CSC301 Lecture 10

1910456 Mir Shafayat Ahmed

August 26, 2021

Regex to CFG

We use the following replacements when converting different operations in regex to CFG.

$$\emptyset = grammar\ with\ no\ rules$$

$$\varepsilon = G \to \varepsilon$$

$$A^* = G \to GA \mid \varepsilon$$

$$A + B = G \to A \mid B$$

$$AB = G \to AB$$

So, to convert (0+1)*111, We can divide it into two parts, one with the ()* and one with the 111 and use the other replacements to find,

$$S \to N111$$

$$N \to N0 \mid N1 \mid \varepsilon$$

Ambiguity

If a string has more than one parse tree, we call the CFG ambiguous. Such as for the string 1+2*2, without the parenthesis, the CFG would be ambiguous.

For a CFG that has a rule $N \to NN$ compared to a rule that says $N \to DN$, there is ambiguity on where to add the next sequence. (To add to the left or to the right).

$$S \to SS \mid x$$

But we can fix CFG of Figure 1 by simply changing it to

$$S \to Sx \mid x$$

And now, subsequent additions tot the tree can only take one path.

Figure 1: Different parse tree for same string

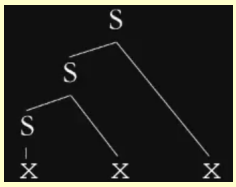

Figure 2: New Unique Parse Tree

Disambiguation

When we are looking to make a language unambiguous, we might not succeed as some languages are inherently unambiguous. Also, there is no general procedure tot follow.

In programming languages, ambiguity comes from not using parenthesis. In spoken languages as well, there can be some ambiguity.

> "She fixed the light on the top floor"

Parsing

For the CFG,

$$S \to 0S1 \mid 1S0S \mid T$$

$$T \to S \mid \varepsilon$$

If we want to build a parse tree for the string '0011', we can go on and on without any final destination.

It is because of a few reasons,

- The string might not be in the language and so we are on a never-ending search.
- For longer strings, the permutation of parse trees would be too many.

We therefore have an idea to:

Stop when the length of a derived string exceeds the required string

But here we encounter two problems:

1. Due to empty transitions, strings containing Variables may turn into empty strings and reduce the length of the string later on.
2. Rules that have Unit transitions like $A \to B$ may continue looping forever.

So, to resolve the two issues, we modify our CFG so that there are no ε and unit productions. After which, **we can be sure that any derivation can only grow the string and not shrink or remain constant in length**.

Removing ε-productions

If the start variable is nullable,

$$S \to \varepsilon$$

we add a new start variable S' and add a new production,

$$S' \to S \mid \varepsilon$$

For other nullable variables,

1. We identify a nullable variable and remove the ε-production
2. We look for references of that variable in other productions and replace that reference with what would've happened if the nullable variable wasn't removed.

Example

For the CFG,

$$\begin{aligned} S &\to ACD \\ A &\to a \\ B &\to \varepsilon \\ C &\to ED \mid \varepsilon \\ D &\to BC \mid b \\ E &\to b \end{aligned} \tag{1}$$

We identify B as a nullable variable, so we remove $B \to \varepsilon$. Now we look for references and see that B is referenced in $D \to BC$. So we replace B with what it was supposed to be, i.e 'empty'. And so, we get,

$$\begin{aligned} S &\to ACD \\ A &\to a \\ C &\to ED \mid \varepsilon \\ D &\to C \mid b \\ E &\to b \end{aligned} \tag{2}$$

Now we look for more nullable variables and see that C is nullable. So we remove $C \to \varepsilon$. We see that C is referenced in the beginning. so we change it to $S \to AD$. But, now we cannot form something like 'AEDD' that was possible before, therefore we keep $S \to ACD$ as well.
D also references C, so we replace it with $D \to \varepsilon$. But again, D could be ED before, so we keep $D \to C$

$$\begin{aligned} S &\to ACD \\ S &\to AD \\ A &\to a \\ C &\to ED \\ D &\to C \mid b \mid \varepsilon \\ E &\to b \end{aligned} \tag{3}$$

We repeat and remove $D \to \varepsilon$. Now $S \to ACD$ can be $S \to AC$ as well. For the other reference, the same logic applies and we end up with,

$$\begin{aligned} S &\to ACD \mid AC \\ S &\to AD \mid A \\ A &\to a \\ C &\to ED \mid E \\ D &\to C \mid b \\ E &\to b \end{aligned} \tag{4}$$

Finally we see that there aren't any nullable variables anymore. So we have solved our first problem regarding disambiguation.