20/20

1-2. Transform, step by step, the grammar with rules S → ε, S → aSb, and S → bSa to Chomsky normal form. Show how the word baba will be generated in the resulting Chomsky-normal-form grammar.

$\cancel{S \to \varepsilon}$
$S \to aSb$
$S \to bSa$
0) $S_0 \to S$

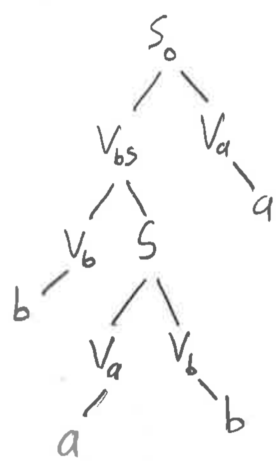
1)
$S \to aSb$
$S \to bSa$
$\cancel{S_0 \to S}$
$S \to ab$
$S \to ba$
$S_0 \to \varepsilon$

2)
$S \to aSb$
$S \to bSa$
$S \to ab$
$S \to ba$
$S_0 \to \varepsilon$
$S_0 \to aSb$
$S_0 \to bSa$
$S_0 \to ab$
$S_0 \to ba$

3)
$V_a \to a$
$V_b \to b$
$S \to V_a S V_b$
$S \to V_b S V_a$
$S \to V_a V_b$
$S \to V_b V_a$
$S_0 \to \varepsilon$
$S_0 \to V_a S V_b$
$S_0 \to V_b S V_a$
$S_0 \to V_a V_b$
$S_0 \to V_b V_a$

4)
$V_{aS} \to V_a S$
$V_{bS} \to V_b S$
$V_a \to a$
$V_b \to b$
$S \to V_{aS} V_b$
$S \to V_{bS} V_a$
$S \to V_a V_b$
$S \to V_b V_a$
$S_0 \to \varepsilon$
$S_0 \to V_{aS} V_b$
$S_0 \to V_{bS} V_a$
$S_0 \to V_a V_b$
$S_0 \to V_b V_a$

Tracing baba

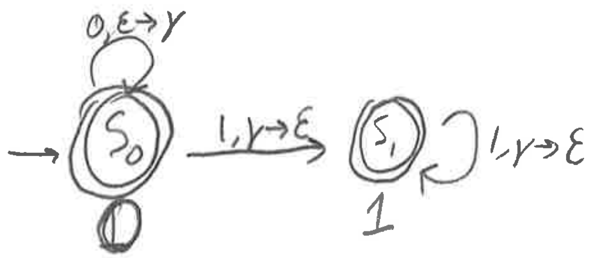


20/20

3-4. Use a general algorithm for transforming PDA into CFG to design a CFG that corresponds to the following pushdown automaton. This automaton has two states: the starting 0-state s_0 and the 1-state s_1. Both states are final. The transitions are:

- From s_0 to s_0, the transition is $0, \varepsilon \to y$, for some symbol y
- From s_0 to s_1, the transition is $1, y \to \varepsilon$.
- From s_1 to s_1, the transition is $1, y \to \varepsilon$.

Show, step by step, how the word 01 will be generated by the resulting grammar.



1) $A_{00} \to \varepsilon$
$A_{11} \to \varepsilon$

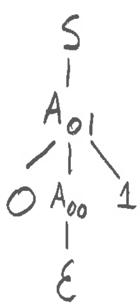
3) $0 \xrightarrow{0,\varepsilon\to y} 0 \qquad 0 \xrightarrow{1,y\to\varepsilon} 1$

$A_{01} \to 0 A_{00} 1$

$0 \xrightarrow{0,\varepsilon\to y} 0 \qquad 1 \xrightarrow{1,y\to\varepsilon} 1$

$A_{01} \to 0 A_{01} 1$

- Tracing 01



9/10

5. Use the general stack-based algorithms to show:

- how the compiler will transform the expression (1 − 10) / (8 − 11) into postfix form, and
- how it will compute the value of the resulting postfix expression.

(1 - 10) / (8 - 11)

1 10 - 8 11 - /

| 1 | 10 | - | 8 | 11 | - | / |

-3

10|10

6. Illustrate the pumping lemma for context-free grammars by showing how it will represent the word w = +1.00 which is generated by the CFG with starting variable V and rules V → SN.N, V → N.N, S → +, S → −, N → DN, N → D, D → 0, and D → 1 as uvxyz. Show, step-by-step, how the corresponding word uvvxyyz can be derived from this CFG.

