

# Backus-Naur Form

COMP360

*“Metaphors, similes, puns - all manner of metonymy - I'm interested in language that cannot be parsed by a machine - language that can only be understood through acculturation.”*

Joshua Cohen

# Homework

- Two assignments are posted on Blackboard
  1. Answer 9 questions that include FSA and Mealy machine parse tables
    - Due by noon on Monday, February 6, 2017
  2. Write a parser for the Snowflake language
    - Due by noon on Friday, February 10, 2017

# Syntactical Analysis

- The Syntactical Analysis stage or Parsing stage of a compiler accepts the list of tokens created by lexical scanner and creates a parse tree for use by the semantic analyzer
- This portion of a compiler detects most of the syntax errors
- We assume that our language can be defined as a context free language

# Backus-Naur Form

- Context Free Grammars can be defined using the Backus-Naur Form or **BNF**
- Created by John Backus and Peter Naur.  
Independently created by Noam Chomsky
- BNF is a **metalanguage**, a language used to describe a language

# BNF Fundamentals

- Terminal symbols are tokens or symbols in the language. They come from the lexical scanner
- Nonterminal symbols are “variables” that represent patterns of terminals and nonterminal symbols
- A BNF consists of a set of **productions** or **rules**
- A language description is called a **grammar**

# BNF Structure

- Nonterminal symbols are sometimes enclosed in brackets to differentiate them from terminal symbols
- I will use **bold font** for terminal symbols and no brackets
- Productions are of the form:  
nonterminal  $\rightarrow$  nonterminals or terminals
- Multiple definitions are separated by | meaning OR  
whatever  $\rightarrow$  this | that

# BNF to Define a Language

- There must be one nonterminal start symbol which must appear at least once on the left hand side of a rule
- The start symbol defines the language and all other rules follow from it



# BNF Example

wloop → **while** ( logical ) stmt

logical → exp relation exp

relation → > | < | == | !=

stmt → *something not defined here*

exp → *something not defined here*

# Recursive Lists

- BNF rules can be recursive generating arbitrarily long statements

stuff  $\rightarrow$  thing | thing , stuff

- Following this rule, stuff can be:

thing *or* thing, thing *or* thing, thing, thing, thing, thing

## A BNF is

- A. A language to describe languages
- B. An algorithm for parsing
- C. A computer language for string processing
- D. A silly idea Dr. Williams dreamed up

## Try It

- Write a BNF to describe a language that allows

:f(name)

:s(name)

:(name)

:s(name)f(name)

# Possible Solution

choice → : always | : good | : fail | : good fail

always → ( **name** )

good → **s( name )**

fail → **f( name )**

# Another Possible Solution

choice  $\rightarrow$  : always | : **f** always | : **s** always |

: **f** always **s** always

always  $\rightarrow$  ( **name** )

# Recognizers and Generators

- A BNF can be used to generate an example of the language
- A string is in a language if and only if it can be generated by the language
- There are programming techniques that can determine if a string is a correct example of a language

# BNF for Assignment Statement

1.  $\text{assign} \rightarrow \text{var} = \text{exp}$
2.  $\text{var} \rightarrow \mathbf{X} \mid \mathbf{Y} \mid \mathbf{Z}$
3.  $\text{exp} \rightarrow \text{var} + \text{exp}$
4.            $\mid \text{var} * \text{exp}$
5.            $\mid (\text{exp})$
6.            $\mid \text{var}$



# Derivations

- A derivation is an example of the language created by successively applying rules
- Derivations always begin with the start nonterminal symbol
- Rules are applied left to right

# Deriving $X = Z * (X + Y)$

1 var = exp

2 **X** = exp

4 **X** = var \* exp

2 **X** = **Z** \* exp

5 **X** = **Z** \* ( exp )

3 **X** = **Z** \* ( var + exp )

2 **X** = **Z** \* ( **X** + exp )

6 **X** = **Z** \* ( **X** + var )

2 **X** = **Z** \* ( **X** + **Y** )

1. assign  $\rightarrow$  var = exp

2. var  $\rightarrow$  **X** | **Y** | **Z**

3. exp  $\rightarrow$  var + exp

4. | var \* exp

5. | ( exp )

6. | var

# Write a derivation

- Show a derivation for  $Z = Y + X * Z$

1. assign  $\rightarrow$  var = exp
2. var  $\rightarrow$  **X** | **Y** | **Z**
3. exp  $\rightarrow$  var + exp
4. | var \* exp
5. | ( exp )
6. | var

