20 The Membership Problem and the CYK Algorithm

Memebership Problem for CFLs: Given a contex-free grammar G and a word w, is $w \in L(G)$?

A similar memebership problem for regular languages, when we are given a dfa A and are asking whether $w \in L(A)$, is trivial, since we can just run A on w, and see whether it stops in a final state. It is not much harder if we are given an nfa instead of a dfa. However, in case of a context-free grammar, it is not quite obvious how to solve this problem. The naive approach would be to systematically apply the productions to generate all strings up to some length (the length of w), and see if w is among them. This method, however, is too time consuming.

The method we use is called *dynamic programming.* The general idea of dynamic programming is to introduce a number of sub-instances of the problem, and compute their solution in order of increasing size. The instances are defined in such a way that the solution for large instances can be computed efficiently from the solutions of smaller instances.

In our application, it works as follows. We assume first that the grammar G is in CNF (otherwise, we can always convert it to CNF.) Let $w = a_1 a_2 \ldots a_n$, where $a_1, \ldots, a_n \in \Sigma$. For all $i \le j$ we define a set $V_{i,j}$ that consists of those variables from which we can derive $a_i a_{i+1} \ldots a_j$, that is:

$$V_{i,j} \quad = \quad \{A \,|\, A \overset{*}{\Rightarrow} a_i a_{i+1} \ldots a_j\}.$$

We will compute all sets $V_{i,j}$ in an appropriate order. Notice that once we have computed all sets $V_{i,j}$, we are done, because $w \in L(G)$ if and only if $S \in V_{1,n}$.

We start by computing all sets $V_{i,i}$. This is easy because each $V_{i,i}$ is just the set of all variables A such that G has the production $A \to a_i$.

Now, for $i < j$, we can express $V_{i,j}$ in terms of some sets $V_{i',j'}$ such that $j' - i' < j - i$. To compute $V_{i,j}$, we need to find all variables A such that $A \overset{*}{\Rightarrow} a_i a_{i+1} \ldots a_j$. Since $i < j$, in the derivation $A \overset{*}{\Rightarrow} a_i a_{i+1} \ldots a_j$ we must have first applied some production $A \to BC$ (because of the Chomsky Normal Form). So this derivation must look like this:

$$A \;\Rightarrow\; BC \overset{*}{\Rightarrow} a_i a_{i+1} \ldots a_j$$

But then B must derive some initial segment of $a_i a_{i+1} \ldots a_k$ and C must derive the rest, that is

$$B \overset{*}{\Rightarrow} a_i a_{i+1} \ldots a_k \quad \text{and} \quad C \overset{*}{\Rightarrow} a_{k+1} a_{l+2} \ldots a_j.$$

for some k. This, in turn, means that $B \in V_{i,k}$ and $C \in V_{k+1,j}$. So we conclude that to determine $V_{i,j}$, we need to find all A's for which there is a production $A \to BC$ where $B \in V_{i,k}$ and $C \in V_{k+1,j}$, for some k.

Since $V_{i,j}$ depends on $V_{i,k}$ and $V_{k+1,j}$, for $k = i, \ldots, j-1$, when we compute $V_{i,j}$ we must have all sets $V_{i,k}$ and $V_{k+1,j}$ already computed. One ordering that will satisfy this property is to vary $j = 2, \ldots, n$, and for each j vary $i = j, j-1, \ldots, 1$. (So we compute the $V_{i,j}$ column by column, each column visited bottom-up.)

The detailed algorithm is given below.

The CYK Algorithm: Let P be the set of productions of G.

```
for $i = 1$ to $n$ do $V_{i,i} := \{A \,|\, A \to a_i \in P\}$;
for $j := 2$ to $n$ do
    for $i := j$ downto $1$ do  begin
    $V_{i,j} := \bigcup_{k=i}^{j-1} \{A \,|\, A \to BC \in P$ for some $B \in V_{i,k}$ and $C \in V_{k+1,j}\}$
    **end** ;
  **if** $S \in V_{1,n}$ then $w \in L(G)$ **else** $w \notin L(G)$
```

Example 1. Let G be the grammar below.

$$\begin{cases} S & \rightarrow & AB \,|\, SS \,|\, a \\ A & \rightarrow & BS \,|\, CD \,|\, b \\ B & \rightarrow & DD \,|\, b \\ C & \rightarrow & DE \,|\, a \,|\, b \\ D & \rightarrow & a \\ E & \rightarrow & SS \end{cases}$$

Let $w = abaab$. We represent the sets $V_{i,j}$ in a matrix, as follows:

$$\begin{array}{cccccc} \text{a} & V_{1,1} & V_{1,2} & V_{1,3} & V_{1,4} & V_{1,5} \\ & \text{b} & V_{2,2} & V_{2,3} & V_{2,4} & V_{2,5} \\ & & \text{a} & V_{3,3} & V_{3,4} & V_{3,5} \\ & & & \text{a} & V_{4,4} & V_{4,5} \\ & & & & \text{b} & V_{5,5} \end{array}$$

