1 Give an example CFG which has useless productions, a nullable variable, and a unit production. For this grammar show the result of applying the algorithm from class for eliminating each of these.

Consider the CFG:

$S \Rightarrow aCb$

$B \Rightarrow aCb$] — claim is a useless production ⊛

$C \Rightarrow aCbCa$

$C \Rightarrow D$] — unit production

$D \rightarrow \varepsilon$

C, D are nullable: $C \Rightarrow^* \varepsilon$ as $C \Rightarrow D \Rightarrow \varepsilon$, and $D \Rightarrow \varepsilon$ shows D nullable.

To show ⊛ we apply the algorithm from class to check for useful variables:

Step 1 set $V_1 = \{\}$

Step 2 cycle over variable checking if they occur as LHS of a rule whose right hand side variables are already in V_1.

Pass 1 D satisfies this via rule $D \Rightarrow \varepsilon$. So $V_1 = \{D\}$

Pass 2 C satisfies this via $C \Rightarrow D$ and b/c $D \in V_1$ So $V_1 = \{C, D\}$

Pass 3 S and B added since $S \Rightarrow aCb$ & $B \Rightarrow aCb$ and $C \in V_1$ $\therefore V_1 = \{S, B, C, D\}$ Next pass no change so done

Step 3 see w/c variables reachable from start variable

Initialize $V_2 = \{S\}$

Pass 1 C is added to V_2 b/c all variables on RHS of $S \Rightarrow aCb$ in V_1. So now $V_2 = \{S, C\}$

Pass 2 D is added to V_2 b/c all variables on RHS of $C \Rightarrow D$ in V_1 & $C \in V_2$. So now $V_2 = \{S, C, D\}$

Next pass no change, so $V_2 = \{S, C, D\}$ is the set of useful variables.

∴ B is useless & $B \Rightarrow aCb$ is a useless production w/c we can delete ⇒

(1 cont'd)

After deleting useless productions our grammar is.

$S \to aCb$

(**) $C \to aCbCa$

$C \to D$

$D \to \varepsilon$

To delete ε-rules we compute the set of nullable variable.

Pass 1 Look for rules of form $V \to \varepsilon$ add V to nullable set.

After this pass $N = \{D\}$.

Pass 2 For each rule $X \to X_1 \ldots X_n$ where each $X_i \in N$ add X to N.

After this pass $N = \{C, D\}$. No further passes change this.

Now to eliminate ε-rules we do substitutions for the nullable variables and eliminate ε rules.

After eliminating ε-rules and substituting for D have

$S \to aCb$

$C \to aCbCa \mid abCa \mid aCba \mid aba$

$C \to \varepsilon$

After eliminating ε-rules and substituting for C have

$S \to aCb \mid ab$

$C \to aCbCa \mid abCa \mid aCba \mid aba$

Final answer.

Note: if the start variable has an ε-rule we can't in general eliminate it with this technique.

Note: we eliminated nullable variables in the order we substituted in the order we found nullable variables.

This grammar no longer has unit rules. To see the algorithm for eliminating unit rules in action we work with the grammar (**) above.

$\Rightarrow$

(I cont'd)

Let's eliminate unit rules from

$S \to aCb$

$C \to aCbCa$

$C \to D$

$D \to \varepsilon$

Step 1 Use reachability algorithm to determine pairs (X, Y_i) for which a unit derivation $X \Rightarrow^* Y_i$ exists.

In this case, get $\{(C, D)\}$.

Step 2 For each such pair (X, Y_i) if Y_i appears in the rules $Y_i \to y_1 | y_2 \cdots | y_n$ add the rule $X \to y_1 | y_2 | \cdots | y_n$, then delete any unit rule involving X on the LHS.

In our case, this gives:

$S \to aCb$

$C \to aCbCa$

$C \to \varepsilon$. ← after unit rules eliminated.

Note: as describe algorithm is too vague...

Consider

$A \to B$

$B \to A$

$A \to a$

We get pairs $\{(A,B), (B,A)\}$

Eliminating 1st pair

$A \to A$

$B \to A$

$A \to a$

Eliminating 2nd pair gives

$A \to A$

$B \to A$ → could be reintroduced as $B \to A$, $A \to A$ so need to keep track so doesn't happen

$B \to a$

$A \to a$

→ if kept track have

$A \to A$ ← would want to scan to find the pair (A, A) not in original set.

$B \to a$

$A \to a$

Eliminating (A, A) gives

$B \to a$ ↳ again $A \to A$, $A \to A$ would say to add back so keep track.

$A \to a$

2 Consider the grammar $S \to 0 \mid 0S0 \mid 1S0 \mid 0S1$. Use the algorithm from class to step-by-step convert this grammar to CNF.

Step 1 Add a new start variable

$S_0 \to S$, $S \to 0 \mid 0S0 \mid 1S0 \mid 0S1$

Step 2 Eliminate ε-rules.

No such rules, so no change

Step 3 Eliminate unit productions.

