Automata Theory

Tutorial - 2

Contents

- Chomsky Normal Form
- Cocke-Younger-Kasami (CYK) algorithm
- Problems

Chomsky Normal Form

What is the Chomsky Normal Form?

It is a standardised form of the CFG.

Rules for CNF

- A CFG G is in CNF if every rule of G is of the form
 - $Var \rightarrow Var \ Var$
 - $Var \rightarrow ter$
 - Start $Var \rightarrow \epsilon$
- where *Var* can be any variable, including the Start Variable, *Start Var*.

CYK Algorithm

Parsing algorithm for determining whether a string belongs to a language generated by a context-free grammar.

Why an algorithm?

- It is easier to follow an algorithm than to do it by intuition.
- It does so in finite time.

Q: Why is there an importance of CNF?

Prove that a CFG in Chomsky Normal Form has derivations of 2n-1 steps for generating strings $w \in L(G)$ of length n. **Proof:** Note that any CFG in CNF can be written as:

$$A o BC$$
 [B, C are not start variables]
 $A o a$ [a is a terminal]
 $S o \epsilon$ [S is the Start Variable]

We will prove this by **induction**.

(Basic step) Let
$$|w| = 1$$
. Then one application of the second rule would suffice. So any derivation of w would need $2|w| - 1 = 1$ step.

(Inductive hypothesis) Assume the statement of the theorem to be true for any string of length at most k where $k \ge 1$. Now we shall show that it holds for any $w \in L(G)$ such that |w| = k + 1.

Since
$$|w| > 1$$
, any derivation will start from the rule $A \to BC$. So $w = xy$, where $B \stackrel{*}{\Rightarrow} x$, $|x| > 0$ and $C \stackrel{*}{\Rightarrow} y$, $|y| > 0$. But since $|x|, |y| \le k$, and we have that by the inductive hypothesis: (i) number of steps in the derivation $B \stackrel{*}{\Rightarrow} x$ is $2|x| - 1$ and (ii) number of steps in the derivation of $E \stackrel{*}{\Rightarrow} x$ is $E \mid x \mid 0$. So the number of steps in the derivation of $E \mid x \mid 0$.

$$1 + (2|x| - 1) + (2|y| - 1) = 2(|x| + |y|) - 1 = 2|w| - 1 = 2(k + 1) - 1.$$

CYK Algorithm

How does it work? **DYNAMIC PROGRAMMING!**

Q. Let w be the n length string to be parsed. And G represent the set of rules in our grammar with start state S. How to check if the string $w \in L(G)$.

Steps

- Step 1
 - Construct a DP table of size n x n where n is the length of string
- Step 2
 - Handle base case
 - If w = ϵ , if the production rule S -> ϵ exists, accept the string, otherwise reject it.
- Step 3
 - For i = 1 to n:
 - For each variable A:
 - We check if A -> b is a rule and b = wi for some i:
 - If so, we place A in cell (i, i) of our table.

Steps

- Step 4

- For I = 2 to n:
 - For i = 1 to n-l+1:
 - j = j+l-1
 - For k = i to j-1:
 - For each rule A -> BC:
 - We check if (i, k) cell contains B and (k + 1, j) cell contains C:
 - If so, we put A in cell (i, j) of our table.

- Step 5

- We check if S is in (1, n):
 - If so, we accept the string
 - Else, we reject.

Example

Let's say we want to check if w = babba belongs to the grammar G given by

S -> AB | BC

A -> BA | a

B -> CC | b

C -> AB | a

	b	a	a	b	a
b	{B}				
a		{A,C}			
a			{A,C}		
b				{B}	
a	2		3	39	{A,C}

	b	a	a	b	a
b	{B}	{S,A}	Φ	Φ	{S,A,C}
a		{A,C}	{B}	{B}	{S,A,C}
a			{A,C}	{S,C}	{B}
b				{B}	{S,A}
a					{A,C}

Why does it work?

Step-by-step reasoning

- 1. Base case (length 1 substrings)
 - If a substring is just a single terminal (like a), check grammar rules of the form A→a
 - If found, mark that A can generate that 1-character substring.

2. Recursive case (length > 1 substrings)

- For a substring $w[i:i+\ell-1]$ of length ℓ , split it into two parts at position k:
 - Left: w[i:i+k-1]
 - Right: w[i+k:i+ℓ-1]
- o If there's a rule A→BC and:
 - B can produce the left part
 - C can produce the right part
- then AAA can produce the whole substring.

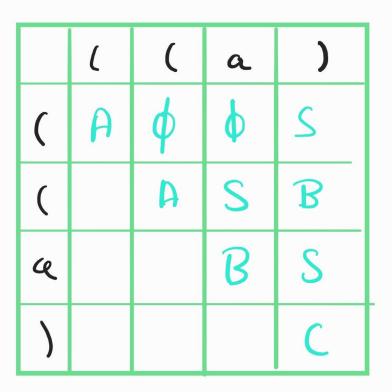
3. Final check

After filling the table, if the start symbol appears for the entire string, it's in the language.

Practice Question

Q. For the grammar given below, check whether the string w = ((a) belongs to it.)

Answer



Resources

https://auto.georgerahul24.in/CYK/index.html

https://www.geeksforgeeks.org/theory-of-computation/cyk-algorithm-for-context-free-grammar/