According to the algorithm, we start by computing the main diagonal of this matrix, then the second column, third column, and so on:

$$\begin{array}{cccccc} \text{a} & S,C,D & \emptyset & \emptyset & S,E & S,E \\ & \text{b} & A,B,C & A & A,S & A,S \\ & & \text{a} & S,C,D & S,A,B,E & S \\ & & & \text{a} & S,C,D & \emptyset \\ & & & & \text{b} & A,B,C \end{array}$$

For instance, in the initialization phase, we have $V_{1,1} = \{S, C, D\}$ because all these nonterminals produce a in one step. Similarly, $V_{2.2} = \{A, B, C\}$.

In the first iteration, for $j = 2$, we compute the entry $V_{1,2}$. Since $V_{1,1} = \{S, C, D\}$, $V_{2,2} = \{A, B, C\}$, take all combinations of variables from $V_{1,1}$ and $V_{2,2}$: SA, SB, SC, CA, CB, CC, DA, DB and DC. There are no productions with these pairs on the right hand side. So we get $V_{1,2} = \emptyset$.

In the second iteration, for $j = 3$, consider, for example, $V_{2,3}$. We combine variables from $V_{2,2}$ and $V_{3,3}$. These combinations are AS, AC, AD, BS, BC, BD, CS, CC, CD. Of thse pairs, only BS appears in the grammar, in production $A \rightarrow BC$. So $V_{2,3} = \{A\}$. Etc, etc.

In column $j = 4$, consider V_{24}. To compute this entry, we need to combine $V_{2,2}$ with $V_{3,4}$ and $V_{2,3}$ with $V_{4,4}$. In both cases, we find all variable pairs, and then we add the variables on the left-hand sides of the corresponding productions to $V_{2,4}$. The combination of $V_{2,2}$ and $V_{3,4}$ has pairs AS, AA, AB, AE, BS, BA, BB, BE, CS, CA, CB, and CE. Since we have productions $S \rightarrow AB$ and $A \rightarrow BS$, we add A and S to $V_{2,4}$. The combination of $V_{2,3}$ and $V_{4,4}$ has pairs AS, AC and AD. None of these appears in the grammar. We conclude that $V_{2,4} = \{A, S\}$.

In general, when we compute an entry $V_{i,j}$, we combine variables from $V_{i,i}$ with $V_{i+1,j}$, then $V_{i,i+1}$ with $V_{i+2,j}$, then $V_{i,i+2}$ with $V_{i+3,j}$, and so on. For each step, we get a number of variable pairs. For each pair we check if it appears in the productions, and if so, we add the corresponding variables from the left-hand sides to $V_{i,j}$.

When we are done filling the table, we examine the entry $V_{1,5}$. Since $S \in V_{1,5}$, we conclude that $w \in L(G)$.

Example 2. Let now G be the following grammar:

$$\begin{cases} S & \rightarrow & AB \,|\, BC \\ A & \rightarrow & BA \,|\, a \\ B & \rightarrow & CC \,|\, b \\ C & \rightarrow & AB \,|\, a \end{cases}$$

Let $w = baaba$. The table $V_{i,j}$ will look like this:

b	B	S, A	$\emptyset$	$\emptyset$	S, A, C
	a	A, C	B	B	S, A, C
		a	A, C	S, C	B
			b	B	S, A
				a	A, C

Since $S \in V_{1,5}$, we conclude that $baaba \in L(G)$.

For the same grammar, let's also consider $w = bbba$. The table $V_{i,j}$ will look like this:

b	B	$\emptyset$	$\emptyset$	A
	b	B	$\emptyset$	A
		b	B	A, S
			a	A, C

Since $S \notin V_{1,4}$, we conclude that $bbba \notin L(G)$.

Constructing a derivation. Note that the algorithm, as presented, only determines whether w is generated by G, but it does not produce a derivation for w (if $w \in L(G)$). This can be easily fixed. We simply retrace the computation backwards. First, we see what production caused S to be added to $V_{1,n}$. Suppose the production was $S \rightarrow AB$, with $A \in V_{1,k}$ and $B \in V_{k+1,n}$. Thus we know that the first production in the derivation is $S \rightarrow AB$, and that from A we derived $a_1 \ldots a_k$ and from B we derived $a_{k+1} \ldots n$. Now we recursively trace back the derivations from A and B. To make it more efficient, we can record this information at the time when we add variables to the sets $V_{i,j}$. For example, if $A \rightarrow BC$ is a production, and $A \in V_{i,k}$, $B \in V_{k+1,j}$, then can we put an entry $A_{k,B,C}$ into $V_{i,j}$, where the subscripts k, B, C indicate how A was obtained.