$S_0 \to 0 \mid 0S0 \mid 1S0 \mid 0S1$ (using each thing S could go to with S_0 now on LHS)

$S \to 0 \mid 0S0 \mid 1S0 \mid 0S1$

Step 4 Convert rules of length > 2

$S_0 \to 0 \mid 0A_1 \mid 1A_2 \mid 0A_3$

$A_1 \to S0$

$A_2 \to S0$

$A_3 \to S1$

$S \to 0 \mid 0A_4 \mid 1A_5 \mid 0A_6$

$A_4 \to S0$

$A_5 \to S0$

$A_6 \to S1$

Step 5 Put binary rules in proper form

Final Answer

$S_0 \to 0 \mid B_1A_1 \mid B_2A_2 \mid B_3A_3$

$B_1 \to 0$

$B_2 \to 1$

$B_3 \to 0$

$A_1 \to SB_4$

$B_4 \to 0$

$A_2 \to SB_5$

$B_5 \to 0$

$A_3 \to SB_6$

$B_6 \to 1$

$S \to 0 \mid B_7A_4 \mid B_8A_5 \mid B_9A_6$

$B_7 \to 0$

$B_8 \to 1$

$B_9 \to 0$

$A_4 \to SB_{10}$

$B_{10} \to 0$

$A_5 \to SB_{11}$

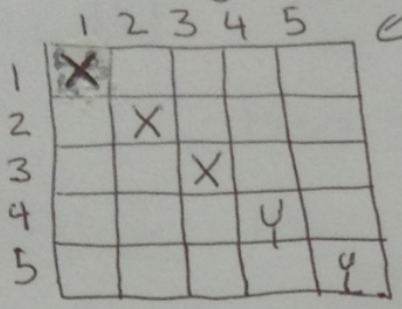
$B_{11} \to 0$

$A_6 \to SB_{12}$

$B_{12} \to 1$

3 Show step-by-step the result of applying the CYK algorithm of the previous problem on the following strings: 00011 and 11.

1st string 00011

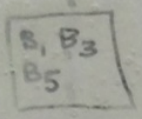


Initialize diagonal (i,1) w/ ways ith char of string could be generated.

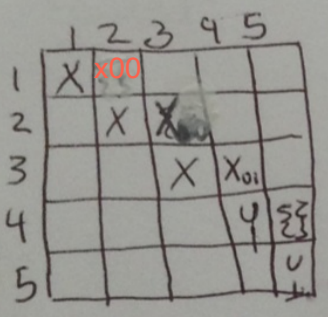
Let

$X = \{B_1, B_3, B_4, B_5, S_0, S, B_7, B_9, B_{10}, B_{11}\}$ ← generate 0

$U = \{B_2, B_6, B_8, B_{12}\}$ ← generate 1



pass 1 how substrings of length 2 could be generated.

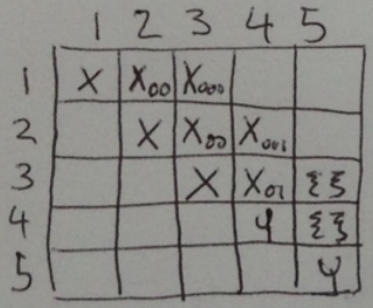


Let

$X_{00} = \{A_1, A_2, A_4, A_5\}$

$X_{01} = \{A_3, A_6\}$

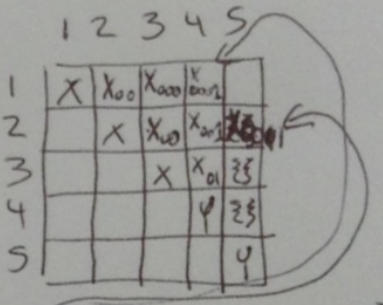
Pass 2 how substrings of length 3 could be generated



$X_{000} = \{S_0, S\}$

$X_{001} = \{S_0, S\}$

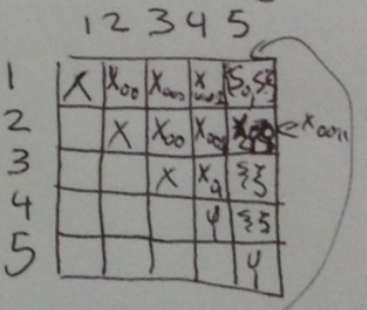
Pass 4 Substrings of length 4



$X_{0001} = \{A_3, A_6\}$

$X_{001} = \{A_3, A_6\}$

Pass 5 the whole string



has S_0 so accept.

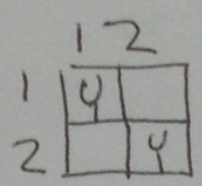
this entry is witnessed by

$S_0 \Rightarrow B_3 A_3$

$\in X \quad \in X_{0011}$

3 cont'd

Step-by-step CYK on 11



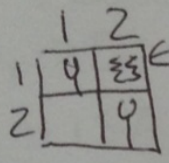
Initialize diagonal with ways ith char of string could be generated

Here $V_1 = \{B_2, B_6, B_8, B_{12}\}$

Pass 2 How substring of length 2 could be generated.

Since no rule in our grammar involves ~~two~~ elements of V_1 there aren't any ways

So final table looks like



← since no start symbol

11 is not generated by our grammar

4 Convert the grammar $S \to 0 | 0S0 | 1S0 | 0S1$ to Greibach Normal Form.

Since we didn't give an algorithm for this in class, we can be somewhat ad hoc. Basically, though if all our rules already begin with terminals (as above), we can introduce new variable for each terminal after the first in a rule to convert it to GNF.

For example,

$S \Rightarrow 0S0$ would become $S \Rightarrow 0SA_1$, $A_1 \to 0$.

Applying this to our grammar yields:

$S \to 0$

$S \to 0SA_1$

$A_1 \to 0$

$S \to 1SA_2$

$A_2 \to 0$

$S \to 0SA_3$

$A_3 \to 1$

We note if our grammar had rules of the form

$A \Rightarrow B$ · some string of variables & terminals, the above method would be insufficient. We would need to somehow "expand the lead variable" B till we got (each possible) lead terminal strings generatable from B.