20/20

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

$\cancel{S \to \varepsilon}$
$\cancel{S \to aSb}$
$\cancel{S \to bSa}$

pre: $\cancel{S_0 \to S}$

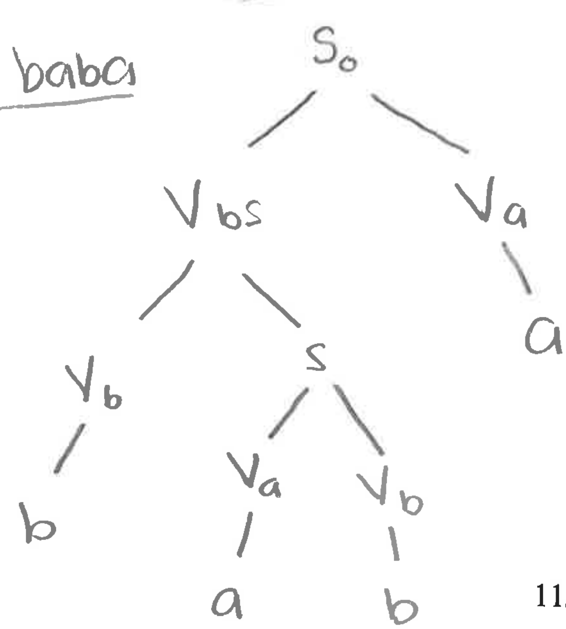
Step 0: $\cancel{S \to ab}$
$\cancel{S \to ba}$
$S_0 \to \varepsilon$

Step 1: $\cancel{S_0 \to aSb}$
$\cancel{S_0 \to bSa}$
$\cancel{S_0 \to ab}$
$\cancel{S_0 \to ba}$

Step 2: $V_a \to a$
$V_b \to b$
$\cancel{S \to V_a S V_b}$
$\cancel{S \to V_b S V_a}$
$S \to V_a V_b$
$S \to V_b V_a$
$S_0 \to \varepsilon$
$\cancel{S_0 \to V_a S V_b}$
$\cancel{S_0 \to V_b S V_a}$
$S_0 \to V_a V_b$
$S_0 \to V_b V_a$

Step 3:
$S \to V_{aS} V_b$, $V_{aS} \to V_a S$
$S \to V_{bS} V_a$, $V_{bS} \to V_b S$
$S_0 \to V_{aS} V_b$
$S_0 \to V_{bS} V_a$
$V_a \to a$
$V_b \to b$
$S \to V_a V_b$
$S \to V_b V_a$
$S_0 \to \varepsilon$
$S_0 \to V_a V_b$
$S_0 \to V_b V_a$

baba

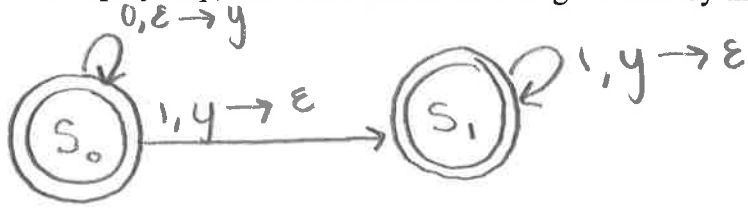


20/20

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 $1, y \to \varepsilon$.

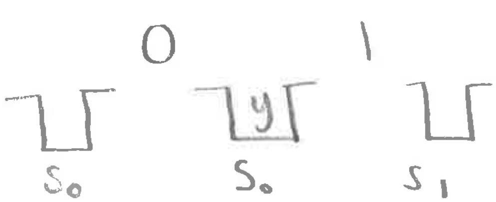
Show, step by step, how the word 01 will be generated by the resulting grammar.



