CS412/CS413

Introduction to Compilers

Radu Rugina

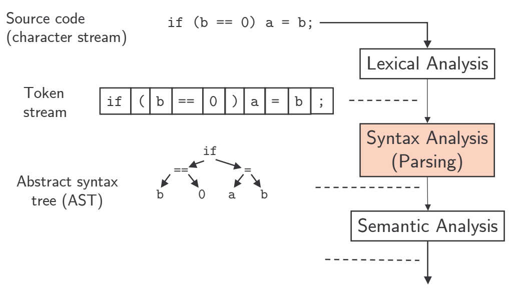
Lecture 5: Grammars

1 Feb 06

Outline

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

Where we Are

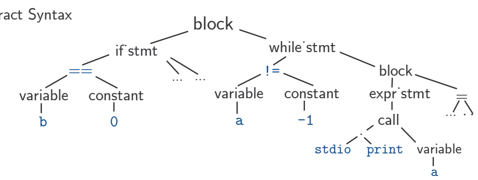


Syntax Analysis Example

Source code
(token stream)

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

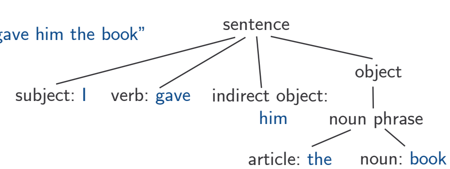
Abstract Syntax
Tree



Parsing Analogy

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

"I gave him the book"



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

    (), ()(), (()), ()(()), (())(()), ((())(())) …

    needs unbounded counting

Context-Free Grammars

- A Context-Free Grammar is a tuple $\langle V,\Sigma,S,\rightarrow\rangle$
  - V is a finite set of nonterminal symbols
  - Σ 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, … are nonterminals
- a, b, c, … are terminals
- …, x, y, z are strings of terminals
- $\alpha, \beta, \gamma, \delta$, … are strings of terminals or nonterminals
- $A \rightarrow \alpha$ denotes production $\langle A,\alpha\rangle$
- In production $A \rightarrow \alpha$
  - A is the left-hand side (LHS)
  - α is the right-hand side (RHS)
- $A \rightarrow \alpha_1 \mid \ldots \mid \alpha_n$ denotes n productions $A \rightarrow \alpha_1, \ldots, A \rightarrow \alpha_n$

Sample Grammar

- $\langle V,\Sigma,S,\rightarrow\rangle$, where
  - V is { S }, i.e., there is one nonterminal S
  - Σ 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 $G = \langle V,\Sigma,S,\rightarrow\rangle$ be a CFG.

  The "directly derives" relation is defined by:

  $$\alpha A \gamma \Rightarrow \alpha\beta\gamma \quad \text{if } A \rightarrow \beta$$

- Examples
  - Let G be the grammar with productions $S \rightarrow aSbS \mid \varepsilon$
  - Then
    - $aSbS \Rightarrow aaSbSbS$
    - $aSbS \Rightarrow abS$

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 $\alpha_0, \alpha_1, \ldots, \alpha_n$ such that $\alpha_0 = S$, $\alpha_0 = x$, $\alpha_i \Rightarrow \alpha_{i+1}$ 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

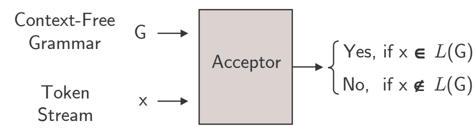
ε	S → ε
a	S → **a**
$R_1 R_2$	$S \to S_1 S_2$
$R_1 \mid R_2$	$S \to S_1 \mid S_2$
$R_1 *$	$S \to S_1 S \mid \varepsilon$

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



- 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 → E + S | E
  E → num | (S)
- Expanded:
  S → E + S
  S → E
  E → num
  E → (S)

  4 productions
  V = { S, E }
  Σ = { (,), +, num }
  start symbol S
- Example accepted input:
  (1+2+(3+4))+5

