˜

˜

CS 373: Theory of Computation
Madhusudan Parthasarathy

Lecture 22: CFG membership problem and the CYK Parsing Algorithm

15 April 2010

We will look at the membership problem for CFGs: that of deciding whether a given word w is in the language defined by a context-free grammar, and show the problem is decidable. We will first present a naive simple algorithm that exploits the fact that grammars in Chomsky Normal form have a small parse trees. If $|w| = n$, this gives a membership algorithm that works in time $O(2^n)$ (for a fixed grammar G). Then, we present the more efficient CYK algorithm that works in time $O(n^3)$, using dynamic programming.

1 A naive membership algorithm

In this section, we prove that CNF give very compact parsing trees for strings in the language of the grammar.

In the following, we will need the following easy observation.

Observation 1.1 *Consider a grammar G which is CNF, and a variable X of G which is not the start variable. Then, any string derived from X must be of length at least one.*

Claim 1.2 *Let $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})$ be a context-free grammar in Chomsky normal form, and w be a string of length one. Furthermore, assume that there is a $\mathsf{X} \in \mathcal{V}$, such that $\mathsf{X} \overset{*}{\Longrightarrow} w$ (i.e., w can be derived from X), and let T be the corresponding parse tree for w. Then the tree T has exactly $2\,|w| - 1$ internal nodes.*

Proof: A *full binary tree* is a tree were every node other than the leaves has two children. It is easy to verify by easy induction that such a tree with m leaves, has $m-1$ internal nodes.

Now, the given tree T is not quite a full binary tree. Indeed, the kth leaf (from the left) of T, denoted by ℓ_k, is the k character of w, and its parent must be a node labeled by a variable, X_k, for $k = 1, \ldots, n$. Furthermore, we must have that the parent of ℓ_k has only a single child. As such, if we remove the n leafs of T, we remain with a full binary tree T' with n leafs (every parent of a leaf ℓ_k became a leaf). This tree is a full binary tree, because any internal node, must correspond to a non-terminal derivation of a CNF grammar, and any such derivation has the form $\mathsf{X} \to \mathsf{Y}VZ$; that is, the derivation corresponds to an internal node with two children in T. Now, the tree T' has $n-1$ internal nodes, by the aforementioned fact about full binary trees. As such, T' has $n-1$ internal nodes. Each leaf of T' is an internal node of T, and T' has n such leafs. We conclude that T has $2n-1$ internal nodes. ∎

Alternative proof for Claim 1.2.

Proof: The proof is by induction on the length of w.

If $|w| = 1$ then we claim that w must be derived by a single rule $\mathsf{X} \rightarrow c$, where c is a character. Otherwise, if the root corresponds to a rule of the form $\mathsf{X} \rightarrow \mathsf{YZ}$, then by the above observation, the generated string for Y and Z are each of length at least one, which implies that the overall length of the word is at least 2, a contradiction. (We are using here implicitly the property of a CNF that the start variable can never appear on the right side of a rule, and that the start symbol is the only symbol that can yield the empty string.)

As such, if $|w| = 1$, then the parsing tree has a single internal node, and the claim holds as $2|w| - 1 = 1$.

So, assume that we proved the claim for all words strictly shorter than w, and consider the parse tree T for w being derived from some variable $\mathsf{X} \in \mathcal{V}$. Since $|w| > 1$, it must be that the root of the parse tree T corresponds to a rule of the form $\mathsf{S} \rightarrow \mathsf{X}_1\mathsf{X}_2$. Let w_1 and w_2 be the portions of w generated by X_1 and X_2 respectively, and let T_1 and T_2 denote the corresponding subtrees of T. Clearly $w = w_1w_2$, $|w_1| > 0$ and $|w_2| > 0$, by the above observation. Clearly, T_1 (resp. T_2) is a tree that derives w_1 (resp. w_2) from X_1 (resp. X_2). By induction, T_1 (resp. T_2) has $2|w_1| - 1$ (resp. $2|w_2| - 1$) internal nodes. As such, T has

$$\begin{aligned} N &= 1 + \begin{pmatrix}\#\text{ internal}\\ \text{nodes of } T_1\end{pmatrix} + \begin{pmatrix}\#\text{ internal}\\ \text{nodes of } T_2\end{pmatrix} \\ &= 1 + (2|w_1| - 1) + (2|w_2| - 1) = 2(|w_1| + |w_2|) - 1 \\ &= 2|w| - 1, \end{aligned}$$

■

The following is the same claim, restated as a claim on the number of derivations used.