- $S \to A_{s_0 s_1}$
- $A_{s_0 s_0} \to \varepsilon$
- $S \to A_{s_0 s_0}$
- $A_{s_1 s_1} \to \varepsilon$

- $A_{s_0 s_1} \to 0 A_{s_0 s_0} 1$

- $A_{s_0 s_1} \to 0 A_{s_0 s_1} 1$



-9 / -3

10/10

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 − /

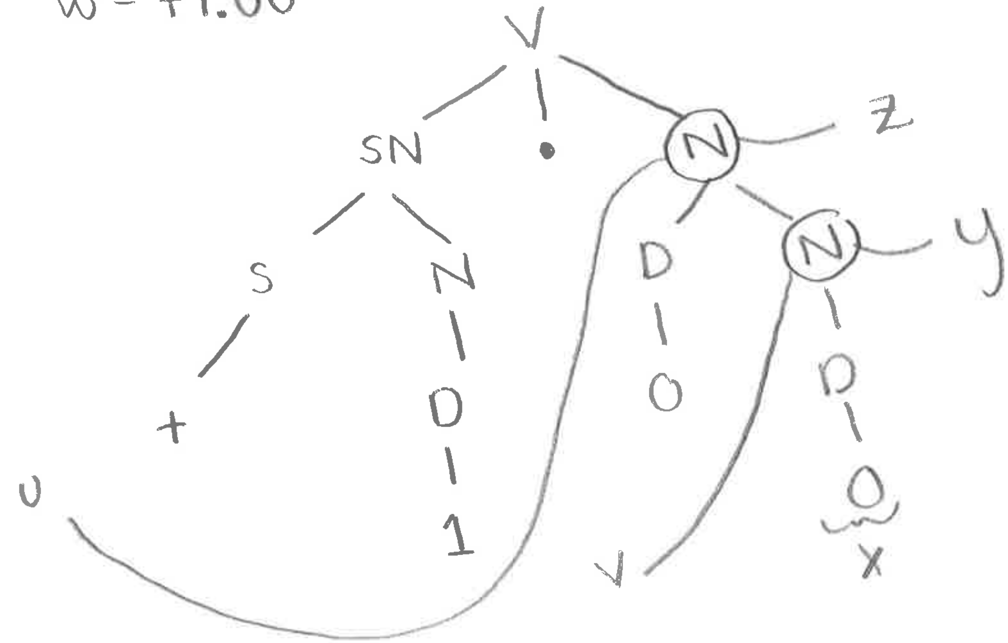
1 | 10 1 | -9 | 8 -9 | 11 8 -9 | -3 -9 | 3

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.

