Implementation of CYK Parsing Algorithm

Maulik Rawal
Department of Computer Science
Memorial University of Newfoundland
St John's, NL. A1B 3X9
Email: maulikr@mun.ca

***Abstract*—In computer science, the CockeYoungerKasami algorithm (alternatively called CYK, or CKY) is a parsing algorithm for context-free grammars, named after its inventors, John Cocke, Daniel Younger and Tadao Kasami. It employs bottom-up parsing and dynamic programming.**

The standard version of CYK operates only on context-free grammars given in Chomsky normal form (CNF). However any context-free grammar may be transformed to a CNF grammar expressing the same language (Sipser 1997).

The importance of the CYK algorithm stems from its high efficiency in certain situations. Using Landau symbols, the worst case running time of CYK is $O(x^3.|G|)$, where n is the length of the parsed string and —G— is the size of the CNF grammar G (Hopcroft & Ullman 1979, p. 140). This makes it one of the most efficient parsing algorithms in terms of worst-case asymptotic complexity, although other algorithms exist with better average running time in many practical scenarios.

I. Introduction

CYK algorithm is required to determine that whether it is possible to derive the desired string from the given Context Free Grammar.

CYK is still relevant as the simplest example of a family of general parsing algorithm based on dynamic programming, ranging over all parsing technique (that I know of) and many syntactic formalisms. There is a simpler general parsing algorithm (below), but where the dynamic programming (DP) aspect is no longer visible. as things are defined more globally.

Which means general parsing algorithm, is a parsing algorithm that can parse all memeber of a family of languages, according to their given grammar, and produce all the possible parse structures when the parsed string is ambiguous, combined in a condensed form called a (shared-)parse-forest. Though this is often considered only for context-free (CF) grammars, it does generalize to other formalism.

Most known such algorithms are for CF grammars: such as CYK, Earley, GLR, GLL with diverse variants. They are also known as chart parsers, and extend beyond formal languages to some logical formalisms, often introduced by computational linguists via so-called feature structures. They can all be seen as extensions or refinement of CYK, though that is often a subjective assessment. They can also be seen as extensions or refinement of the more basic algorithm presented below.

My experience (but new research might prove me wrong) is that the DP structure of CYK is a proper route to understand, in a simple context, some of the issues raised by the various techniques for improving the performance of those parsers in the structured context of the DP calculation, or to introduce various features such as Viterbi selection of the best parses.

Some of the most basic issues, such as the grammatical forms to be used to keep a low time and space complexity (cubic), or how to represent parse-forests. can be more readily viewed and understood from the simplest algorithm, which is the old cross-product construction (Bar-Hillel, Perles and Shamir, 1961) between a CF grammar and a FSA, to produce the grammar of the CF intersection of their languages. This view is developed in a paper by (Lang 1995) along with extensions to other formal languages.

From a practical point of view, i.e. for use in actual parsers, some of the more recent general parsers may be better tool. But I believe that CYK remains an important and simple educational step, more so in my opinion than Earley's algorithm that is much more complex. Which of the other DP algorithm should be used for actual systems remains the object of some debate, as performance issues are somewhat subtle.

A. Algorithm

Pseudo Code of CYK Algorithm

let the input be a string I consisting of n characters: a_1 ... a_n.
let the grammar contain r nonterminal symbols $R_1 ... R_r$, with start symbol R_1.
let P[n,n,r] be an array of booleans. Initialize all elements of P to false.
for each s = 1 to n
    for each unit production $R_v \rightarrow a_s$
        set P[1,s,v] = true
for each l = 2 to n – Length of span
    for each s = 1 to n-l+1 – Start of span
        for each p = 1 to l-1 – Partition of span
            for each production $R_a \rightarrow R_b R_c$
                if P[p,s,b] and P[l-p,s+p,c] **then** set P[l,s,a] = true
if P[n,1,1] is true **then**
    I is member of language
then
    I is not member of language

In informal terms, this algorithm considers every possible substring of the input string and sets $P[l, s, v]$ to be true if the substring of length l starting from s can be generated from non terminal variable R_v. Once it has considered substrings of length 1, it goes on to substrings of length 2, and so on. For substrings of length 2 and greater, it considers every possible partition of the substring into two parts, and checks to see if there is some production $P \rightarrow QR$ such that Q matches the first part and R matches the second part. If so, it records P as matching the whole substring. Once this process is completed, the sentence is recognized by the grammar if the substring containing the entire input string is matched by the start symbol.

