TD3 - September 28, 2010

1 Syntax Analysis

1.1 Cocke-Younger-Kasami (CYK) Algorithm

The Cocke-Younger-Kasami algorithm determines whether a string can be generated by a given context-free grammar and, if so, how it can be generated. The algorithm is an example of dynamic programming. The standard version of CYK can only recognize languages defined by context-free grammars in Chomsky Normal Form (CNF). However, since any context-free grammar can be converted to CNF without too much difficulty, CYK can be used to recognize any context-free language.

For the sake of simplicity, in what follows we consider grammars where each production has at most two symbols. Example - grammar G :

$$S \to SS$$
$$S \to Ab$$
$$A \to aS$$
$$S \to ab$$

Let w be a string. We define the boolean array $P_w[\alpha, i, j]$ where α is a grammar symbol and $1 \leqslant i \leqslant j \leqslant |w|$ in the following way :

The boolean value $P_w[\alpha, i, j]$ means that α derives the substring of w consisting of the ith through the jth symbols in zero or more steps. Hence we have :

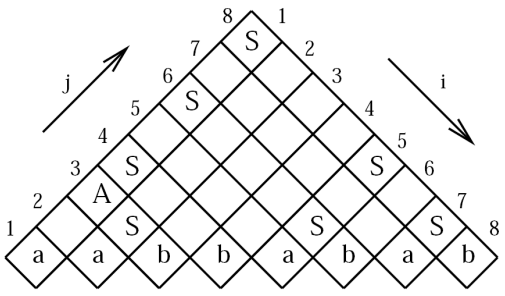
- α is a terminal : if α is the ith symbol of w then $P_w[\alpha, i, i] = true$, else $P_w[\alpha, i, i] = false$. Also, $P_w[\alpha, i, j] = false$ for all $i < j$.

- α is a nonterminal : $P_w[\alpha, i, j] = true$ if there exists some production $\alpha \to \beta\gamma$ and some $i \leqslant k < j$ s.t. $P_w[\beta, i, k] = true$ and $P_w[\gamma, k+1, j] = true$, or if there exists a production $\alpha \to \gamma$ s.t. $P_w[\gamma, i, j] = true$.

How many steps are necessary to compute $P_w[\alpha, i, j]$ for one α ?

What is the complexity to compute G ?

What is the condition that G generates w ? Example : $w = aabbabab$ and the grammar given above. The CYK array is filled in.



Then, consider the context-free grammar G :

$$S \to DE$$
$$D \to HB$$
$$D \to AB$$
$$H \to AD$$
$$E \to CE$$
$$A \to a$$
$$B \to b$$
$$C \to c$$
$$E \to c$$

Apply the CYK algorithm for the string $w = aabbcc$ (you can use a figure similar to the example).

1.2 Top Down Analysis : LL(1)

Exercise 1 Eliminate left recursions and left factors for the following grammar.

$$S \to AEB$$
$$A \to Ax|Ay|Ba|a$$
$$E \to = | \neq$$
$$B \to Ab|b$$

Exercise 2 Consider the following grammar. Note that id, +, [,], and "," are terminals.

$$E \to E + T|T$$
$$T \to id|id[]|id[X]$$
$$X \to E, E|E$$

- Eliminate left recursion in the grammar.
- Perform left factoring for the grammar.
- Compute the `First` set for all symbols in the grammar.
- Compute the `Follow` set for all non-terminals in the grammar.
- Build an LL(1) parser for the grammar.
- Parse the string $id + id[id + id, id[]]$. Show the stack, the input, and the action taken. (hint : `http://csis.pace.edu/~bergin/slides/LLParse.pdf`)
- Build the parse tree while you are parsing. Show your parse tree.

Exercise 3 Consider the following grammar for postfix expressions :

$$E \to EE+$$
$$E \to EE*$$
$$E \to \texttt{num}$$

a. Eliminate left-recursion in the grammar.
b. Do left-factorisation of the grammar produced in question a.
c. Calculate `Nullable` (the nonterminals that can generate the empty string ϵ), `FIRST` for every production and `FOLLOW` for every nonterminal in the grammar produced in question b.
d. Make a LL(1) parse-table for the grammar produced in question b.

Exercise 4 Consider the following grammar :

$$S \to uBDz$$

$$B \to Bv|w$$

$$D \to EF$$

$$E \to y|\epsilon$$

$$F \to x|\epsilon$$

Calculate `Nullable`, `FIRST` and `FOLLOW` sets.
Construct the LL(1) parse-table and give evidence that this grammar is not LL(1).
Give an LL(1) grammar that accepts the same language and built the LL(1) parse table for it.

1.3 Buttom Up Analysis : LR(0)

Exercise 5

Given the grammar

$$S \leftarrow AB\$$$

$$A \leftarrow aA$$

$$A \leftarrow x$$

$$B \leftarrow bB$$

$$B \leftarrow c$$

- Construct the automaton that can represent this grammar (also called Characteristic Finite State Machine for the grammar). Is it LR(0) ? Why or why not ?
- Construct `action` and `goto` tables from the automaton.
- Show the actions of a parser (using the `action` and `goto` tables) when parsing the string "$aaxbc\$$".

Exercise 6

Given the grammar

$$Z \leftarrow E\$$$

$$E \leftarrow T|E+T$$

$$T \leftarrow i|(E)$$

- Prove that this grammar is not LR(0)
- Is this grammar SRL(1) ? If so, give the automaton and the action/goto table.