Compiler Construction (CS-636)

Sadaf Manzoor
UIIT, Rawalpindi

Outline

1. Parse Trees & Abstract Syntax Trees
2. Ambiguity & Ambiguous Grammars
3. Extended Notations: EBNF
4. Syntax Diagram
5. Summary

Parsing Preliminaries

Lecture: 9-10

Parse Trees & Abstract Syntax Trees

- Parse Trees
  - A parsed tree corresponding to a derivation is a labeled tree in which
    - the interior nodes are labeled by nonterminals
    - the leaf nodes are labeled by terminals
    - children of each internal node represents the replacement of the associated nonterminal in one step of the derivation
  - The internal nodes can be numbered to show the order of derivation
  - Parse trees can do **left-most derivation** as well as **right-most derivation**

Parse Trees & Abstract Syntax Trees
(Continue…)

exp → *exp op exp* | **(** *exp* **)** | ***number***
op → **+** | **–** | *****

- Exercise: Construct a parse tree of expression
  - 30 + 45
  - 2 - (4 + 5)
  - 2 + 4 + 6 * 20

Parse Trees & Abstract Syntax Trees
(Continue…)

- Abstract Syntax Trees
  - A parse tree contains much more information than is absolutely necessary for a compiler
  - The meanings or semantics of the expression should be directly related to its syntactic structure
  - To imply this (e.g. 3+4) the root represents the operation by adding the values of two child exp sub-trees

Parse Trees & Abstract Syntax Trees (Continue…)

$$exp \rightarrow exp\ op\ exp \mid \mathbf{(}\ exp\ \mathbf{)} \mid \textbf{\textit{number}}$$
$$op \rightarrow \mathbf{+} \mid \mathbf{-} \mid \mathbf{*}$$

- Exercise: Construct AST for the following expressions;
  - 3 + 4
  - (15 + 10) * 4
  - ((20*15) + 4)

Ambiguity & Ambiguous Grammar

- Parse trees and syntax trees uniquely express the structure of syntax, as do leftmost and rightmost derivations, but not derivations in general

- It is possible for a grammar to permit a string to have more than one parse tree

- A grammar that generates a string with two distinct parse trees is called **ambiguous grammar**

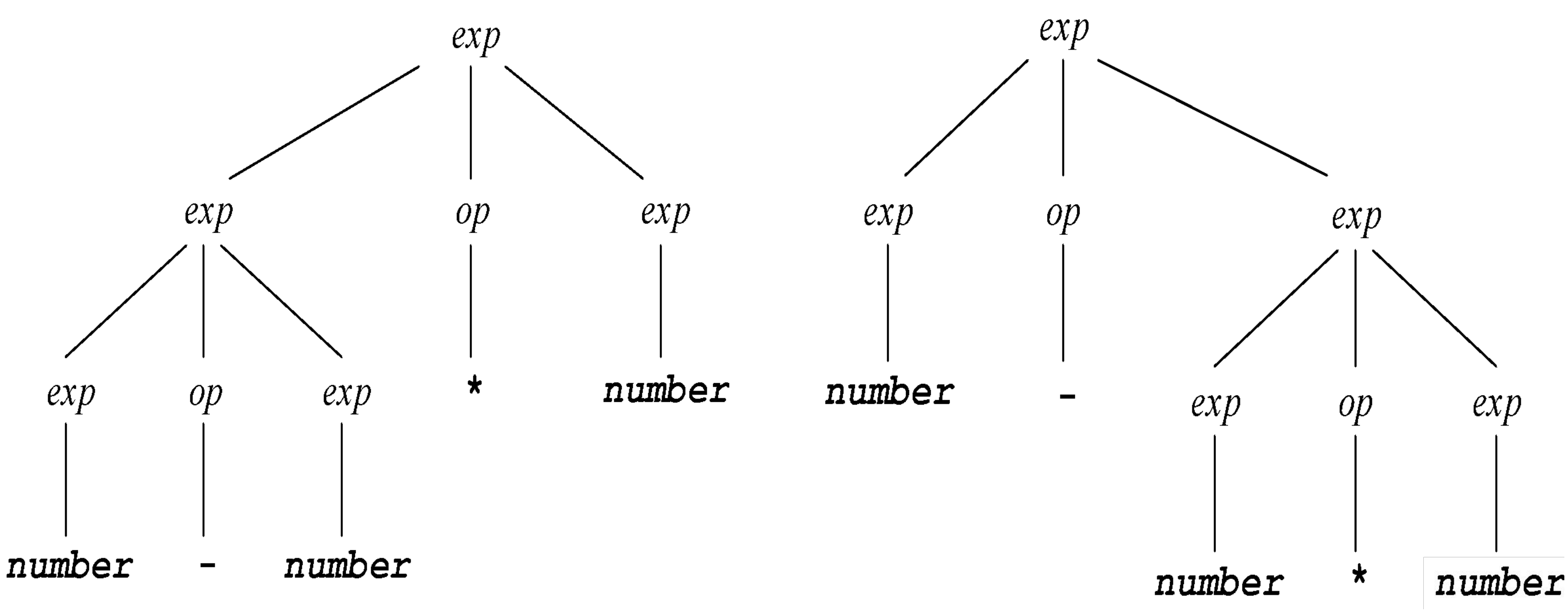
Ambiguity & Ambiguous Grammar (Continue…)

