CS412/CS413

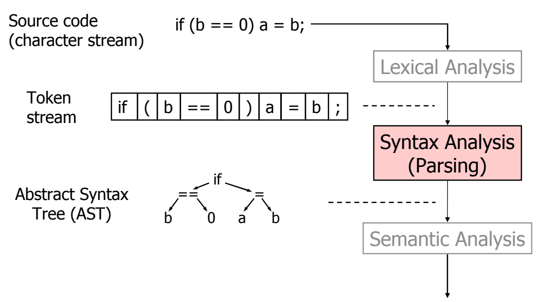
Introduction to Compilers
Tim Teitelbaum

Lecture 5: Context-Free Grammars
31 Jan 07

Outline

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

Where We Are

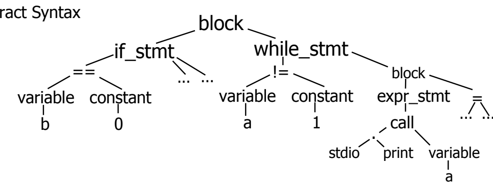


Syntax Analysis Example

Source code
(token stream)

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

Abstract Syntax
Tree



Syntax Analysis Overview

- Goal: determine if the input token stream satisfies the syntax of a legal program, and if so, identify its structure
- We need:
  - An expressive way to describe the syntax
  - An acceptor mechanism that determines if the input token stream satisfies that syntax description
  - A way to recover the syntactic structure

Why Not Regular Expressions?

- Reason: they don't have enough power to express the syntax of programming languages
- Example: nested bracketed constructs (e.g., blocks and expressions)
  - Language of balanced parentheses

    { (), ()(), (()), (())(), ()(()), (())(()), ((())(())), etc. }

    needs unbounded counting to be recognized.

Prerequisites: Language Theory (review)

- Let Σ be finite set of symbols, an alphabet
- Σ^* denotes the set of all finite strings of symbols in Σ
- ε denotes the empty string
- Any subset $L \subseteq \Sigma^*$ is called a language
- If L_1 and L_2 are languages, then $L_1 L_2$ is the concatenation of L_1 and L_2, i.e., the set of all pair-wise concatenations of strings from L_1 and L_2, respectively
- Let $L \subseteq \Sigma^*$ be a language. Then
  - $L^0 = \{\}$
  - $L^{n+1} = L\,L^n$ for all $n \geq 0$

Prerequisites: Binary Relations

- If S_1 and S_2 are sets, $S_1 \times S_2$ denotes the Cartesian product, the set $\{ \langle s_1,s_2 \rangle \mid s_1 \in S_1$ and $s_2 \in S_2\}$
- $S \times S$ is written S^2
- If S_1 and S_2 are sets, a set $R \subseteq S_1 \times S_2$ is called a binary relation between S_1 and S_2
- If $\langle s_1,s_2 \rangle \in R$, we write $s_1\ R\ s_2$

Prerequisites: Composition, Powers and Closures

- If $R_1 \subseteq S_1 \times S_2$ and $R_2 \subseteq S_2 \times S_3$ are relations, $R_1 \bullet R_2$, the composition of R_1 and R_2, is $\{ \langle x,z \rangle \mid x\,R_1\,y$ and $y\,R_2\,z\}$
- If R is a relation in $S \times S$, then
  - $R^0 = \{ <x,x> \mid x \in S \}$, the identity relation over S
  - for $i \geq 0$, $R^{i+1} = R \bullet R^i$
  - in particular
    - $R^1 = R$;
    - R^2 is $R \bullet R$
  - $R^+ = R^1 \cup R^2 \cup R^3 \cup \ldots$, the transitive closure of R
  - $R^* = R^0 \cup R^+$, the transitive reflexive closure of R