$V \to SN.N$

$V \to N.N$

$S \to +$

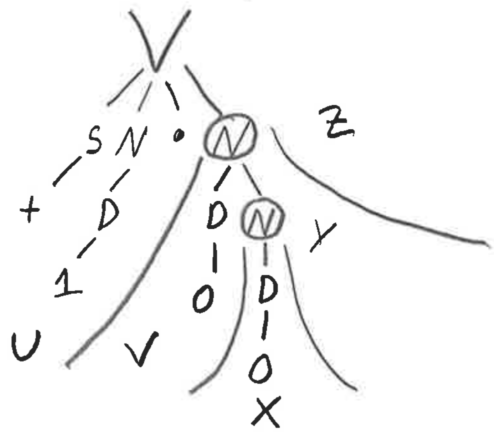
$S \to -$

$N \to DN$

$N \to D$

$D \to 0$

$D \to 1$



$u = +1.$

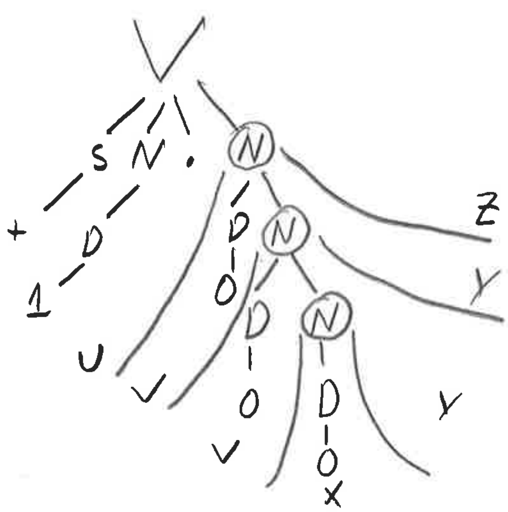
$v = 0$

$x = 0$

$y = \Lambda$

$z = \Lambda$

Tracing/Deriving uvvxyyz



uvvxyyz = +1.000

10/10

7. Prove that the language of all the words of the type $a^nb^nc^nd^n$, $n = 0, 1, 2, \ldots$, is not context-free.

Assume L is CFG and $L = a^nb^nc^nd^n$. Then for every p, let's take

$w = a^pb^pc^pd^p = \underbrace{a\ldots a}_{p \text{ times}}\underbrace{b\ldots b}_{p \text{ times}}\underbrace{c\ldots c}_{p \text{ times}}\underbrace{d\ldots d}_{p \text{ times}}$ so that $len(w) = 4p$.

Thus for every u, v, x, y, z such that $w = uvxyz$ and $len(vxy) < p$. So we have one of the following cases:

1) vxy is in a's, so by pumping lemma we add a's but not b's, c's or d's, so $uvvxyyz \notin L$

2) vxy is in a's and b's, so by pumping lemma, we add a's & b's, but not c's or d's, so $uvvxyyz \notin L$

3) vxy is in b's, so by pumping lemma, we add b's, but not a's, c's or d's, so $uvvxyyz \notin L$

4) vxy is in b's and c's, so by pumping lemma, we add b's & c's, but not a's or d's, so $uvvxyyz \notin L$

5) vxy is in c's, so by pumping lemma, we add c's, but not a's, b's, or d's so $uvvxyyz \notin L$

6) vxy is in c's and d's, so by pumping lemma, we add c's & d's, but not a's or b's, so $uvvxyyz \notin L$

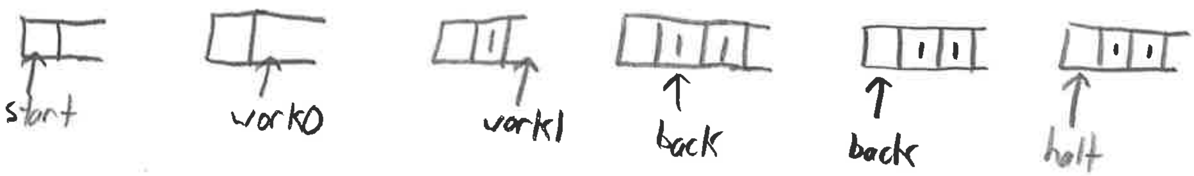
7) vxy is in d's, so by pumping lemma, we add d's, but not a's, b's or c's, so $uvvxyyz \notin L$

In all possible cases, $uvvxyyz$ is not in the language, so there is a contradiction in our assumption that L is CFG, therefore, L is not CFG.

10/10

8. Design a Turing machine that, given a positive unary number n, adds 2 to this number. Test it, step-by-step, on the example of n = 0.

