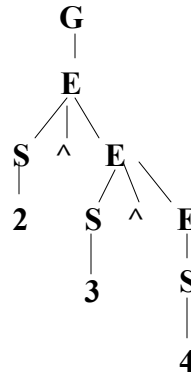


32

# Right Associativity

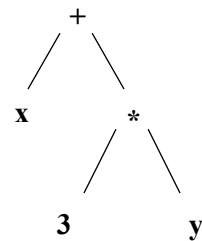
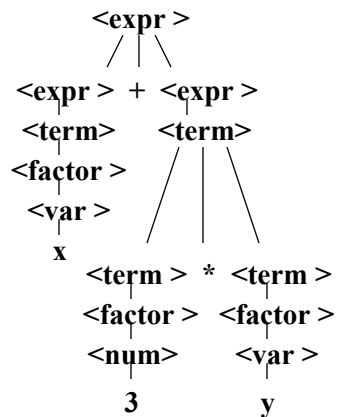
$G ::= E$   
 $E ::= S \wedge E \mid S$   
 $S ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

*What is  $2 \wedge 3 \wedge 4$ ?*  
 *$8^4$  or  $2^{81}$ ?*



33

# Concrete vs. Abstract Syntax



Abstract syntax

34