- One method to deal with ambiguities is to state a disambiguating rule that remove ambiguity without changing grammar but syntactic structure of language is no longer given by the grammar alone

- The alternative is to change the grammar into a form that forces the construction of the correct parse tree, thus removing the ambiguity

Parse Tree of 10-15*2

- Which Parse Tree is correct?



How to Remove Ambiguity?

- To remove ambiguity we could now state a disambiguating rule that establishes a relative precedence of three operations i.e. +, - and *
- The addition and subtraction operation will get same precedence but multiplication will get higher precedence
- The ambiguity introduced by same precedence can be removed by stating a disambiguating rule of associativity of each of the operations

Introducing Precedence

exp → *exp op exp* | **(** *exp* **)** | ***number***

op → **+** | **-** | *****

CHANGED TO

exp → *exp addop exp* | term

addpp → **+** | **-**

term → term mulop term | factor

mulop → *****

factor → **(** *exp* **)** | **number**

Note: We still have a problem in new CFG

Introducing Associativity

- Removed recursion and introduced left-association

exp → *exp addop term* | term

addpp → **+** | **-**

term → term mulop factor | factor

mulop → *****

factor → **(** *exp* **)** | **number**

Extended BNF (EBNF)

- So far: Backus-Naur Form (BNF)
  - Metasymbols are | → ε
- Extended BNF (EBNF):
  - New metasymbols [...] and {...}
  - ε largely eliminated by these
- Parenthesis: Grouping:
  - $exp \rightarrow exp\ (+ \mid -)\ term \mid term$
  - $exp \rightarrow exp + term \mid exp - term \mid term$
- The , operator: Concatenation

EBNF Metasymbols

- Brackets [...] mean "optional" (like ? in regular expressions):
  - $\textit{if-stmt} \rightarrow$ `if ( `*exp*` ) `*stmt*

    $|$ `if ( `*exp*` ) `*stmt* `else` *stmt*

    becomes:

    $\textit{if-stmt} \rightarrow$ `if ( `*exp*` ) `*stmt* [`else` *stmt*]

EBNF Metasymbols (Continue…)

- Braces {…} mean "repetition" (like * in regular expressions
- Replace *only* left-recursive repetition:
  - $exp \rightarrow exp + term \mid term$

