Analysis of a Java Implementation of the Cocke-Younger-Kasami Algorithm

Group 1
Wyatt Bushman, Thomas Dalbavie, Ryan McSweeney,
Mendel Menachem, Kevin Mieu, Sharie Rhea
December 2, 2024
ICSI 409
Dr. Chinwe Ekenna

1. Introduction

   The Cocke-Younger-Kasami (CYK) algorithm is a parsing algorithm that takes two inputs: a context-free grammar in Chomsky Normal Form (CNF) and a string. The algorithm then determines if the given string can be generated using the provided grammar [1]. To do this, the CYK algorithm constructs a triangular table with size n by n, where n is the length of the input string. This triangular table allows for a dynamic programming approach, making the CYK algorithm one of the most efficient ways to solve the membership problem [1].

   The algorithm first considers substrings of length one. Any rule that produces the substring is recorded in the bottom row of the table. Then, for substrings of length two and greater, the substring is divided into every combination of two partitions. If there is a rule such that $A \rightarrow BC$ where B produces the first partition and C produces the second partition, A is inserted into the table in the appropriate cell to indicate that A produces that substring. By nature, each row contains one fewer cell than the row below it. When the final row–which has length one–is reached, a determination may be made. If the start symbol exists in the top cell then the grammar produces the provided string. If the start symbol does not exist in the top cell then the grammar does not produce the provided string.

2. Implementation

   Upon execution of our implementation, a menu is printed to the user. From this menu the user is prompted to select a grammar insertion method. The user may provide a grammar one line at a time in the terminal or provide a file path to load from. When a user loads a grammar, a new Grammar object is created to model the productions and symbols of the grammar. A grammar object holds a single string representing the non-terminal start symbol, as well as 2 hash sets. One for the terminal symbols, and one for the non-terminal symbols. Finally, a hash map is used to model the production rules of the grammar. The map uses the non-terminal symbol as a key, and the set of productions as a value.

   Once the user's grammar has been loaded, they will be prompted with the option to enter a new grammar, or provide a string for parsing. After the sample string is entered execution is transferred to the cykAlgorithm function. The function takes two parameters: the input string and the previously constructed grammar.

   At the start of the CYK implementation, the length of the input string is checked to ensure that the input is not empty. If the input is empty, the grammar is checked to ensure that epsilon can be derived from the start state.

Once it is confirmed that the input is not empty, a two dimensional n by n grid of Java HashSet objects is created–where n is the length of the input string. This two dimensional array serves as CYK's triangular table. From here, for length n, the bottom row of the table is initialized with productions that produce the input strings terminal symbols.

The remaining loop structure in the program is responsible for filling in the rest of the triangular table, and traversing the table to do so. Sequentially, each set is filled from left to right, bottom to top. Finally, once all squares of the table are resolved, the set at the top of the pyramid is queried to determine whether it is holding the grammar object's start state.

3. Testing

   In order to test our implementation of the CYK algorithm we used Input Space Partitioning (ISP) and a few different coverage criteria. For some grammars, we felt that Base Choice Coverage (BCC) was sufficient. For others we opted for Multiple Base Choice Coverage (MBCC) for a more thorough testing of strings that were not produced by the grammar. Regardless of the coverage criterion chosen, the procedure follows the same basic principles.

   First, partition the input into multiple spaces based on characteristics. Values for each characteristic must be complete and disjoint. In other words, the values must cover the entire input space for that characteristic and there must be no overlap between values. For example, one characteristic might be whether the input string is empty or not with values true and false. Another characteristic might be how many levels of nesting are present with values zero, one, and greater than one. Next, choose a base choice using one value from each characteristic. Usually this is a typical or most common case. Finally, create test requirements by altering one characteristic at a time, keeping all other values the same as the base choice. For MBCC, make more than one base choice and follow the same procedure. Remove any duplicate tests requirements and alter any infeasible requirements.

4. Time Complexity

   The time complexity of our implementation of the CYK algorithm is $O(n^3 \cdot |G|)$, where n is the length of the input string and $|G|$ is the number of production rules in the CNF grammar.

   In the worst case it takes $O(n \cdot |G|)$ to fill one square in the table . The number of squares to be filled is equivalent to $\frac{n(n+1)}{2}$ because only the lower left of the grid is used.

Therefore, the time complexity is $O(\frac{n(n+1)}{2} \cdot n \cdot |G|)$ or $O(\frac{n^3+n^2}{2} \cdot |G|)$ or $O(n^3 \cdot |G|)$.

5. Space Complexity

   The space complexity of our implementation of the CYK algorithm is $O(n^2 \cdot |G|)$, where n is the length of the input string and $|G|$ is the number of production rules in the CNF grammar.

   The only space used for this algorithm is in the triangular table. Our triangular table is a n by n matrix of hash sets. However, only the lower left of this matrix is used. The number of sets used is equivalent to $\frac{n(n+1)}{2}$. Each set may have a maximum of $|G|$ elements in it. Thus, the space used for the triangular table is $\frac{n(n+1)}{2} \cdot |G|$, which simplifies to $\frac{n^2+n}{2} \cdot |G|$ or $O(n^2 \cdot |G|)$. Note that the CYK algorithm must always use each cell of the table and may not terminate early.

   Space is used to store the grammar's terminals, nonterminals, and production rules. However, this is not included in our algorithm analysis because it is part of the setup to run the algorithm, not part of the algorithm itself. For example, testing the membership of multiple strings on the same grammar only requires storing one copy of the grammar.

6. References

[1] J. E. Hopcroft, Rajeev Motwani, and J. D. Ullman, Introduction to automata theory, languages, and computation. Harlow: Pearson, 2014.