Yet More CFLs; Turing Machines

CS154
Chris Pollett
Mar 8, 2006.

Outline

- Algorithms for CFGs
- Pumping Lemma for CFLs
- Turing Machines

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

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 subtring 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.

Languages that are not Context Free

- We can prove languages are not context free by using the Pumping Lemma for context-free languages:

Pumping Lemma for Context Free Languages: If A is a context free language, then there is a number p (the pumping length) where, if s is any string A of length at least p, then s maybe divided into five pieces s= uvxyz satisfying the conditions:

1. for each $i>=0$, uv^ixy^iz is in A.
2. $|vy| > 0$, and
3. $|vxy| <= p$.

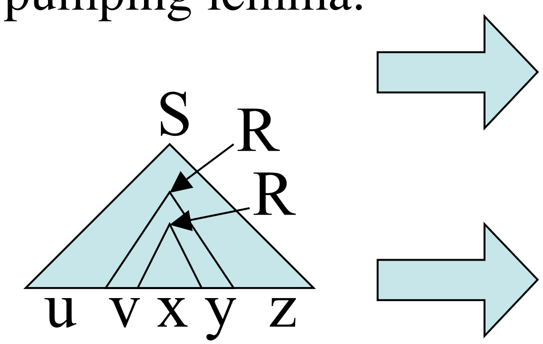
Example use of the CFL Pumping Lemma

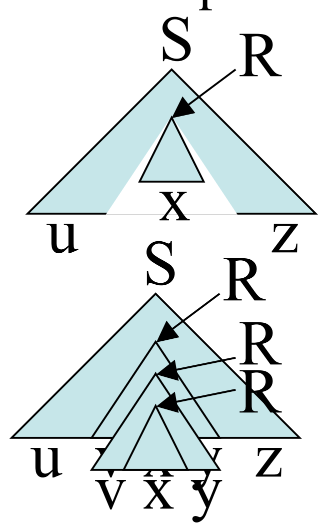
- Let $C = \{a^i b^j c^k \mid 0 <= i <= j <= k\}$
- Argue by contradiction. Let p be the pumping length of C and consider the string $s=a^p b^p c^p$.
- Then s can be written as uvxyz. There are two cases:

1. Both v and y contain only one type of alphabet symbol. So one of a, b, or c does not appear in v or y. So there are three subcases
   a) The a's do not appear. By the pumping lemma, $uv^0xy^0z= uxz$ must be in the language. This string has the same number of a's but fewer b's or c's so cannot be in C giving a contradiction.
   b) The b's do not appear. Then either a's or c's must appear in v and y. If a's appear, then uv^2xy^2z will have more a's then b's giving a contradiction. If c's appear, then uv^0xy^0z will have more b's then c's giving a contradiction.
   c) The c's do not appear. Then uv^2xy^2z will have more a's or b's then c's giving a contradiction.
2. When either v or y contain more than one symbol uv^2xy^2z will not contain the symbols in the right order giving a contradiction.

Proof of the Pumping Lemma for CFGs.

Let G be a CFG for our context free language A. Let |V| be the number of variables in G. Let b be the maximum number of symbols on the right hand side of a rule. So the maximum number of leaves a parse tree of height d can have is b^d. We set the pumping length to $p = b^{|V|}+1$. So if s is in A of length bigger than p, its smallest parse tree must be of height greater than |V|+1. So some variable R must be repeated. So we can do the following kind of surgeries on the parse tree to show condition 1 of the pumping lemma:





Condition 2 of the pumping lemma will hold since if v and y were the empty string then the pumped down tree would be a smaller derivations of s contradicting our choice of parse tree. Condition 3 can be guarenteed by choosing R among the laset |V|+1 of the longest path in the tree.

General Models of Computation

- So far we have looked at machines that either have bounded memory or access to memory limited to stack operations.
- We would like to consider models of computation which correspond to general purpose computers.
- In 1936, Alan Turing presented such a general model of a computer now called a Turing Machine.
- In this model, the machine has a finite control and a arbitrarily long tape of data consisting of squares able to hold one symbol. The machine also has a read head which can read one square at a time. Initially, this tape is black except for the n first squares which have the input. The machine in one step is allowed to read what's under its tape head, write a new symbol, move left or right one square and change its state. The machine has special states for accepting or rejecting.
- The first actual computer developed for code-breaking during World War II was actually based on his model.
- It turns out this model is actually equivalent to what can be done on modern computers.

Example

- Let B be the language {w#w | w is a string over 0,1}
- A Turing Machine M that could accept this language might operate as follows:

On input w:

1. Zig-zag across the tape to the corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found do into the reject state. If they have the same symbol change the symbol to a new symbol X.
2. When all the symbols on the left side of the # have been X'd out, check if there are any more symbols to the right of the #. If yes reject; if not accept.