ICSI 409/509 – Automata Theory

Spring 2025

Group 1 Report: CYK Algorithm Implementation and Testing

Due Date: May 12, 2025

Group Members: Joshua Abreu, Jake Burleski, Collin Gebauer, Michael Levitskiy

Implementation Overview:

Language and Structure:

- Was implemented in Java as file CYK.java which has a main method that reads inputted grammar rules and test strings.
- Grammar is stored in a HashHashMap<String, Set<String>>Production_Rules, and maps each nonterminal to its set of right hand sides.
- Start symbol is captured as the left hand side of the first rule that is entered.

Grammar Input Rules:

Left side must be a nonterminal name with one uppercase letter, optionally also followed by digits, for example S, A1, Z1.

The right side must be exactly one of the following:

- e or the empty string, and only if you are defining the start symbol.
- One terminal character or wany single symbol that is not an uppercase letter.

- Two back to back nonterminals such as AB or X1Y2

If one of these rules is not followed, the program will output “Invalid rule” and ask you to enter it again.

CYK Algorithm (step by step):

- Seed the table: Set an nxn table, where n is the length of the input string. For each single character at ith position, look up all the nonterminals that can produce said character, and fill the cell (i, i) with that set.

-Build up longer spans: Consider gradually substring of length 2, 3, .. etc up to n. For each substring for i to j, split it at all possible muddle k, combine whatever nonterminals were found in (i,k) with those found in (k+1, j). Any rule A to BC that matches a pair means that A can make the whole substring so add it to cell (i, j).

-Decide acceptance: If the start symbol is seen in cell (0, n-1) or the whole string, the grammar makes the string so accept it. Else reject.

Testing Process:

We created a TestCYK.java program that will run many tests in one run. Each test uses a small Case record with three fields:

- filename(String)- This is the CNF grammar

- input(String) - This is string to test

- expected(Boolean) This is where we accept true or false.

How Each Tests Runs:

For every Case in our list:

- Reset the Grammar: Clear out any productions used before, and clear the stored start symbol
- Load the CNF rules from the file: Read and take each LHS to RHS string, split it, and add it to the Production_Rules. The first rules LHS become the start symbol.
- Set the testing String: Assign CYK.input = the case input.
- Handle the empty input specially: If the input is empty “”, we make sure that the start symbol has an epsilon production. Else we call the CYK_Algorithm().
- Check the result: Compare the output of the algorithm to the expected output and print PASS or FAIL next to the input

Complexity Analysis and Optimizations

- Time Complexity is $O(n^3 \cdot P)$ where n is the input length and P is the number of binary productions because of the triple loop.
- Space Complexity: $O(n^2 \cdot V)$ for the CYK table where V is the number of nonterminals.

- Optimizations: This program uses HashMap and HashSets for O of 1 lookups. It also skips over null cells in init.

How to reproduce:

- Place CYK.java , TestCYK.java, and test files in the same directory.
- Compile javac CYK.java TestCYK.java
- Run: java TestCYK
- Run the test file and check the PASS/FAIL table to see which cases succeed or fail.