Context-Free Grammars

- A Context-Free Grammar (CFG) is a 4-tuple $\langle V,\Sigma,S,\rightarrow \rangle$, where
  - 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
- Sample CFG $\langle V,\Sigma,S,\rightarrow \rangle$, where
  - V is { S }, i.e., there is one nonterminal S
  - Σ is { a, b }, i.e., there are two terminal symbols "a" and "b"
  - S is start symbol
  - $\rightarrow$ is $\{ \langle S,aSbS \rangle, \langle S,\varepsilon \rangle \}$
    i.e., there are two productions $S \rightarrow aSbS$ and $S \rightarrow \varepsilon$

More notation and typographical conventions

- A, B, C, … are nonterminals
- a, b, c, … are terminals
- …, X, Y, Z are either terminals or nonterminals
- …, w, 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 lefthand side (LHS)
  - α is the righthand side (RHS)
- $A \rightarrow \alpha_1 | \ldots | \alpha_n$ denotes the n productions $A \rightarrow \alpha_1, \ldots, A \rightarrow \alpha_n$

Sample Grammar (ctd)

- It is not uncommon to just say:

  Let G be the grammar with productions

  $S \rightarrow aSbS \mid \varepsilon$

  and infer the nonterminals, terminals, and start symbol from the productions by invoking the conventions

Direct Derivations

- Let G = ⟨V,Σ,S,→⟩ be a CFG. The "directly derives" relation (⇒) is defined as { ⟨αAγ, αβγ⟩ | A→β }.

- Example
  - Let G be the grammar with productions S → aSbS | ε
  - Then
    - S ⇒ aSbS
    - aSbS ⇒ aaSbSbS
    - aaSbSbS ⇒ aabSbS
    - aabSbS ⇒ aabbS
    - aabbS ⇒ aabbaSbS
    - aabbaSbS ⇒ aabbabS
    - aabbabS ⇒ aabbab

nonterminal is LHS of production
string is RHS of production

Context Free Languages

- Let G = ⟨V,Σ,S,→⟩ be a CFG. The language generated by G, denoted L(G) = { x | S ⇒* x }

- Let $\alpha_0 \Rightarrow^* \alpha_n$. A derivation of α_n from α_0 is a sequence of strings $\alpha_0, \alpha_1, \ldots, \alpha_n$ such that $\alpha_i \Rightarrow \alpha_{i+1}$ for $0 \le i < n$. We write $\alpha_0 \Rightarrow \alpha_1 \ldots \Rightarrow \alpha_n$.

- Context Free Languages (CFLs) are the languages generated by context-free grammars

Example

- Let G be the grammar with productions S → aSbS | ε

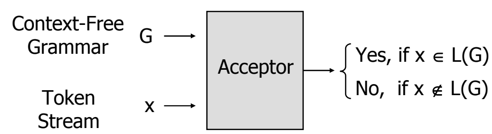
- Then

  S ⇒ aSbS ⇒ aaSbSbS ⇒ aabSbS ⇒ aabbS ⇒ aabbaSbS ⇒ aabbabS ⇒ aabbab

- I.e., aabbab is in L(G)

Grammars and Acceptors

- Acceptors for context-free grammars



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

Every Regular Language is a Context Free Language

- Inductively build a CFG for each RE

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

where:

G_1 = grammar for R_1, with start symbol S_1
G_2 = grammar for R_2, with start symbol S_2

Sum Grammar

- Grammar:

  S → E + S | E
  E → number | (S)

- Expanded:

  S → E + S
  S → E
  E → number
  E → (S)

  4 productions
  2 nonterminals: S E
  4 terminals: ( ) + number
  start symbol S

- Example accepted input:

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

Derivation Example

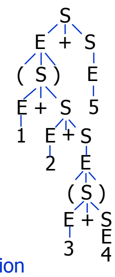
$S \rightarrow E + S \mid E$
$E \rightarrow \text{number} \mid (S)$

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

S $\Rightarrow$ E+S
$\Rightarrow$ (S)+S
$\Rightarrow$ (E+S)+S
$\Rightarrow$ (1+S)+S
$\Rightarrow$ (1+ E+S)+S
$\Rightarrow$ (1+2+S)+S
$\Rightarrow$ (1+2+E)+S
$\Rightarrow$ (1+2+(S))+S
$\Rightarrow$ (1+2+(E+S) +S
$\Rightarrow$ (1+2+(3+S))+S
$\Rightarrow$ (1+2+(3+E))+S
$\Rightarrow$ (1+2+(3+4))+S
$\Rightarrow$ (1+2+(3+4))+E
$\Rightarrow$ (1+2+(3+4))+5

