Tutorial 10

CS 2AC3

March 24, 2024

1 The Pumping Lemma for CFLs

The Pumping Lemma for CFLs can be used to show that certain sets are not context-free.

Pumping Lemma for CFLs (Contrapositive):
For every CFL A, there exists $k \geq 0$ such that every $z \in A$ of length at least k can be broken up into five sub-strings: $z = uvwxy$, such that $vx \neq \epsilon$ and $\mid vwx \mid \leq k$, there exists an $i \geq 0$, $uv^iwx^iy \notin A$.

Essentially, the demon will pick some $k \geq 0$, we get to choose some $z \in A$ that is at least length k, and then, the demon will pick $uvwxy$ and we get to choose i such that $uv^iwx^iy \notin A$.

1.1 Example 22.4

Use the Pumping Lemma for CFLs to show that $A = \{ww \mid w \in \{a,b\}^*\}$ is not context free. This language can be simplified by intersecting it with the regular set $L(a^*b^*a^*b^*)$, which gets us $A' = \{a^nb^ma^nb^m \mid m, n \geq 0\}$. We can do this since CFLs are closed under intersection with regular sets, and now we can use pumping lemma on this language.

After the demon picks some k, we can pick $z = a^kb^ka^kb^k$, which is in A', and $|z| \geq k$. The demon will then pick some u, v, w, x, y such that $z = uvwxy$, $vx \neq \epsilon$, and $\mid vwx \mid < k$. If we choose $i = 2$, we can win.

If one of v or x contains both a and b, then uv^2wx^2y will not be in the form of $a^n b^m a^n b^m$, so it is not in A'.

Another case that could occur is that v and x are apart of the same block, (a block is a sub-string in the form of either a^n or b^m in this case), which means that one block will be longer than the other 3 blocks in the string.

Finally, there is also the possibility that v and x are in different blocks, so it could be a block of a next to a block of b which are adjacent to each other. Since two of the blocks are a different size compared to the other two blocks, this will not result in a string in the form of $a^n b^m a^n b^m$

Since we were able to find an i that works for all possible cases, we can concludes that A' is not a CFL, which means that A is also not a CFL by the pumping lemma.

2 Complement of CFLs

Generally, CFLs are not closed under complement, but it is possible to get a CFL by taking the complement of a non-CFL language, such as the language in the previous section.

By doing $\{a, b\}^* - \{ww \mid w \in \{a, b\}^*\}$, it results in a set of strings that is of odd length and of the form $xayubv$ or $ubvxay$, where $x, y, u, v \in \{a, b\}*$, and $|x| = |y|$ as well as $|u| = |v|$. To show that this is a CFL, the following grammar can be derived:

$$\begin{aligned}
S &\to AB \mid BA \mid A \mid B \\
A &\to CAC \mid a \\
B &\to CBC \mid b \\
C &\to a \mid b
\end{aligned}$$

In this grammar, A generates all strings in the form of xay where $\mid x \mid = \mid y \mid$. B generates all strings in the form of ubv where $\mid u \mid = \mid v \mid$.

3 The CKY Algorithm

The CKY algorithm is a dynamic programming algorithm that is used to determine if a string x can be created by a particular grammar G in the CNF form.

3.1 Example

Consider the following grammar written in CNF form:

$$
\begin{aligned}
S &\to BS \mid SA \mid AC \mid AB \\
A &\to a \\
B &\to b \\
C &\to SB
\end{aligned}
$$

Check to see if the string 'abaab' can be derived from the above CNF.

The first step is to split this string into sub-parts by creating $n + 1$ lines to split apart each of the characters in the string. In this case, n is the length of the string, which is 5 in this case, so we will need to create 6 lines to partition the string. The string can be split like so: $\mid a \mid b \mid a \mid a \mid b \mid$. Each line is numbered from 0 to n.

For $0 \leq i < j \leq n$, we say that $x_{i,j}$ denotes a sub-string from position i to j. For example, $x_{0,2}$ is the sub-string 'ab'. Now, construct a table with $\binom{n}{2}$ entries for all the possible $x_{i,j}$ pairs.

0					
—	1				
—	—	2			
—	—	—	3		
—	—	—	—	4	
—	—	—	—	—	5

To start, we fill in the top diagonal of the table with the non terminals that result in the terminals in the string. For each sub-string $s = x_{i,i+1}$, if there is a non terminal production X such that $X \rightarrow s \in G$, then we write that non terminal in that spot of the table.

0					
A	1				
—	B	2			
—	—	A	3		
—	—	—	A	4	
—	—	—	—	B	5

Now, we check to see if sub-strings of size 2 can be created. These are the strings in the spots $x_{i,i+2}$. To see if this sub-string can be formed, we look at $x_{i,i+1}$ and $x_{i+1,i+2}$ in the table. Let X represent $x_{i,i+1}$ and Y represent $x_{i+1,i+2}$. If there is a production in G such that $Z \rightarrow XY$, then we can write the non terminal Z in the position $x_{i,i+2}$ in the table, otherwise, we write $\varnothing$.

0					
A	1				
S	B	2			
—	∅	A	3		
—	—	∅	A	4	
—	—	—	S	B	5

Now, for creating strings of length 3 and above, the sections that need to be checked are different. We first check the first entry in the column, and the first entry in the row (entry next to sub string being created). If it is possible to create that sub-string, then we put the associated non-terminal in that spot, otherwise, we check the second term in the column and row. If one of the terms is $\varnothing$, then a sub-string can not be created. If it is not possible to create a sub-string using any of the pairings, then we write down $\varnothing$ in the table.

For example, for $x_{0,3}$, we first check $x_{0,1}$ and $x_{1,3}$, since $x_{1,3}$ is $\varnothing$, we move on to the next pair, which is $x_{0,2}$ and $x_{2,3}$. In this case, we have the pair SA, which can be created using the grammar with $S \to SA$, so we write down S in the spot $x_{0,3}$.

0					
A	1				
S	B	2			
S	∅	A	3		
—	∅	∅	A	4	
—	—	∅	S	B	5

The process outline for strings of length 3 can then be continued until we have filled out the table. In the final table below, we can see that the position $x_{0,5}$ has a C. If this ending position has anything except a S, this means that it is not possible to create this string using the specified grammar. By having the bottom left entry have an S, this implies that the string from $x_{0,n}$ can be derived from the start state, which implies that it can be created from the grammar.

0					
A	1				
S	B	2			
S	$\varnothing$	A	3		
S	$\varnothing$	$\varnothing$	A	4	
C	$\varnothing$	$\varnothing$	S	B	5