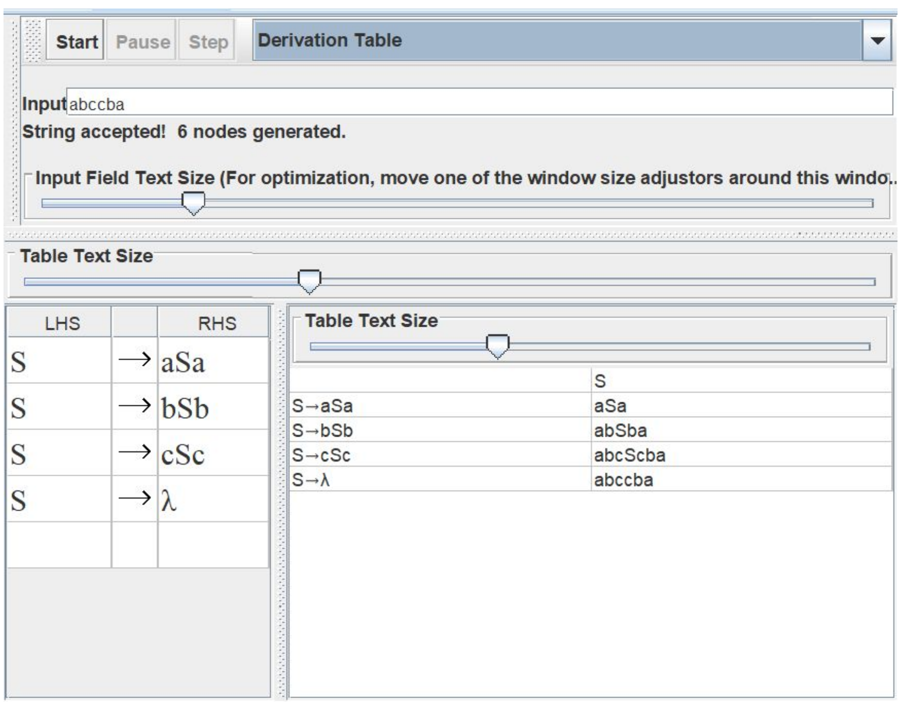
Homework #3

1. I am defining the CFG “G” to be a 4-tuple over the alphabet {a,b,c} and with the production S -> aSa | bSb | cSc | ∈ which describes strings in the language {ww^R | w in {a,b,c}*} (In this language R means reverse string). The brute force parsing algorithm to verify if a string is in a CFG simply looks at each level of derivation from the start variable and checking to see if that string is the string we want to except or if the number of derivations from the start is past the length of the string that we want and we still haven’t found it. For the case of the string “abccba” the algorithm starts at the first derivations from the start variable S which would be **aSa, bSb, cSc or λ**. For these derivations in step 1, we check to see if the string we want is produced and it is not so we keep going. For a 2 step derivation we see **aaSaa, abSba, acSca, aa, baSab, bbSbb, bcScb, bb, caSac, cbSbc, ccScc or λ.** Again, the string is not there and we also see the string **λ** again so we prune that branch completely. This continues for this string until it is finally accepted. The derivation table shown below shows only the branch that would be accepted by the algorithm and excludes the others that stem from bSb and cSc.



2. Give with proof an example of a non-regular language accepted by an *s*-grammar.

We need to prove that a non-regular language can be accepted by an *s*-grammar.

Let L={w ∈ {a,b}* : w = a^n b^n, for some n ≥ 0 } where we all know that this language is not regular because if we process a string w ∈ L, we need to remember how many a's we have seen, and then we need to count the number of b's to compare with a's and accept w if and only if the numbers are equal.

- We can prove it by using the pumping lemma.

* Assume is a DFA that recognizes L
* Let p be M's pumping length
* Consider the string w = a^p b^p This string is in the language and has length greater than p.
* So by the pumping lemma w=xyz, where |xy|≤p, |y|>0, and where xy^iz is in the language for all i≥0. That means x=a^k and y= a^j where k+j ≤ p and j > 0. But then taking i=0, xz = $a^{(p-j)}$ b^p should be in C. As p−j is not equal to p this gives a contradiction. So C is not regular.

- By using L, we can write the following s-grammar

S -> aSb| ∈

S -> aSb

S -> aSB

B -> b

S ->∈

-Every production has a single terminal on the R hand side and has a combination of non-terminal.
-There is a single terminal on the left hand side.
-Every pair of non terminals appears once.

3. Use the algorithm from class to convert the following CFG to Chomsky Normal Form.

S -> B | ε| aS | IS

I -> iS | iSeS

B -> {S}

Use the algorithm from class to convert the above CFG to Chomsky Normal Form.