B. Example

Let us assume that we have defined the following grammar.

$$
\begin{aligned}
S &\rightarrow NP\ VP \\
VP &\rightarrow VP\ PP \\
VP &\rightarrow V\ NP \\
VP &\rightarrow eats \\
PP &\rightarrow P\ NP \\
NP &\rightarrow Det\ N \\
NP &\rightarrow she \\
V &\rightarrow eats \\
P &\rightarrow with \\
N &\rightarrow fish \\
N &\rightarrow fork \\
Det &\rightarrow a
\end{aligned}
$$

And using this grammar, the sentence ***she eats a fish with a fork*** needs to be determined.

The algorithm will form the following table, in $P[i, j, k]$, i is the member of the row(starting at the bottom at the 1), and j is the number of the column (starting from the left at 1).

The CYK table for the grammar will be as follows.

CYK table

S						
	VP					
S						
	VP			PP		
S		NP			NP	
NP	V, VP	Det.	N	P	Det	N
she	eats	a	fish	with	a	fork

For readability, the CYK table for P is represented here as a 2-dimensional matrix M containing a set of non-terminal symbols, such that R_k is in $M[i, j]$ if, and

only if, $P[i, j, k]$. In the above example, since a start symbol S is in $M[7, 1]$, the sentence can be generated by the grammar.

C. Generating a parse tree

The above algorithm is a recognizer that will only determine if a sentence is in the language. It is simple to extend it into a parser that also construct a parse tree, by storing parse tree nodes as elements of the array, instead of the boolean 1. The node is linked to the array elements that were used to produce it, so as to build the tree structure. Only one such node in each array element is needed if only one parse tree is to be produced. However, if all parse trees of an ambiguous sentence are to be kept, it is necessary to store in the array element a list of all the ways the corresponding node can be obtained in the parsing process. This is sometimes done with a second table $B[n, n, r]$ of so-called backpointers. The end result is then a shared-forest of possible parse trees, where common trees parts are factored between the various parses. This shared forest can conveniently be read as an ambiguous grammar generating only the sentence parsed, but with the same ambiguity as the original grammar, and the same parse trees up to a very simple renaming of non-terminals, as shown by Lang (1994).

D. Parsing non-CNF context-free grammars

As pointed out by Lange and Leib (2009), the drawback of all known transformations into Chomsky normal form is that they can lead to an undesirable bloat in grammar size. The size of a grammar is the sum of the sizes of its production rules, where the size of a rule is one plus the length of its right-hand side. Using g to denote the size of the original grammar, the size blow-up in the worst case may range from g^2 to 2^{2g}, depending on the transformation algorithm used. For the use in teaching, Lange and Leib propose a slight generalization of the CYK algorithm, "without compromising efficiency of the algorithm, clarity of its presentation, or simplicity of proofs" (Lange and Leib 2009).

E. Parsing weighted context-free grammars

It is also possible to extend the CYK algorithm to parse strings using weighted and stochastic context-free grammars. Weights (probabilities) are then stored in the table P instead of booleans, so $P[i, j, A]$ will contain the minimum weight (maximum probability) that the substring from i to j can be derived from A. Further extensions of the algorithm allow all parses of a string to be enumerated from lowest to highest weight (highest to lowest probability).

F. Valiant's algorithm

The worst case running time of CYK is $\Theta(n^3.|G|)$, where n is the length of the parsed string and —G— is the size of the CNF grammar G. This makes it one of the most efficient algorithms for recognizing general context-free languages in practice. Valiant (1975) gave an extension of the CYK algorithm. His algorithm computes the same parsing table as the CYK algorithm; yet he showed that algorithms for efficient multiplication of matrices with 0-1-entries can be utilized for performing this computation.
Using the CoppersmithWinograd algorithm for multiplying these matrices, this gives an asymptotic worst-case running time of $O(n^{2.38}.|G|)$. However, the constant term hidden by the Big O Notation is so large that the CoppersmithWinograd algorithm is only worthwhile for matrices that are too large to handle on present-day computers (Knuth 1997), and this approach requires subtraction and so is only suitable for recognition. The dependence on efficient matrix multiplication cannot be avoided altogether: Lee (2002) has proved that any parser for context-free grammars working in time $O(n^{3-\epsilon}.|G|)$ can be effectively converted into an algorithm computing the product of (n x n) matrices with 0-1-entries in time $O(n^{(3-\epsilon)/3})$.

II. Example of Implementation

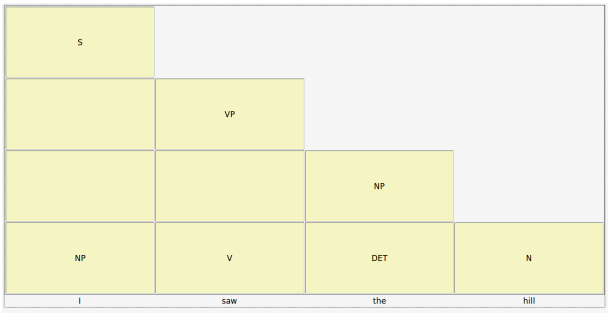
Let us define the following grammar, and we can check we can check whether the desire sentence can be generated with the grammar or not.

