CS 4510 : Automata and Complexity
Cocke-Younger-Kasami Algorithm

Subrahmanyam Kalyanasundaram

January 29, 2010

The Cocke-Younger-Kasami (CKY) Algorithm, is an algorithm to determine if a given string is in a language generated by a CFG. The algorithm runs in polynomial time, $O(n^3)$ to be precise, and requires that the CFG is given in the Chomsky Normal Form.

Definition 1 (Chomsky Normal Form)**.** A context-free grammar is in the *Chomsky normal form* if every rule is in the form

$$\begin{aligned} A &\rightarrow BC \\ A &\rightarrow a \end{aligned}$$

where a is any terminal, and A, B and C are any variables – except that B and C may not be the start variable. The rule $S \rightarrow \varepsilon$ is also allowed, where S is the start variable.

The CYK algorithm exploits the fact that all the rules have a special form in the Chomsky normal form to give an efficient algorithm. The algorithm is a dynamic programming algorithm (for those who are familiar with the term).

- Given grammar $G = \{V, \Sigma, R, S\}$ in Chomsky normal form and a string $w = a_1 a_2 \dots a_n$, where $|w| = n$ and $a_i \in \Sigma$.
- The goal is to determine if $w \in L(G)$, efficiently.
- Define $w_{i,j}$ for $1 \leq i \leq j \leq n$ as the substring $w_{i,j} = a_i \dots a_j$. For example, $w_{1,1} = a_1$ the first terminal in w, and $w_{1,n} = w$ the whole string.
- CYK algorithm builds the sets $T_{i,j}$ for all $1 \leq i \leq j \leq n$, where

  $$T_{i,j} = \{A \mid A \stackrel{*}{\Rightarrow} w_{i,j}\}$$

- At the end the algorithm checks if the start state $S \in T_{1,n}$.

  $$S \in T_{1,n} \iff S \stackrel{*}{\Rightarrow} w$$

  which means that $w \in L(G)$.

This is the way the algorithm works. The only part that needs explaining is how the algorithm calculates $T_{i,j}$ for each i, j. As defined before $T_{i,j}$ stands for the set of variables from which the substring $w_{i,j}$ can be derived. T's are calculated for pairs $(i, i+k)$ starting from $k = 0$ (corresponds to substrings of length 1) to $k = n-1$ (the whole string), in this order.

First of all, for $k = 0$, we have $T_{i,i} = \{A \mid A \stackrel{*}{\Rightarrow} w_{i,i} = a_i\}$. This is just the set of all variables A such that there is a rule $A \to a_i$. This is because of the structure obtained from the Chomsky NF. If a variable produces just one terminal, it has to directly produce it. For the general case, we use the values of T's corresponding to shorter substrings to fill up the T's corresponding to longer substrings : to check if $A \in T_{i,i+k}$, the algorithm checks all pairs of variables $B \in T_{i,i+j}$ and $C \in T_{i+j+1,i+k}$ and checks if there is a rule $A \to BC$, for any $0 \leq j \leq k-1$. Again, because of the structure provided by the Chomsky NF, this is enough. If a $k+1$ long substring is derived from A, the first step has to be a two way split into two other variables B and C, where B produces the initial $j+1$ long substring, and C produces the remaining $k-j$ long substring. The algorithm starts with all the $T_{i,i}$'s and then computes $T_{i,i+k}$'s for k from 0 to $n-1$. That is, the algorithm calculates the set of variables that generate all the substrings of length 1, 2 and so on until n. The algorithm pseudocode is given below.

Algorithm 1 CYK Algorithm. Given string $w = a_1 \dots a_n$ and CFG G. Checks if $w \in L(G)$.

```
1:  If w = ε, w ∈ L(G) ⟺ ⟨S → ε⟩ ∈ R(G)          // R(G) is the set of rules of G
2:  for i = 1 to n do
3:      A ∈ T_{i,i} ⟺ ⟨A → a_i⟩ ∈ R(G)           // Constructing T_{i,i}'s
4:  for k = 1 to n − 1 do
5:      for i = 1 to n − k do
6:          for j = 0 to k − 1 do
7:              Check all rules A → BC.
8:              if T_{i,i+j} = B AND T_{i+j+1,i+k} = C then
9:                  T_{i,i+k} = T_{i,i+k} ∪ {A}
10: if S ∈ T_{i,n} then
11:     w ∈ L(G)
12: else
13:     w ∉ L(G)
```

The dominating part of the running time of the algorithm comes from the three nested FOR loops. Each of them can run $O(n)$ iterations. The time taken inside the third FOR loop is proportional to the number of rules r. But for a fixed grammar, r is fixed and can be treated as a constant. So the running time is $O(n^3)$. The correctness is straightforward from the explanation given above.

Let us now see an example. Consider the following grammar G

$$\begin{aligned} S &\to AB \mid AC \\ A &\to BA \mid \texttt{a} \\ B &\to CC \mid \texttt{b} \\ C &\to AB \mid \texttt{c} \end{aligned}$$

How would the CYK algorithm check if the string `baaba` is in the language $L(G)$ or not? It constructs the following table.

5	SAC	SAC	B	SA	AC
4	–	B	SC	B	
3	–	B	AC		
2	SA	AC			
1	B				
	1	2	3	4	5

Note that the first subscript of $T_{i,j}$ denotes the **column** – it is the entry in column i and row j. Since $T_{1,n}$, the top left hand entry, contains S, we conclude that `baaba` $\in L(G)$. The algorithm we described above just concludes if the string is in the language generated by the grammar. But it is not too hard to modify the algorithm so that if the string is in the language, the algorithm finds the parse tree as well. All it needs to do is to keep track, which two variables were used together to get the variable in $T_{i,j}$ for each i, j.