# Possible Solution

- Show a derivation for  $Z = Y + X * Z$

**1** var = exp

**2** Z = exp

**3** Z = var + exp

**2** Z = Y + exp

**4** Z = Y + var \* exp

**2** Z = Y + X \* exp

**6** Z = Y + X \* var

**2** Z = Y + X \* Z

1. assign  $\rightarrow$  var = exp

2. var  $\rightarrow$  X | Y | Z

3. exp  $\rightarrow$  var + exp

4. | var \* exp

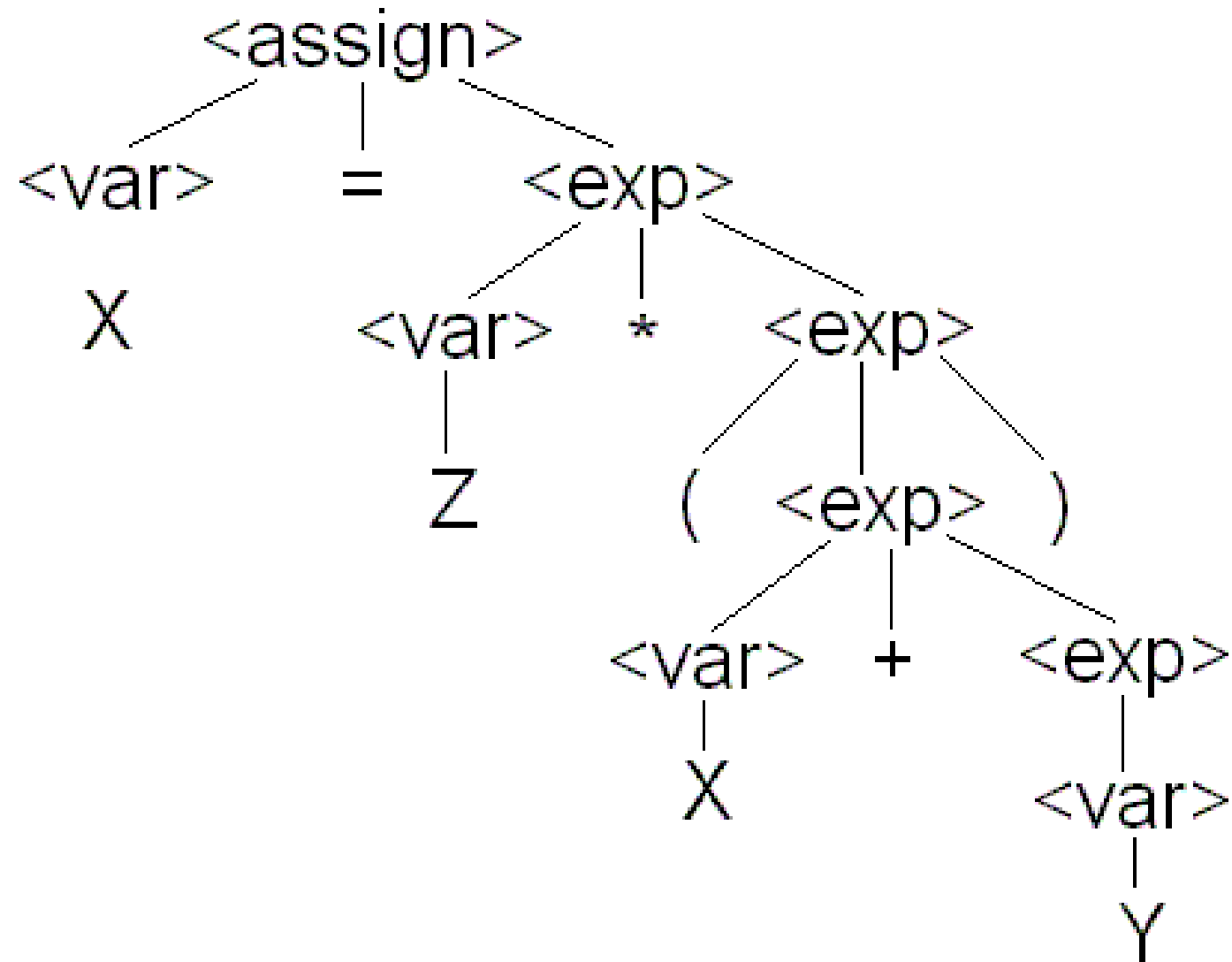
5. | ( exp )

6. | var

# Parse Tree

- A parse tree graphically shows a derivation
- Each nonterminal in the tree has branches representing the symbols on the right side of a rule defining that nonterminal
- All of the leaves of a parse tree are terminal symbols

# Derivation Tree



# Ambiguous Grammars

- For each string in a language, there must be one and only one derivation or parse tree that creates that string
- A grammar is ambiguous if the same member of the language can be created by two or more different derivations

