CS412/CS413
Introduction to Compilers
Radu Rugina

Lecture 5: Grammars
1 Feb 06

Outline

• Context-Free Grammars (CFGs)
• Derivations
• Parse trees and abstract syntax
• Ambiguous grammars

Where we Are

Source code (character stream) if (b == 0) a = b;

Token stream

if (b == 0) a = b;

Abstract syntax tree (AST)

Lexical Analysis

Syntax Analysis (Parsing)

Semantic Analysis

Syntax Analysis Example

Source code (token stream)

if (b == 0) a = b;
while (a != -1) {
    stdio.print(a);
    a = a - 1;
}

Abstract Syntax Tree

Parsing Analogy

• Natural languages: recognize whether a sentence is grammatically well-formed and identify the function of each component.

"I gave him the book"

sentence

subject: I

verb: gave

indirect object: him

object

noun phrase

article: the

noun: book

Syntax Analysis Overview

• Goal: check that the input token stream satisfies the syntactic structure of the language

• What we need:
  – An expressive way to describe the syntax
  – An mechanism that:
    • Checks if the input token stream has correct syntax
    • And determines what the syntactic structure is
Why Not Regular Expressions?

- Regular expressions can expressively describe tokens
  - easy to implement, efficient (using DFAs)
- Why not use regular expressions (on tokens) to specify programming language syntax?
- Reason: they don't have enough power to express the syntax in programming languages
- Typical constructs: nested expressions, nested statements
  - Similar to the language of balanced parentheses
  - \( ()^+ \cup ((()^+))^+ \) ...
  - needs unbounded counting

Context-Free Grammars

- A Context-Free Grammar is a tuple \((V, \Sigma, S, \rightarrow)\)
  - \( V \) is a finite set of nonterminal symbols
  - \( \Sigma \) is a finite set of terminal symbols
  - \( S \in V \) is a distinguished nonterminal, the start symbol
  - \( \rightarrow \subseteq V \times (V \cup \Sigma)^* \) is a finite relation, the productions
- Context Free Grammar is abbreviated CFG
  - Note: CFG also stands for "control flow graph"

Typographical Conventions

- \( A, B, C, \ldots \) are nonterminals
- \( a, b, c, \ldots \) are terminals
- \( \ldots, x, y, z \) are strings of terminals
- \( \alpha, \beta, \gamma, \delta, \ldots \) are strings of terminals or nonterminals
- \( A \rightarrow \alpha \) denotes production \((A, \alpha)\)
- In production \( A \rightarrow \alpha \)
  - \( A \) is the left-hand side (LHS)
  - \( \alpha \) is the right-hand side (RHS)
- \( A \rightarrow \alpha_1 \ldots \alpha_n \) denotes \( n \) productions \( A \rightarrow \alpha_1, \ldots, A \rightarrow \alpha_n \)

Sample Grammar

- \((V, \Sigma, S, \rightarrow)\), where
  - \( V \) is \( \{ S \} \), i.e., there is one nonterminal \( S \)
  - \( \Sigma \) is \( \{ a, b \} \), i.e., there are two terminals "a" and "b"
  - \( \rightarrow \) is defined by two productions \( S \rightarrow aSbS \) and \( S \rightarrow \varepsilon \)
- What language does this grammar describe?

Direct Derivations

- Let \( G = (V, \Sigma, S, \rightarrow) \) be a CFG.
  The "directly derives" relation is defined by:
  \( \alpha \gamma \Rightarrow \alpha \beta \gamma \) if \( A \rightarrow \beta \)
- Examples
  - Let \( G \) be the grammar with productions \( S \rightarrow aSbS \mid \varepsilon \)
    - Then
      - \( aSbS \Rightarrow a \varepsilon bS \)
      - \( aSbS \Rightarrow abS \)

Context Free Languages

- The language generated by grammar \( G \) is:
  \( L(G) = \{ x \mid S \Rightarrow^* x \} \)
- \( L(G) \) is the set of strings of terminals derived from \( S \) by repeatedly applying the productions as rewrite rules
  - Context Free Languages (CFLs) are the languages generated by context-free grammars
- If \( x \in L(G) \), then a derivation of \( x \) is a sequence of strings \( \alpha_0, \alpha_1, \ldots, \alpha_i \) such that \( \alpha_0 = S, \alpha_i = x, \alpha_i \Rightarrow \alpha_{i+1} \) for \( i=0..n-1 \). We write \( S \Rightarrow \alpha_1 \ldots \Rightarrow \alpha_n \Rightarrow x \)
Every Regular Language is a CFL