Derivations and Parse Trees

Parse Tree



- Parse Tree = tree representation of the derivation
- Leaves of tree are terminals
- Internal nodes: nonterminals
- No information about order of derivation steps

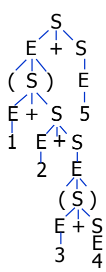
Derivation

S $\Rightarrow$ E + S $\Rightarrow$ (S) + S $\Rightarrow$ (E + S) + S $\Rightarrow$ (1 + S)+S $\Rightarrow$ (1 + E + S) + S $\Rightarrow$ …
$\Rightarrow$(1 + 2 + (S)) + S $\Rightarrow$ (1 + 2 + (E + S))+S $\Rightarrow$ … $\Rightarrow$ (1 + 2 + (3 + E))+S
$\Rightarrow$ … $\Rightarrow$(1 + 2+ (3+4))+5

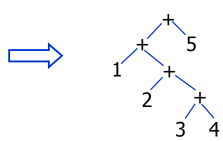
Parse Tree vs. AST

- 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 $\Rightarrow$ 1 + S
- Rightmost derivation: Always replace rightmost nonterminal
  E + S $\Rightarrow$ E + E + S

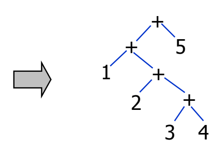
Example

- $S \rightarrow E + S \mid E$
  $E \rightarrow \text{number} \mid (S)$
- Left-most derivation
  S $\Rightarrow$ E+S $\Rightarrow$ (S) + S $\Rightarrow$ (E + S)+ S $\Rightarrow$ (1 + S)+S $\Rightarrow$ (1+E+S)+S $\Rightarrow$ (1+2+S)+S $\Rightarrow$ (1+2+E)+S $\Rightarrow$ (1+2+(S))+S $\Rightarrow$ (1+2+(E+S))+S $\Rightarrow$ (1+2+(3+S))+S $\Rightarrow$ (1+2+(3+E))+S $\Rightarrow$ (1+2+(3+4))+S $\Rightarrow$ (1+2+(3+4))+E $\Rightarrow$ (1+2+(3+4))+5
- Right-most derivation
  S $\Rightarrow$ E+S $\Rightarrow$ E+E $\Rightarrow$ E+5 $\Rightarrow$ (S)+5 $\Rightarrow$ (E+S)+5 $\Rightarrow$ (E+E+S)+5 $\Rightarrow$ (E+E+E)+5 $\Rightarrow$ (E+E+(S))+5 $\Rightarrow$ (E+E+(E+S))+5 $\Rightarrow$ (E+E+(E+E))+5 $\Rightarrow$ (E+E+(E+4))+5 $\Rightarrow$ (E+E+(3+4))+5 $\Rightarrow$ (E+2+(3+4))+5 $\Rightarrow$ (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:

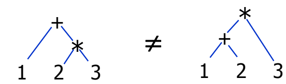
  S → S + S | S * S | number

- Ambiguous grammar = different derivations of the same string (may) produce different parse trees

Differing Parse Trees

S → S + S | S * S | number

- 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** * 3 ⇒ S + **S** * 3 ⇒ **S** + 2 * 3 ⇒ 1 + 2 * 3



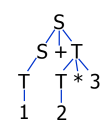
Impact of Ambiguity

- Different parse trees correspond to different evaluations!
- Meaning of program not defined

Eliminating Ambiguity

- Often can eliminate ambiguity by adding nonterminals & allowing recursion only on right or left

S → S + T | T
T → T * **num** | **num**



- T nonterminal enforces precedence
- Left-recursion : left-associativity

Context Free Grammars

- Context-free grammars allow concise syntax specification of programming languages
- A CFG specifies how to convert token stream to parse tree (if unambiguous!)