Context Free Grammars

CS154
Chris Pollett
Mar 1, 2006.

Outline

- Formal Definition
- Ambiguity
- Chomsky Normal Form

Formal Definitions

- A context free grammar is a 4-tuple (V, Σ, R, S) where
  1. V is a finite set called the **variables**
  2. Σ is a finite set, disjoint from V called the **terminals**.
  3. R is a finite set of **rules**, with each rule being a pair consisting of a variable and a string of variables and terminals, and
  4. $S \in V$ is a start variable.
- For a rule A--> w where w is a string over ($V \cup \Sigma$), and for other strings u and v, we say uAv **yields** uwv, written uAv => uwv. We say u derives v, written $u \Rightarrow^{*} v$, if there is a finite sequence:

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v.$$

Example

- Consider the grammar G= (V, Σ, R, <EXPR>) where V is

  {<EXPR>, <TERM>, <FACTOR>}

  and Σ is (a, +, x, (,))

  and the rules are:

  <EXPR> --> <EXPR> + <TERM> | <TERM>

  <TERM> --> <TERM> x <FACTOR> | <FACTOR>

  <FACTOR> --> (<EXPR>) | a

- One can verify that <EXPR> =>* (a+a) x a.

Techniques for Designing CFGs

- Many CFLs are the union of simpler CFLs. So one can design a CFG for each in turn with start states S_1, S_2,… S_n .Then take the union of the rules and add a new start variable with a rule $S \to S_1 | S_2 | \ldots | S_n$. For example, take the language $\{0^n1^n \mid n>=0\} \cup \{1^n0^n \mid n>=0\}$. First we could make CFGs for each language separately. Say, $S_1 \to 0\ S_1\ 1 | \varepsilon$ and $S_2 \to 1\ S_2\ 0 | \varepsilon$. Then add the rule $S \to S_1 | S_2$.
- A CFG for a language that is regular can be had by first make a DFA for the language. For each state q_i make a variable R_i and for each transition $\delta(q_i ,a) = q_j$ make a rule $R_i \to a\ R_j$. Have the start state variable be R_0. Add rules $R_i \to \varepsilon$ for each final state.

More Techniques for Designing CFGs

- For CFL which contain two substrings which are linked in the sense that a machine for such a language would need to remember information about one on the strings to verify information about the other substring, you might want to consider rules of the form R --> u R v. Here u and v should satisfy the property you are trying to verify.

Ambiguity

- Sometime a grammar can generate string in more than one way.
- Such a string will have several different parse trees. As the parse tree is supposed to give us the meaning, such a string would have more than one meaning.
- A string with more than one parse tree with respect to a grammar is said to be **ambiguously** derived in that grammar.
- For example, consider <EXPR> --> <EXPR>+ <EXPR>| <EXPR> x <EXPR>|(<EXPR>) |a.
- Then a + a x a can be derived with two different parse trees.

Leftmost Derivations

- We want to formalize the notion of ambiguity in terms of derivations rather than parse trees as derivations are easier to work with syntactically.
- We say that a derivation of a string w in a grammar G is a **leftmost derivation** if at every step the leftmost remaining variable is the one replaced.
- A string w is derived **ambiguously** in G if it has two or more different leftmost derivations. A CFG is called **ambiguous** if it generates some string ambiguously.
- There are often many different CFGs for the same language. Even though one of these may be ambiguous some other may be unambiguous. We say a language is **inherently ambiguous** if one can never find an unambiguous CFG for it. One of the problems in the book asks you to prove that $\{a^i b^j c^k | i=j \text{ or } j=k\}$ is inherently ambiguous.

Chomsky Normal Form

- When working with CFGs it is convenient to have them in some kind of normal form in order to do proofs.
- Chomsky Normal Form is often used.
- A CFG is in **Chomsky Normal Form** if every rule is of the form A-->BC or of the form A-->a, where A,B,C are any variables and a is a terminal. In addition the rule S--> ε is permitted.

Conversion to Chomsky Normal Form

Any CFL L can be generated by a CFG in Chomsky Normal Form

Proof Let G be a CFG for L. First we add a new start variable and rule S_0 -->S. This guarentees the start variable does not occur on the RHS of any rule. Second we remove any ε-rules A--> ε where A is not the start variable. Then for each occurrence of A on the RHS of a rule, say R--> uAv, we add a rule R--> uv. We do this for each occurrence of an A. So for R--> uAvAw, we would add the rules R-->uvAw, R--> uAvw, R--> uvw. If we had the rule R-->A, add the rule R--> ε unless we previously removed the rule R--> ε. Then we repeat the process with R. Next we handle unit rule A--> B. To do this, we delete this rule and then for each rule of the form B--> u, we add then rule A-->u, unless this is a unit rule that was previously removed. We repeat until we eliminate unit rules. Finally, we convert all the remaining rules to the proper form. For any rule A--> $u_1u_2 \ldots u_k$ where k>=3 and each ui is a variable or a terminal symbol, we replace the rule with A --> u_1A_1, A_1 --> u_2 A_2, … A_{k-2} --> $u_{k-1}u_k$. For any rule with k=2, we replace any nonterminal with a new variable U_i and a rule U_i --> u_i.