Claim 1.3 *If G is a context-free grammar in Chomsky normal form, and w is a string of length $n \geq 1$, then any derivation of w from any variable X contains exactly $2n - 1$ steps.*

Theorem 1.4 *Given a context free grammar $\mathcal{G}$, and a word w, then one can decide if $w \in L(\mathcal{G})$ (i.e. there is an algorithm for this problem that always halts).*

Proof: Convert $\mathcal{G}$ into Chomsky normal form, and let $\mathcal{G}'$ be the resulting grammar tree. Let $n = |w|$. Observe that w has a parse tree using $\mathcal{G}'$ with $2n - 1$ internal nodes, by Claim 1.2. Enumerate all such possible parse trees (their number is large, but finite), and check if any of them is (i) a legal parse tree for $\mathcal{G}'$, and (ii) it derives the word w. If we found such a legal tree deriving w, then $w \in L(\mathcal{G})$. Otherwise, w can not be generated by $\mathcal{G}'$, which implies that $w \notin L(\mathcal{G})$. ■

2 CYK parsing

2.1 Discussion

We already saw that one can decide if a word is in the language defined by a context-free grammar, but the construction made no attempt to be practical. There were two problems with this algorithm. First, we converted the grammar to be in Chomsky Normal Form

(CNF). Most practical applications need the parse to show the structure using the original input grammar. Second, our method blindly generated all parse trees of the right size, without regard to what was in the string to be parsed. This is inefficient since there may be a large number of parse trees (i.e., exponential in the length of the word) of this size.

The Cocke-Younger-Kasami (CYK) algorithm solves the second of these problems, using a table data-structure called the ***chart***. This basic technique can be extended (e.g. Earley's algorithm) to handle grammars that are not in Chomsky Normal Form and to linear-time parsing for special types of CFGs.

In general, the number of parses for a string w is exponential in the length of w. For example, consider the sentence "`Mr Plum kill Ms Marple at midnight in the bedroom with a sword`." There are three prepositional phrases: "`at midnight`", "`in the bedroom`," and "`with a sword`." Each of these can either be describing the main action (e.g. the killing was done with a sword) or the noun right before it (e.g. there are several bedrooms and we mean the one with a sword hanging over the dresser). Since each decision is independent, a sentence with k prepositional phrases like this has 2^k possible parses.[1]

So it's really bad to organize parsing by considering all possible parse trees. Instead, consider all substrings in our input. If w has length n, then it has $\Sigma_{k=1}^{n}(n-k+1) = \frac{n(n+1)}{2} = \binom{n+1}{2}$ substrings.[2] CYK computes a table summarizing the possible parses for each substring. From the table, we can quickly tell whether an input has a parse and extract one representative parse tree.[3]

2.2 CYK by example

Suppose the input sentence w is "`Jeff trains geometry students`" and the grammar has start symbol Sand the following rules:

$$\boxed{\begin{array}{ll} \Longrightarrow & \mathsf{S} \rightarrow \mathsf{N}\ \mathsf{V_P} \\ & \mathsf{VN} \rightarrow \mathsf{N}\ \mathsf{N} \\ & \mathsf{V_P} \rightarrow \mathsf{V}\ \mathsf{N} \\ & \mathsf{N} \rightarrow \texttt{students} \mid \texttt{Jeff} \mid \texttt{geometry} \mid \texttt{trains} \\ & \mathsf{V} \rightarrow \texttt{trains} \end{array}}$$

Given a string w of length n, we build a triangular table with n rows and n columns. Conceptually, we write w below the bottom row of the table. The ith column correspond to the ith word. The cell at the ith column and the jth row (from the bottom) of the table corresponds to the substring starting the ith character of length j. The following is the table, and the substrings each entry corresponds to.

[1]In real life, long sentences in news reports often exhibit versions of this problem.

[2]Draw an n by $n+1$ rectangle and fill in the lower half.

[3]It still takes exponential time to extract all parse trees from the table, but we usually interested only in one of these trees.

len				
4	`Jeff trains geometry students`			
3	`Jeff trains geometry`	`trains geometry students`		
2	`Jeff trains`	`trains geometry`	`geometry students`	
1	`Jeff`	`trains`	`geometry`	`students`
	`Jeff`	`trains`	`geometry`	`students`

first word in substring

CYK builds a table containing a cell for each substring. The cell for a substring x contains a list of variables V from which we can derive x (in one or more steps).

	4				
	3				
length	2				
	1				
		`Jeff`	`trains`	`geometry`	`students`

first word in substring

The bottom row contains the variables that can derive each substring of length 1. This is easy to fill in:

	4				
	3				
length	2				
	1	N	N,V	N	N
		Jeff	trains	geometry	students

first word in substring

Now we fill the table row-by-row, moving upwards. To fill in the cell for a 2-word substring x, we look at the labels in the cells for its two constituent words and see what rules could derive this pair of labels. In this case, we use the rules $\mathsf{N} \to \mathsf{N}\ \mathsf{N}$ and $\mathsf{V_P} \to \mathsf{V}\ \mathsf{N}$ to produce:

	4				
	3				
length	2	N	$\mathsf{N},\mathsf{V_P}$	N	
	1	N	N,V	N	N
		Jeff	trains	geometry	students

first word in substring

For each longer substring x, we have to consider all the ways to divide x into two shorter substrings. For example, suppose x is the substring of length 3 starting with "trains". This can be divided into divided into (a) "trains geometry" plus "students" or (b) "trains" plus "geometry students."