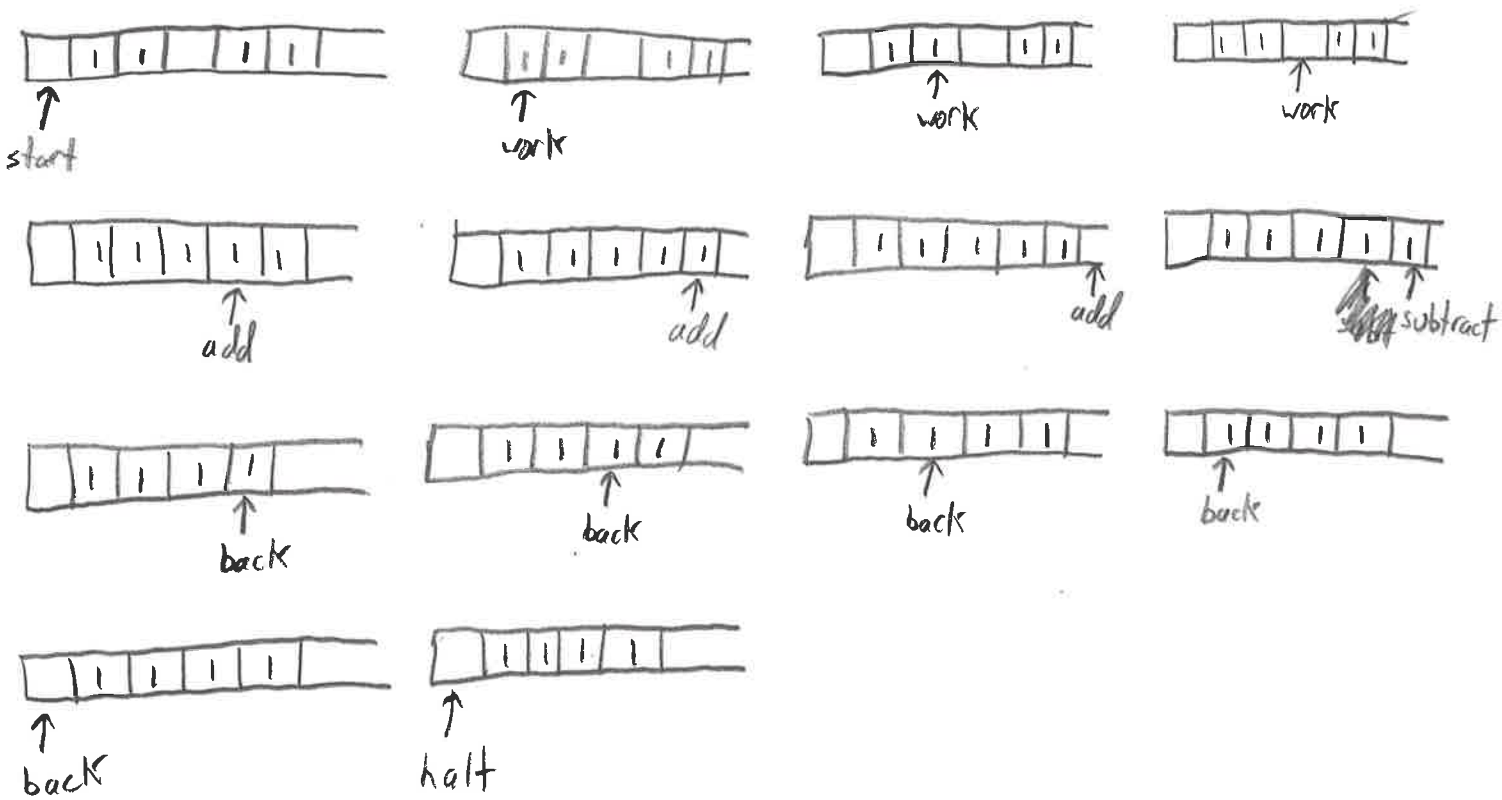
start, ⊔ → R, work0
work0, 1 → R
work0, ⊔ → 1, work1, R
work1, ⊔ → 1, back, L
back, 1 → L
back, ⊔ → halt



10/10

9. (For extra credit) Design a Turing machine that, given two unary numbers, computes their sum. The input is represented as two numbers separated by blank space. Test it, step-by-step, on the example of 2 + 2.

start, ␣ → R, work
work, 1 → R
work, ␣ → 1, add, R
add, 1 → R
add, ␣ → subtract, L
subtract, 1 → ␣, back, L
back, 1 → L
back, ␣ → halt

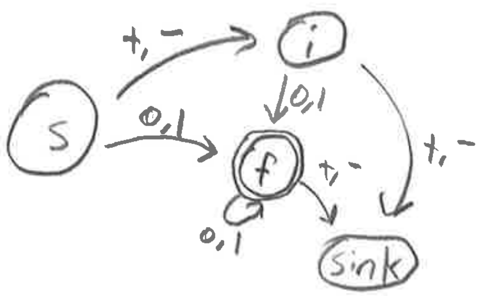


10/10

10. (For extra credit) Let us consider possibly signed binary integers. Such numbers can be described by the following finite automaton. This automaton has a starting state s, an intermediate state i, a final state f, and a sink state k, and the following transitions:

- from s, + or − lead to i, while 0 or 1 lead to f;
- from i, 0 or 1 lead to f, while + or − leads to sink;
- from f, 0 or 1 lead to f, while + or − lead to sink.

Use the general algorithm to transform this finite automaton into a Turing machine. Show, step-by-step, how your Turing machine will accept the word +01.



s_0, ⊔ → s, R
s, + → i, R
s, − → i, R
s, 0 → f, R
s, 1 → f, R

i, 0 → f, R
i, 1 → f, R
i, + → ~~sink~~, reject
i, − → reject

f, 0 → f, R
f, 1 → f, R
f, + → reject
f, − → reject
f, ⊔ → accept