Grammar

$$
\begin{aligned}
S &\rightarrow NP \quad VP \\
NP &\rightarrow DET \quad N \\
NP &\rightarrow NP \quad PP \\
PP &\rightarrow P \quad NP \\
VP &\rightarrow V \quad NP \\
VP &\rightarrow VP \quad PP \\
DET &\rightarrow the \\
NP &\rightarrow I \\
N &\rightarrow man \\
N &\rightarrow telescope \\
P &\rightarrow with \\
V &\rightarrow saw \\
N &\rightarrow cat \\
N &\rightarrow dog \\
N &\rightarrow pig \\
N &\rightarrow hill \\
N &\rightarrow park \\
N &\rightarrow roof \\
P &\rightarrow from \\
P &\rightarrow on \\
P &\rightarrow in
\end{aligned}
$$

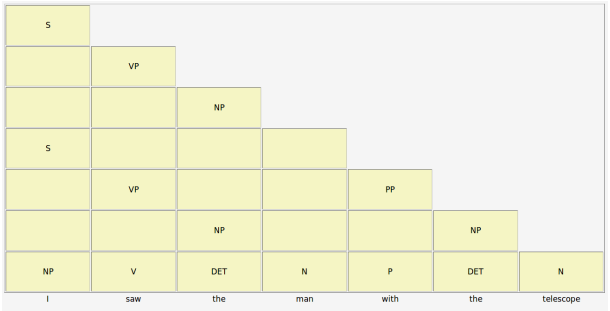
Sentence 1: I saw the hill

Output :



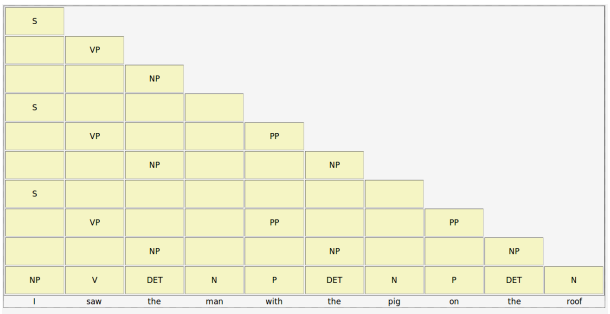
Sentence 2: I saw the man with the telescope

Output :



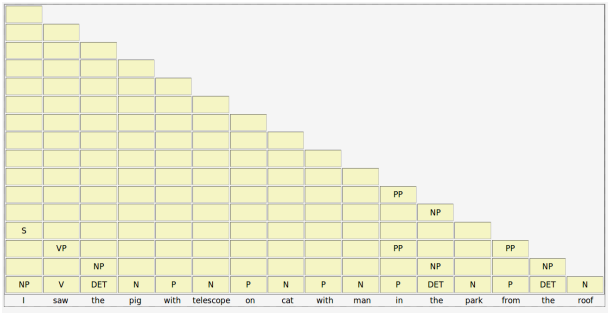
Sentence 3: I saw the man with the pig on the roof

Output :



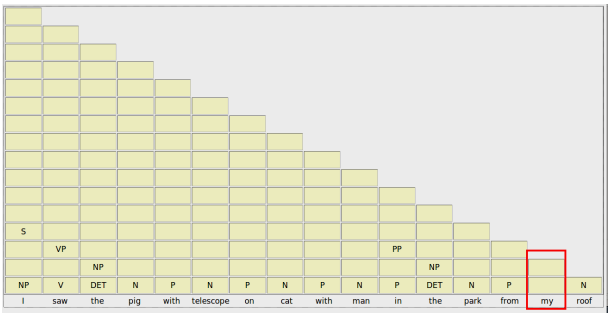
Sentence 4: I saw the pig with telescope on cat with man in the park from the roof

Output :



Sentence 5: I saw the pig with telescope on cat with man in the park from my roof

Output :



In the last senetence, ”my” is not defined in the grammar, hence it can not be parsed. And the desired sentence can not be formed.

REFERENCES

[1] http://cs.stackexchange.com/questions/32320/is-cyk-still-relevant-today
[2] en.wikipedia.org
[3] http://log-cse.blogspot.ca/2014/01/cyk-algorithm-implementation.html
[4] http://martinlaz.github.io/demos/cky.html