Consider option (a). Looking at the lower rows of the table, "students" has label N. One label for "trains geometry" is $\mathsf{V_P}$, but we don't have any rule whose righthand side contains

```
CYK ( $\mathcal{G}, w$ )
        $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})$, $\Sigma \cup \mathcal{V} = \{\mathsf{X}_1, \ldots, \mathsf{X}_r\}$, $w = w_1 w_2 \ldots w_n$.
begin
    Initialize the 3d array $\mathsf{B}[1 \ldots n, 1 \ldots n, 1 \ldots r]$ to FALSE
    for $i = 1$ to $n$ do
        for $(\mathsf{X}_j \to x) \in \mathcal{R}$ do
            if $x = w_i$ then $\mathsf{B}[i, i, j] \leftarrow$ TRUE.
    for $i = 2$ to $n$ do      /* Length of span */
        for $L = 1$ to $n - i + 1$ do      /* Start of span */
            $R = L + i - 1$      /* Current span $s = w_L w_{L+1} \ldots w_R$ */
            for $M = L + 1$ to $R$ do      /* Partition of span */
                /* $x = w_L w_{L+1} \ldots w_{M-1}$, $y = w_M w_{M+1} \ldots w_R$, and $s = xy$ */
                for $(\mathsf{X}_\alpha \to \mathsf{X}_\beta \mathsf{X}_\gamma) \in \mathcal{R}$ do
                    /* Can we match $\mathsf{X}_\beta$ to $x$ and $\mathsf{X}_\gamma$ to $y$? */
                    if $\mathsf{B}[L, M-1, \beta]$ and $\mathsf{B}[M, R, \gamma]$ then
                        $\mathsf{B}[L, R, \alpha] \leftarrow$ TRUE      /* If so, then can generate $s$ by $\mathsf{X}_\alpha$! */
    for $i = 1$ to $r$ do
        if $\mathsf{B}[1, n, i]$ then return TRUE
    return FALSE
```

Figure 1: The CYK algorithm.

$\mathsf{V_P}$ followed by N. The other label for "trains geometry" is N. In this case, we find the rule N $\to$ N N. So one label for x is N. (That is, x is one big long compound noun.)

Now consider option (b). Again, we have the possibility that both parts have label N. But we also find that "trains" could have the label V. We can then apply the rule $\mathsf{V_P} \to$ V N to add the label $\mathsf{V_P}$ to the cell for x.

	4				
	3		N,$\mathsf{V_P}$		
length	2	N	N,$\mathsf{V_P}$	N	
	1	N	N,V	N	N
		Jeff	trains	geometry	students

first word in substring

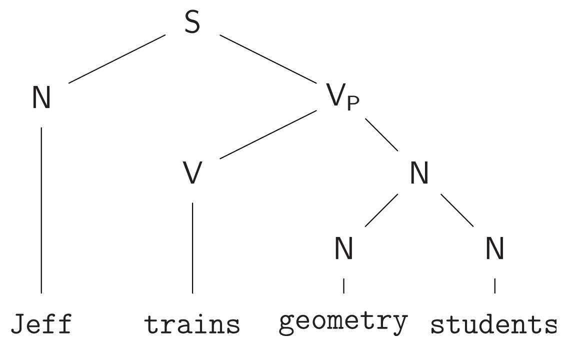
Repeating this procedure for the remaining two cells, we get:

	4	N,S			
	3	N,S	N,$\mathsf{V_P}$		
length	2	N	N,$\mathsf{V_P}$	N	
	1	N	N,V	N	N
		Jeff	trains	geometry	students

first word in substring

Remember that a string is in the language if it can be derived from the start symbol S. The top cell in the table contains the variables from which we can derive the entire input string. Since S is in that top cell, we know that our string is in the language.

By adding some simple annotations to these tables as we fill them in, we can make it easy to read out an entire parse tree by tracing downwards from the top cell. In this case, the tree:



We have $O(n^2)$ cells in the table. For each cell, we have to consider n ways to divide its substring into two smaller substrings. So the table-filling procedure takes only $O(n^3)$ time.

3 The CYK algorithm

In general, we get the following result.

Theorem 3.1 *Let $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, \mathsf{S})$ be a grammar in CNF with $r = |\Sigma| + |\mathcal{V}|$ variables and terminals, and $t = |\mathcal{R}|$ rules. Let $w \in \Sigma^*$ be a word of length n. Then, one can compute a parse tree for w using $\mathcal{G}$, if $w \in \mathsf{L}(\mathcal{G})$. The running time of the algorithm is $O(n^3 t)$.*

The result just follow from the CYK algorithm depicted in Figure 1. Note, that our pseudo-code just decides if a word can be generated by a grammar. With slight modifications, one can even generate the parse tree.