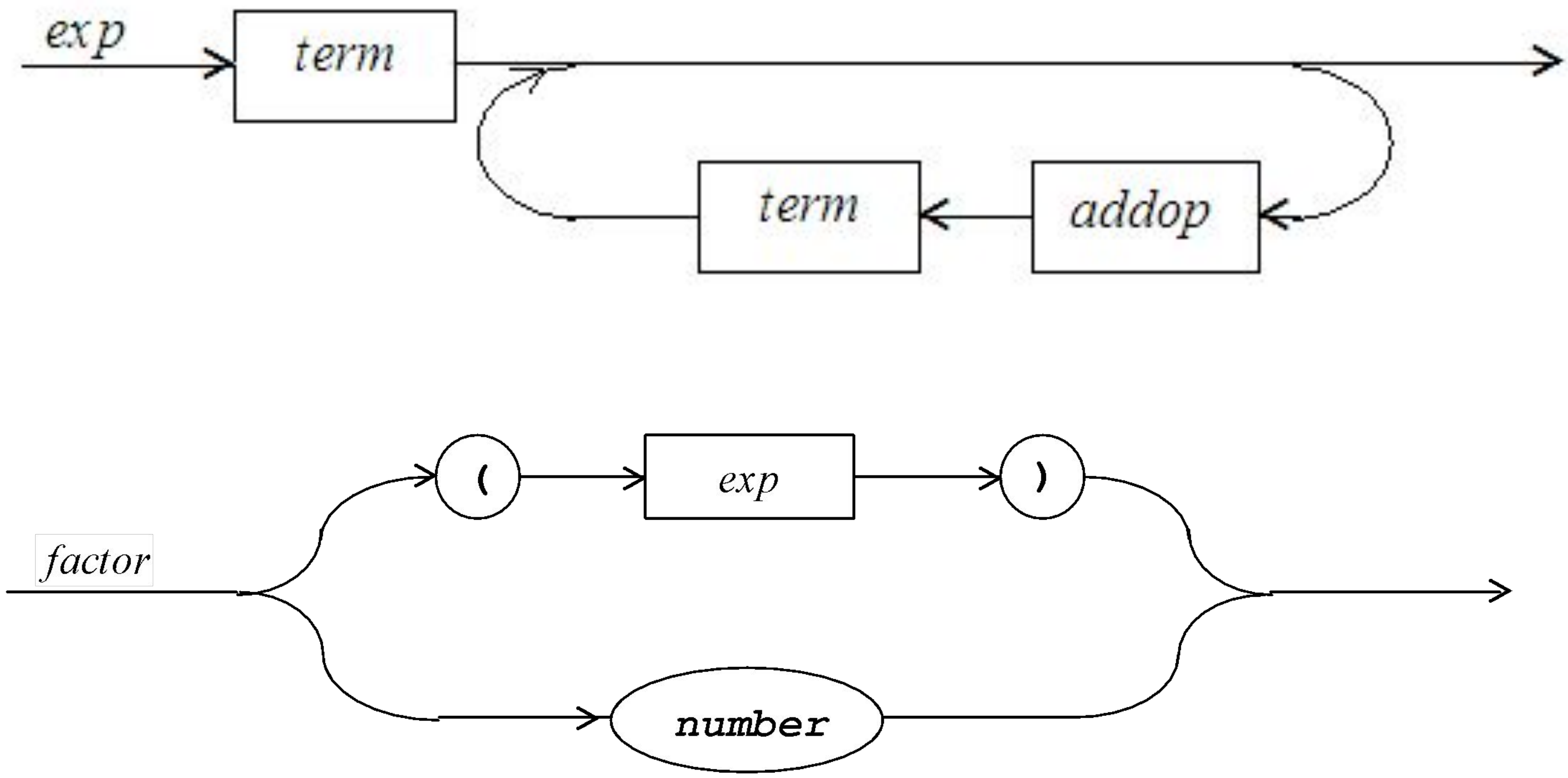
  becomes:

  $exp \rightarrow term \ \{ + term \}$

Syntax Diagrams

- Graphical representation for visually representing EBNF rules are called **syntax diagrams**
- In syntax diagrams
  - Nonterminal labels for each diagram represent the grammar rule defining that nonterminal
  - Arrow lines represent sequence and choices
  - A square or rectangle box is used to indicate non-terminals
  - A round or oval box is used to indicate a terminal

Syntax Diagrams (Continue…)



Syntax Diagrams (Continue…)

- Exercise
  - Draw a syntax diagram for variable declaration
    - e.g. int x;
    - e.g. int x,y;
    - e.g. int x=10,y=20,z;

Summary

Any Questions?