CS 461

Programming Language Concepts

Gang Tan
Computer Science and Engineering
Penn State University

CH2 SYNTAX

BNF Example

- A simple arithmetic expression language
  - Non-terminals: e, n, d; identify grammatical categories
  - Terminals: +, -, 0, 1, 2, 3, 4, 5, …, 9; identify the basic alphabet
  - Production rules

```
<e> -> <n> | <e>+<e> | <e>–<e>
<n> -> <d> | <n><d>
<d> -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

  - "|" indicates a choice
  - the right hand side of a rule can be a sequence of terminal or non-terminal symbols
  - Start symbol: e
  - Note: the grammar has **recursion**
- Example numbers: 4, 27, 704
- Example expressions: 27 – 4 + 704

Derivations

- Derivation: a sequence of replacement steps
  - starting from the start symbol
  - Replace a nonterminal by the rhs (right hand side) of a rule
  - Keep doing it until resulting in a string of terminals

<e> → <e>+<e> → <e>–<e>+<e> → <n>–<n>+<n> → <n><d>–<d>+<d> → <d><d>–<d>+<d>
→ … → 27 – 4 + 3

Grammar defines a language: any terminal string that can be derived belongs to the language of the grammar

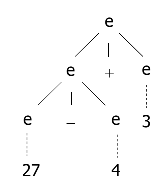
Left vs. right-most derivations

- Left-most derivation
  - At each step, always replace the leftmost nonterminal by one of its alternatives
- Right-most derivation
  - At each step, replace the rightmost nonterminal by one of its alternatives
- An example
  - 4 + 3
  - Both derivations correspond to the same parse tree

Parse tree

- Derivation represented by a tree

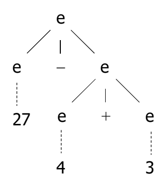
<e> → <e>+<e> → <e>–<e>+<e> → <n>–<n>+<n> → <n><d>–<d>+<d> → <d><d>–<d>+<d>
→ … → 27 – 4 + 3



Tree shows parenthesization of expressions;
Two derivations may correspond to the same parse tree

Parse Tree and Ambiguity

- The expression 27 – 4 + 3 has two parse trees



- Problem: 27 – (4 + 3) $\neq$ (27 – 4) + 3
- A grammar is ambiguous if some terminal string has more than one parse tree. Otherwise it is unambiguous
  - Note: not two derivations; to show a grammar is ambiguous, just need to find one terminal string with two parse trees

7

Ways to resolve operator ambiguity

- One way: add parentheses explicitly
  - <e> -> <n> | (<e> + <e>) | (<e> – <e>)
  - 27 – 4 + 3 is not an expression

8

Ways to resolve operator ambiguity

- Second way: design a new grammar that enforces associativity and precedence
  <e> -> <n> | <e>+<n> | <e>–<n>
  - 27 – 4 +3 has only one parse tree in this grammar; which one it is?
- Q: how many parse trees for 27 + 4 + 3
  - The grammar makes the +/- operator left associative (by convention)
    - By using left recursion
  - Q: what if we want to make + and – right associative?
- However, what if we really want 27 – (4 + 3)?
  <e> -> <t> | <e>+<t> | <e>–<t>
  <t> -> <n> | (<e>)
  - Ex: parse tree for 27 – (4+3)

9

Associativity & Precedence

- Associativity
  - Parenthesize operators of equal precedence to the left (or the right)
  - + is left associative, parse 3 + 4 + 5 as (3 + 4) + 5
  - Parse 3 – 4 + 5 as (3 – 4) + 5
  - Example : the exponentiation operator is right associative
- Precedence: an operator has a higher precedence than another operator if the first one binds tighter
  - Group * before +
  - Parse 2 + 3*4 as 2+ (3*4)

10

E-BNF (Extended BNF)

- Put alternative parts in parentheses and separate them with vertical bars
  - <exp> -> <exp> (+ | -) <exp>
  - Really just an abbreviation for two BNF rules
  - Note: "(", "|", and ")" are meta-symbols, while "+" and "-" are terminals
- Put repetition (0 or more) in curly braces { }
  - <num> -> <digit> {<digit>}
  - Example: 101
- Optional parts are placed in square brackets [ ]
  - <if-stmt> -> if <test> then <stmt> [else <stmt>]
- Examples:
  - <Expr> -> <Term> { (+ | -) <Term>}
  - <proc_call> -> <ident> ([<expr_list>])

11

E-BNF is no more powerful than BNF

- <exp> -> <exp> (+ | -) <exp>
  - Same as

  <exp> -> <exp> + <exp> |<exp> - <exp>
- <num> -> <digit> {<digit>}
  - Same as
    <num> -> <digit> | <digit> <num>
- <if-stmt> -> if <test> then <stmt> [else <stmt>]
  - Same as
    <if-stmt> -> if <test> then <stmt>
    | if <test> then <stmt> else <stmt>

12

Real Examples: Excerpt from Java Grammar

Variable declarations in Java

<var-dec>	→	<type-name> <declarator-list> **;**
<type-name>	→	boolean | byte | short | int | long | char | float | double
<declarator-list>	→	<declarator> | <declarator> **,** <declarator-list>
<declarator>	→	<variable-name> | <variable-name> **=** <expr>

Assume <variable-name> and <expr> are defined elsewhere