Step 1: Add a new start variable to get:

S0 -> S

S -> B | ε | aS | IS

I -> iS | iSeS

B -> {S}

Step 2: Remove ε rules.(S->ε)

S0 -> S | ε

S -> B | aS | IS | a | I

I -> iS | iSeS | i | iSe | ie | ieS

B -> {S}|{}

3: Remove unit rules.   We will substitute S in S0

S0 -> B | aS | IS | a | I | ε

S -> B | aS | IS| a | I

I ->iS | iSeS | i | iSe | ie | ieS

B -> {S} | {}

Now, we will substitute I in S0 and S

S0 -> {S} | {} | aS | IS | a | iS | iSeS | i | iSe | ie | ieS | ε

S-> {S} | {} | aS | IS| a | iS | iSeS | i | iSe | ie | ieS

I -> iS | iSeS | i | iSe | ie | ieS

B -> {S} | {}

Step 4: Split up rules with RHS of a length longer than 2

S0 -> {A | {} | aS | IS | a | iS | iC | i | iF | ie | iD | ε

S-> {A | {} | aS | IS| a | iS | iC | i | iF | ie | iD

I -> iS | iC | i | iF | ie | iD

B -> {S} | {}

A->S}

C->SD

D->eS

F->Se

Step 5: Put each rule with RHS of length 2 into the correct format:
S0 -> {A | {} | A1S | IS | A1 | I2S | I2C | i | I2F | I2E | I2D | ε

S-> {A | {} | A1S | IS| A1 | I2S | I2C | i | I2F | I2E | I2D

I -> I2S | I2C | i | I2F | I2E | I2D

A1->a

I2->i

E->e

B -> {S} | {}

A->S}

C->SD

D->ES

F->SE

The grammar after Step 5 is the final answer.

4. Show how to modify the CYK algorithm to work for grammars in 2NF.

In class we saw that CYK algorithm works as follow

On input w=w1w2...wn:

1. If w=ε and S→ε is a rule, accept.
2. For i=1 to n: [set up the substrings of length 1 case]
3. For each variable A:
4. Test whether A→b is a rule
5. If so, place A in table(i,i).
6. For l=2 to n:[Here l is a possible length of a substring]
7. For i=1 to n−l+1: [i is the start of the substring]
8. Let j=i+l−1, [j is the end of the substring]
9. For k=i to j−1: [k is a place to split the substring]
10. For each rule A→BC
11. If table(i,k) contains B and table(k+1,j) contains C put A in table(i,j).
12. If S is in table(1,n) **accept**. Otherwise, **reject**.

This algorithm is really good for context-free grammars, but in the case of grammars in 2NF form (cases where A-> xy where *x* and *y* can be either a single variable or a single terminal are also allowed) it does not work. We will modify the CYK algorithm to allow those rules.

If we do the following modification it is going to work:

First,

Let G be a grammar in 2NF. The unit relation UG and its inverse,ÛG: = {(y,A) | (A,y) ∈ UG},can be computed in time and space O(|G|).

We view ÛG as a relation on V and call IG = (V,ÛG) the inverse unit graph of G.

Now, Let G = (N,Σ,S,→) be a grammar. Then for all x,y ∈ V we have: x ⇒* y iff (x,y) ∈ UG*.

After we have created G.

Let G = (N,Σ,S,→) be a grammar in 2NF, nd let (V,ÛG) be its inverse unit graph

Given a grammar G = (N,Σ,S,→) in 2NF, its graph IG and a word w ∈ Σ+, Algorithm CYK decides in time O(|G|·|w|3) and space O(|G|·|w|2) whether or not w ∈ L(G).

If we let the input: a CFG G = (N,Σ,S,→) in 2NF, its graph (V,ÛG),a word w = a1...an ∈ Σ+

CYK(G,ÛG,w) =

```
for i=1,...,n do
    T(i,i) : = Û*G({ai}),"we need to check if the ath character is also a variable and add it to table i, i"
    for j=2,...,n do
        for i=j-1,...,1 do
            T' (i,j) : = ∅, we let the table T i,j equals to empty
            for h=i,...,j-1 do
                for all A → yz
                    if y ∈ T(i,h) and z ∈ T(h+1,j) then
                        T' (i,j) : = T' (i,j) ∪{ A }
            T(i,j) : = Û*G(T' (i,j)) we will then add it to table i,j
    if S ∈ T(1,n) then return yes else return no
```

It will decide whether or not w ∈ L(G).

I marked all the modifications to the algorithm in **bold**. As you can see, we will pretend the ath character is also a variable and add it to the table. On the next modification we will let the table T’ equals to empty, and lastly if it is in the for loop we will add it to the table i,j.