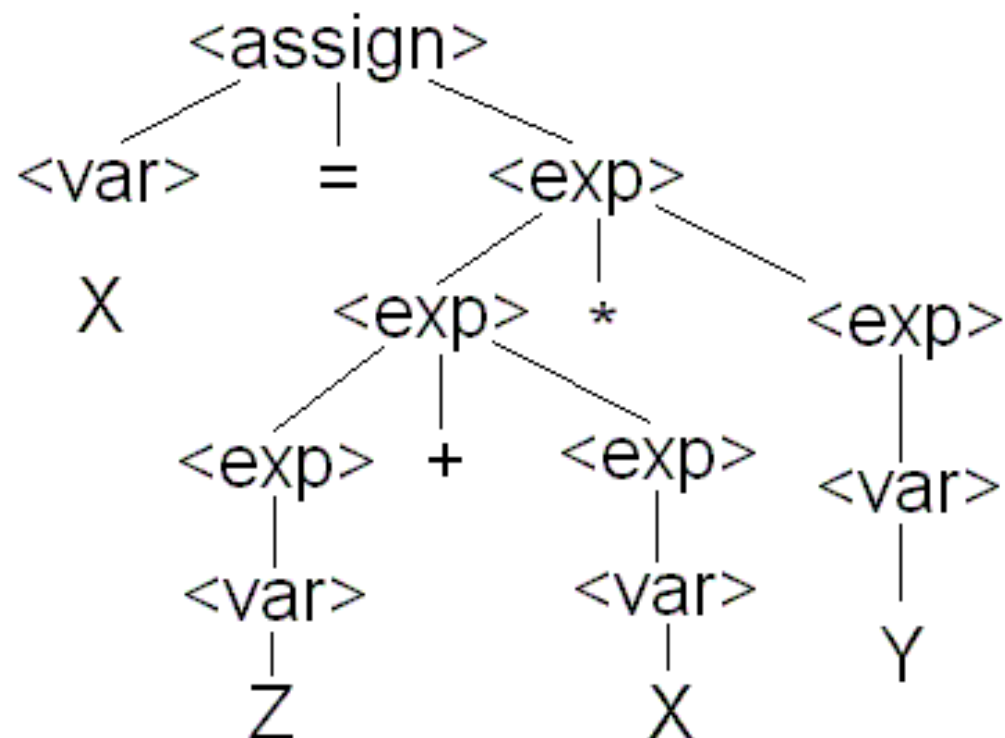
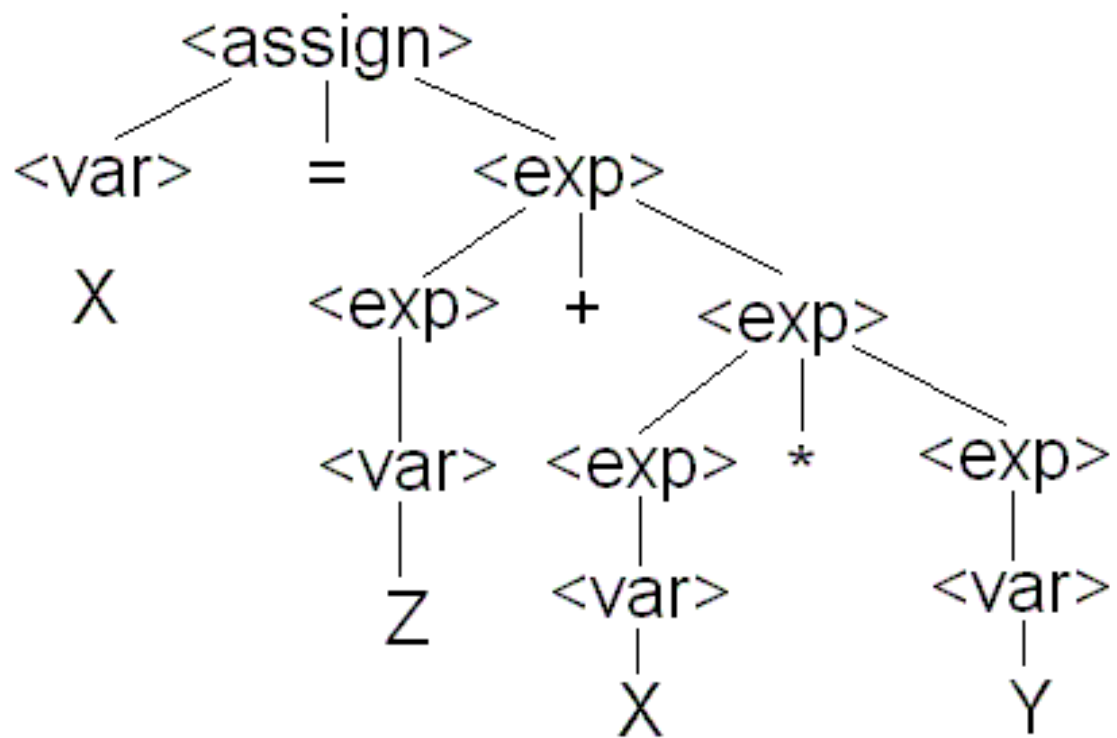
# Ambiguous Expression Grammar

