CS502: Compilers & Programming Systems

Context Free Grammars

Zhiyuan Li
Department of Computer Science
Purdue University, USA

PURDUE
UNIVERSITY

Course Outline

- Languages which can be represented by regular expressions are called *regular languages*.
- Most language constructs are more complex than regular languages.
- Example: It is impossible to use a DFA to recognize all sequences of balanced (possibly nested) parentheses.
  - The "pumping lemma" is often used to prove that a certain language is too complex to be regular.
- Context-free grammars (CFGs) are commonly used to define a wider class of languages because they are powerful enough to specify common syntax rules.

PURDUE UNIVERSITY

What is the grammar used for?

- It defines the correct forms of program constructs.
- Program semantics will be defined in terms of program constructs.
- Given a context-free grammar, the compiler writer tries to construct a parser to recognize syntax constructs.
  - The parser checks to see whether the program conforms to the grammar, i.e. whether it has the correct syntax structure.
- For an arbitrary context free grammar, we may or may not be able to build a parser automatically that recognizes all programs that conform to the grammar.
  - Recall that, given an arbitrary regular expression, R, any of the three methods we studied (*NFA*, *NFA*→*DFA*, *DFA*) can be used to build a lexical analyzer automatically that recognizes all strings defined by R, without backtracking.
- We may need to rewrite the grammar (manually) in a form for which we know how to build a parser.

Impact of the parser on semantics processing

- Not only must the parser recognize the correct syntactic forms, it must also be suitable for triggering correct *semantic actions that*
  - Build the correct abstract syntax tree
    - This is vital to the generation of the correct final code
- Hence, we must study how to properly design the grammar for a programming language we want to implement.

Basic Concepts

- A language L is a set of strings formed by symbols from an *alphabet*. In the parsing phase, such symbols are tokens.
- A program is viewed as a sequence of tokens.
- L is also often said to be a set of *sentences*. For programming languages, each sentence is a program(!)
- We use an example to explain the following terminology:
  - Production rules and grammar symbols
  - The *start* symbol
  - A derivation step and a derivation sequence
  - A *terminal* is a grammar symbol which derives nothing but itself. (The set of terminals form the vocabulary of L.)

- Beginning with the start symbol, every time we replace a nonterminal by the right hand phrase of one of its production rules, we have performed a derivation step and derived a new *sentential form.*
- A sentence is a special case of sentential forms
- Left-most (lm) vs. right-most (rm) derivations.
- Given a program, the parser in a modern compiler essentially performs lm (or rm) derivations.
  - In each derivation step, a new node (and some new edges) may get inserted in the AST, or some new type information may get extracted and placed in the symbol table.
- If a sequence of tokens can be derived from the start symbol, then it is accepted by the CFG.

A Parse Tree

- A parse tree corresponds to a set of derivation sequences for a given input
  - Given a parse tree, there exist a unique left-most derivation sequence and a unique right-most derivation sequence
- The parser can be viewed as incrementally (and implicitly) constructing a parse tree.
- A CFG is called *ambiguous* if and only if there exist a sentence for which there exist more than one parse tree.
  - A CFG which contains a cycle is definitely ambiguous. Why?
  - Why ambiguous grammars are bad?
    - Program semantics is defined in terms of program constructs
    - the ambiguity in the CFG often causes ambiguity in the definition of program constructs and operation orders.

- A key issue is whether the correct AST can be built by following that set of preference rules.
- Sometimes, additional rules are introduced (in English descriptions, e.g.) in order to define such constructs or orders unambiguously
  - E.g. how to handle the “dangling else” case

PURDUE
UNIVERSITY

Some common forms of production rules

- Use left recursion or right recursion to define a list of constructs.
  - Example: List of statements.
- Use a “mirrored“ recursion to define nested pairs.
  - Example: balanced and nested pairs of parentheses.
- Binary expressions

Parsers

- There are two fundamental approaches to parsing: top-down vs. bottom-up.
  - With the top-down approach, the parser performs left-most derivations, beginning with the start nonterminal.
  - With the bottom-up approach, the parser traces rightmost derivations backward, beginning with the given sentence (i.e. the sequence of tokens).

PURDUE UNIVERSITY