- Inductively build a CFG for each RE
  - $\varepsilon \rightarrow \varepsilon$
  - $a \rightarrow a$
  - $R_1, R_2 \rightarrow S_1, S_2$
  - $R_1 \mid R_2 \rightarrow S_1 \mid S_2$
  - $R_1^* \rightarrow S \rightarrow S_1 \mid S_2 \mid \epsilon$

where:
- $G_1$ = grammar for $R_1$, with start symbol $S_1$
- $G_2$ = grammar for $R_2$, with start symbol $S_2$

Grammars and Acceptors

- Acceptors for context-free grammars
  - Context-Free
  - Grammar $G \rightarrow$ Acceptor $\rightarrow$
    - Yes, if $x \in L(G)$
    - No, if $x \notin L(G)$

- Syntax analyzers (parsers) = CFG acceptors. They also output the corresponding derivation when the token stream is accepted
  - Various kinds: LL(k), LR(k), SLR, LALR

Another Example: Sum Grammar

- Grammar:
  - $S \rightarrow E + S \mid E$
  - $E \rightarrow \text{num} \mid (S)$

- Expanded:
  - $S \rightarrow E + S$
  - $S \rightarrow E$
  - $E \rightarrow \text{num}$
  - $E \rightarrow (S)$
  - 4 productions
  - $V = \{ S, E \}$
  - $\Sigma = \{ +, \cdot, \text{num} \}$
  - start symbol $S$

Example accepted input:
- $(1+2+(3+4))+5$

Derivation Example

Derive $(1+2+(3+4))+5$

- $S \rightarrow E + S \mid E$
- $E \rightarrow \text{num} \mid (S)$

Derivation:
- $S \Rightarrow E + S \Rightarrow E + S + S \Rightarrow E + (S + S) \Rightarrow E + ((3 + 4) + 5) \Rightarrow \boxed{E + (3 + 4) + 5}$

Derivations and Parse Trees

- The Parse Tree is a tree representation of the derivation
- Leaves = terminals
- Internal nodes = nonterminals
- No information about order of derivation steps

Parse Tree vs. AST

- Parse tree also called "concrete syntax"

Parse Tree (Concrete Syntax)

Discards (abstracts) unneeded information
### Derivation Order

- Can choose to apply productions in any order; select any nonterminal \( A \) such that \( \alpha \gamma \Rightarrow \alpha \beta \gamma \)
- Two standard orders: leftmost and rightmost -- useful for different kinds of automatic parsing
- **Leftmost derivation**: Always replace leftmost nonterminal \( E + S \Rightarrow 1 + S \)
- **Rightmost derivation**: Always replace rightmost nonterminal \( E + S \Rightarrow E + E + S \)

### Example

- \( S \Rightarrow E + S | E \)
  
- **Leftmost derivation**
    
    \[
    S \Rightarrow E + S \Rightarrow (E + S) \Rightarrow (1 + S) \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow \]
    
    \[
    (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow (1 + S) + S \Rightarrow \]
    
- **Right-most derivation**
    
    \[
    S \Rightarrow E + S \Rightarrow E + S \Rightarrow \]
    
- **Same parse tree**: same productions chosen, different order

### Parse Trees

- In example grammar, leftmost and rightmost derivations produced identical parse trees
- \( + \) operator associates to right in parse tree regardless of derivation order

\[
\begin{align*}
(1 + 2 + (3 + 4)) + 5 & \quad \Rightarrow \\
1 + 2 + (3 + 4) + 5 & \quad \Rightarrow
\end{align*}
\]

### An Ambiguous Grammar

- \( + \) associates to right because of right-recursive production \( S \Rightarrow E + S \)
- Consider another grammar:
  
  \[
  S \Rightarrow S + S \mid S * S \mid \text{num}
  \]
- **Ambiguous grammar**: a string in the language has multiple parse trees

### Different Parse Trees

\[
S \Rightarrow S + S \mid S * S \mid \text{num}
\]

- Consider expression \( 1 + 2 * 3 \)
- Derivation 1: \( S \Rightarrow S + S \Rightarrow 1 + S \Rightarrow 1 + S + S \Rightarrow 1 + 2 * S \Rightarrow 1 + 2 * S \Rightarrow 1 + 2 * S \)
- Derivation 2: \( S \Rightarrow S * S \Rightarrow S * S \Rightarrow 1 + S * S \Rightarrow 1 + 2 * S \Rightarrow 1 + 2 * S \)

These derivations correspond to different parse trees!
- Hence, the grammar is ambiguous