1.  $\text{assign} \rightarrow \text{var} = \text{exp}$
2.  $\text{var} \rightarrow \mathbf{X} \mid \mathbf{Y} \mid \mathbf{Z}$
3.  $\text{exp} \rightarrow \text{exp} + \text{exp}$
4.  $\quad \quad \quad \mid \text{exp} * \text{exp}$
5.  $\quad \quad \quad \mid (\text{exp})$
6.  $\quad \quad \quad \mid \text{var}$



# Different Trees

- Two different derivations of  $X = Z + X * Y$



# Unambiguous Expression Grammar

1. assign  $\rightarrow$  var = exp
2. var  $\rightarrow$  X | Y | Z
3. exp  $\rightarrow$  exp + term
4.       | term
5. term  $\rightarrow$  term \* factor
6.       | factor
7. factor  $\rightarrow$  ( exp )
8.       | var

# Precedence

- Note that this unambiguous grammar also provides priority for multiplication over division
- An expression ( $exp$ ) will be recognized before a term
- The terms group multiplications which are then grouped for addition

# The Sad Truth

- Study of computer science theory has proven that certain problems cannot be solved
- It is not possible to write a program that will read in a grammar and determine that it is not ambiguous

# Grammar Problems

- Ambiguous Grammars
  - There exists more than one interpretation of what the input means
- Grammars that are not unique within a limited context
- Ambiguous Languages
  - There exist context-free languages that do not have an unambiguous grammar

# Unique Productions

- Parsers only look a few characters ahead, usually one character
- It is difficult to parse a language if different rules in the grammar have identical right hand sides

$$A \rightarrow B C D$$
$$X \rightarrow B C D$$

# Limited Context

- It is difficult to parse a language if the first several symbols of a right hand side are similar to another rule

$$\begin{array}{l} A \rightarrow B C D E X \\ \quad | B C D E Y \end{array}$$

The production  $A \rightarrow B C D E X \mid B C D E Y$   
can be simplified to

A

$$A \rightarrow Z X \mid Z Y$$
$$B C D E \rightarrow Z$$

B

$$A \rightarrow Z X \mid Z Y$$
$$Z \rightarrow B C D E$$



# Parsing Complexity

- Parsing algorithms that work for any unambiguous grammar are complex and slow
- General parsing algorithms are  $O(n^3)$
- Algorithms for grammars described in particular formats can be  $O(n)$

# Chomsky Normal Form

- All productions of a Chomsky Normal Form BNF are in the format:

$$A \rightarrow B C$$
$$A \rightarrow \mathbf{x}$$

- All rules have a right side with two non-terminals or one terminal symbol
- All non-ambiguous grammars can be converted to Chomsky Normal Form

# Greibach Normal Form

- All productions of a Greibach Normal Form BNF are in the format:

$$A \rightarrow \mathbf{x}$$

$$A \rightarrow \mathbf{x} B$$

$$A \rightarrow \mathbf{x} B \text{ more terminals and nonterminals}$$

- All rules have a right side starting with one terminal symbol then possibly a non-terminal and maybe more symbols
- All non-ambiguous grammars can be converted to Greibach Normal Form (GNF)

# Simple Grammar in GNF

1. assign  $\rightarrow$  **var** = exp
2. exp  $\rightarrow$  **var** + exp
3. | **var** \* exp
4. | ( exp )
5. | **var**

# Write a BNF

- A Snowflake assignment statement can be

`var = var`

`var pattern = var`

where pattern can be

`var`

`var | var | var | var ...`

# Possible Solution

asg      → **var = var**  
          | **var pattern = var**  
pattern → **var**  
          | **var more**

# Possible Solution in GNF

asg      → **var** rside  
          | **var** pattern rside  
pattern → **var**  
          | **var** more  
more     → | pattern  
rside    → = **var**

# General Parsing Rules

- The lexical scanner is simpler than the parser
- Avoid long productions
- Chomsky and Greibach Normal Forms are useful, although not necessary



# Homework

- Two assignments are posted on Blackboard
  1. Answer 9 questions that include FSA and Mealy machine parse tables
    - Due by noon on Monday, February 6, 2017
  2. Write a parser for the Snowflake language
    - Due by noon on Friday, February 10, 2017