Derivation Example

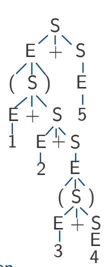
Derive (1+2+(3+4))+5

S ⇒ E+S
⇒ (S)+S
⇒ (E+S)+S
⇒ (1+S)+S
⇒ (1+ E+S)+S
⇒ (1+2+S)+S
⇒ (1+2+E)+S
⇒ (1+2+(S))+S
⇒ (1+2+(E+S) +S
⇒ (1+2+(3+S))+S
⇒ (1+2+(3+E))+S
⇒ (1+2+(3+4))+S
⇒ (1+2+(3+4))+E
⇒ (1+2+(3+4))+5

S → E + S | E
E → number | (S)

Derivations and Parse Trees

Parse Tree



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

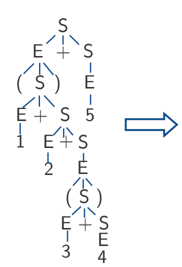
Derivation

S ⇒ E + S ⇒ (S) + S ⇒ (E + S) + S ⇒ (1 + S)+S ⇒ (1 + E + S) + S ⇒ …
⇒(1 + 2 + (S)) + S ⇒ (1 + 2 + (E + S))+S ⇒ … ⇒ (1 + 2 + (3 + E))+S
⇒ … ⇒(1 + 2+ (3+4))+5

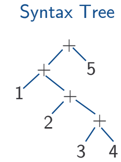
Parse Tree vs. AST

- Parse tree also called "concrete syntax"

Parse Tree (Concrete Syntax)



Abstract Syntax Tree



Discards (abstracts) unneeded information

Derivation Order

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

  E + S ⇒ 1 + S
- Rightmost derivation: Always replace rightmost nonterminal

  E + S ⇒ E + E + S

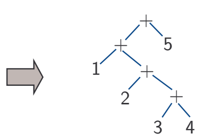
Example

- S → E + S | E
  E → num | (S)
- Left-most derivation
  S ⇒ E+S ⇒ (S) + S ⇒ (E + S)+ S ⇒ (1 + S)+S ⇒ (1+E+S)+S ⇒ (1+2+S)+S ⇒ (1+2+E)+S ⇒ (1+2+(S))+S ⇒ (1+2+(E+S))+S ⇒ (1+2+(3+S))+S ⇒ (1+2+(3+E))+S ⇒ (1+2+(3+4))+S ⇒ (1+2+(3+4))+E ⇒ (1+2+(3+4))+5
- Right-most derivation
  S ⇒ E+S ⇒ E+E ⇒ E+5 ⇒ (S)+5 ⇒ (E+S)+5 ⇒ (E+E+S)+5 ⇒ (E+E+E)+5 ⇒ (E+E+(S))+5 ⇒ (E+E+(E+S))+5 ⇒ (E+E+(E+E))+5 ⇒ (E+E+(E+4))+5 ⇒ (E+E+(3+4))+5 ⇒ (E+2+(3+4))+5 ⇒ (1+2+(3+4))+5
- 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

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



An Ambiguous Grammar

- + associates to right because of right-recursive production S → E + S
- Consider another grammar:

  S → S + S | S * S | num
- Ambiguous grammar : a string in the language has multiple parse trees

Different Parse Trees

S → S + S | S * S | num

- Consider expression 1 + 2 * 3
- Derivation 1: **S** ⇒ **S** + S ⇒ 1 + **S** ⇒ 1 + **S** * S ⇒ ⇒ 1 + 2 * **S** ⇒ 1 + 2 * 3
- Derivation 2: **S** ⇒ **S** * S ⇒ **S** + S * S ⇒ 1 + **S** * S ⇒ 1 + 2 * **S** ⇒ 1 + 2 * 3
- These derivations correspond to different parse trees!
- Hence, the grammar is ambiguous