Normal Forms and Parsing

CS154

Chris Pollett

Mar 14, 2007.

Outline

- Chomsky Normal Form
- The CYK Algorithm
- Greibach Normal Form

Chomsky Normal Form

- To get an efficient parsing algorithm for general CFGs it is convenient to have them in some kind of normal form.
- 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--> λ 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 guarantees the start variable does not occur on the RHS of any rule. Second we remove any λ -rules A--> λ 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--> λ unless we previously removed the rule R--> λ. 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--> $u_1u_2 \dots u_k$ 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 terminal with a new variable U_i and a rule U_i --> u_i.

Example

- Use the algorithm to convert: S--> Aba, A-->aab, B--> Ac to Chomsky Normal Form
  - Step 1: Add new start variable to get:
    - S_0-->S, S--> Aba, A-->aab, B-->Ac
  - Step 2: Remove λ rules. In this case, there aren't any so we still have:
    - S_0-->S, S--> Aba, A-->aab, B-->Ac
  - Step 3: Remove unit rules. Only have one, involving S_0.
    - S_0--> Aba, S--> Aba, A-->aab, B-->Ac
  - Step 4: Split up rules with RHS of length longer than 2:
    - S_0--> AC_1, C_1-->ba, S--> AD_1, D_1-->ba, A-->aE_1, E_1-->ab, B-->Ac
  - Step 5: Put each rule with RHS of length 2 into the correct format:
    - S_0--> AC_1,
    - C_1-->B_1A_1 , B_1 -->b, A_1 --> a,
    - S--> AD_1,
    - D_1--> B_2A_2 , B_2 -->b, A_2 --> a,
    - A-->A_3E_1, A_3 --> a
    - E_1--> A_4 B_4, B_4 -->b, A_4 --> a,
    - B-->AC_2 , C_2 -->c

The answer

Introduction to Cocke-Younger-Kasami (CYK) algorithm (1960)

- This is an $O(n^3)$ algorithm to check if a string w is can be generated by a CFG in Chomsky Normal Form.
- As cubic algorithms tend to be slow, in practice people use algorithms based on restricted types of CFGs with a fixed amount of lookahead. Either top down LL parsing or bottom-up LR parsing. These algorithms are based on the PDA model.
- There have been improvements to CYK algorithm which reduce the run-time slightly below cubic ($n^{2.8}$) and to quadratic in the case of an unambiguous grammar.

The CYK algorithm

- The idea is to build a table such that table(i,j) contains those variables that can generate the substring of w start at location i until location j.

Algorithm:

On input $w = w_1 w_2 \ldots w_n$:

1. If w = ε and S--> ε is a rule accept.
2. For i = 1 to n: [set up the substring of length 1 case]
3. For each variable A:
4. Test whether A--> b is a rule, where $b=w_i$
5. If so, place A in table(i,i).
6. For l = 2 to n: [Here l is a length of a substring]
7. For i = 1 to n - l + 1: [i is the start of the substring]
8. Let j = i + l - 1, [j is the end of the substring]
9. For k = i to j-1: [k is a place to split substring]
10. For each rule A-->BC
11. If table(i,k) contains B and table(k+1, j) contains C put A in table(i,j).
12. If S is in table(1,n) accept. Otherwise, reject.

Example

- Consider the context free grammar S-->AT, S-->c, T-->SB, A-->a, B-->b.
- Let's look at the steps CYK would do to check if aacbb was in the language.
- We'll abbreviate table(i,j) as T(i,j)
- First, lines 2-5 would be used to set T(1,1) = {A}, T(2,2) = {A}, T(3,3) = {S}, T(4,4) = {B}, T(5,5) = {B}.
- The l=2 pass of lines 8-11 then fills in the table for substrings of length 2. We get T(1,2)={}, T(2,3)={}, T(3,4) = {T}, T(4,5) ={}. Notice we added T to T(3,4) because T(3,3) was {S} and T(4,4) was {B} and we have a rule T-->SB.
- The l=3 pass of lines 8-11 then fill is the table for substrings of length 3. We get T(1,3)={}, T(2,4)={S}, T(3,5)={}. Here T(2,4)={S}, since T(2,2)={A} and T(3,4)={T} and we have the rule S--> AT
- The l=4 pass of lines 8-11 then fill is the table for substrings of length 4. We get T(1,4)={} and T(2,5)={T}. This follows as T(2,4)={S} and T(5,5)={B} and T-->SB.
- Finally, when the l=5 pass of lines 8-11 then fill is the table for substrings of length 5. i.e., the whole string aacbb. In this case, T(1,5)={S} since T(1,1)={A} and T(2,5)={T} and we have the rule S-->AT. As T(1,5) has the start variable we know the string aacbb is generated by the whole grammar.

Greibach Normal Form

- A CFG is said to be in Greibach Normal Form if all productions are of the form A-->ax where a is a terminal and x is a string of variables (possibly the empty string).
- It turns out that any CFG is equivalent to one in Greibach Normal Form.
- Notice unlike an s-grammar, a grammar in Greibach Normal Form is allowed to have multiple rules with the same (A, a).
- Greibach Normal Form is interesting for two reasons: (1) it can be used in proofs of equivalences of CFL with those languages recognized by a certain type of automaton with a stack. (2) it also gives a slightly different upper bound on the time/space needed to parse a string in a CFG.
- To see notice, a string is generated by a a CFG in Greibach Normal Form then as each rule consumes one string letter, the derivation will be of linear length.
- The brute force algorithm on this string might still take exponential time since if we are currently reading an a, there might be several applicable rules.
- However, as all particular derivations are of linear length, the algorithm is a linear space algortihm.
- Further, if we could nondeterministically guess which rule to apply, we could find verify a derivation in linear time. This shows the context free languages are in nondeterministic linear time.

Example Converting to Greibach Normal Form

- Consider the CFG S-->abSb|aa.
- To convert to GNF we introduce new variables A, B with rules A-->a and B-->b.
- The the grammar

  S--> aBSB|aA

  A--> a

  B-->b

  is in GNF.