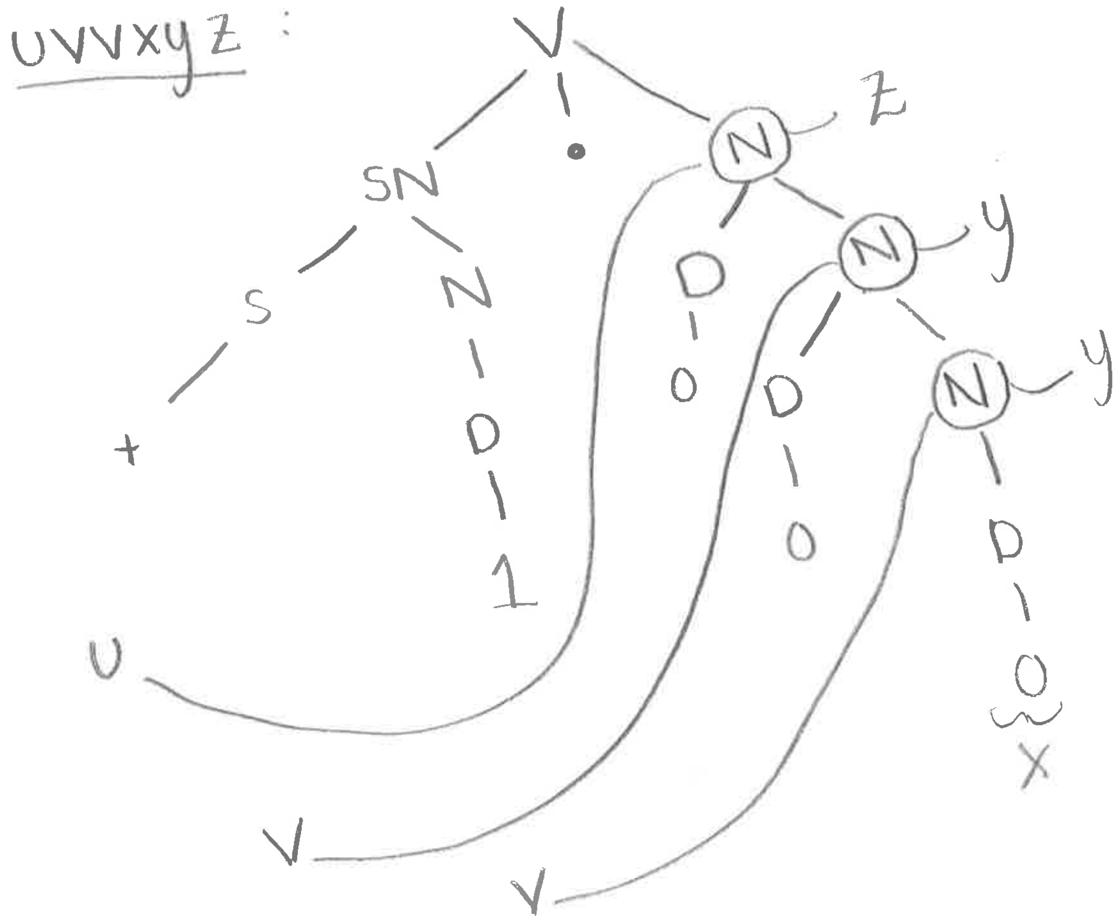
$V \to SN.N$
$V \to N.N$
$S \to +$
$S \to -$
$N \to DN$
$N \to D$
$D \to 0$
$D \to 1$

w = +1.00



uvvxyz :

u = +1.
v = 0
x = 0
y = Λ
z = Λ



 = +1.000

8/10

7. Prove that the language of all the words of the type $a^n b^n c^n d^n$, n = 0, 1, 2, ..., is not context-free.

$L = \{a^n b^n c^n d^n\}$

By contradiction, let's assume L is CFG. Then by pumping Lemma $\exists p \forall w \in L(\text{length}(w) \geq p \rightarrow \exists uvxyz : w = uvxyz$ & $\text{length}(vy) > 0$ & $\text{length}(vxy) \leq p$ & $\forall i(uv^i xy^i z \in L))$ This is i not dash

Let's take $w = a^p b^p c^p d^p$, its length $(w) = 4p \geq p$, so we can represent it as $uvxyz \Rightarrow a, \ldots ab \ldots bc \ldots cd \ldots d$

The fragment vxy cannot contain a's, b's, c's and d's because then length(vxy) is greater than p because all b's are already length p. we have the following options.

1. vxy is in a's - we add a's but not b's, c's or d's: we now have more a's than b's, c's, and d's
2. vxy is in a's and b's - we add a's and b's but not c's. we now have more a's and b's than c's and d's
3. vxy is in b's - we add b's but not a's or c's or d's: we now have more b's than a's, c's and d's
4. vxy is in b's and c's - we add b's and c's but not a's or d's: we now have more b's and c's than a's and d's
5. vxy is in c's - we add c's but not a's, b's or d's - we now have more c's than a's, b's and d's
6. vxy is in c's and d's - we add c's and d's but not a's or b's, we now have more c's and d's than a's or b's
7. vxy is in d's - we add d's but not a's, b's or c's. we now have more d's than a's, b's, c's. So ~~对于~~ $uv^2xy^2z \notin L$ but

This is a contradiction and thus our assumption that L is CFG is false

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.

Rules

start, ⊔ → working, R

working, 1 → working, R

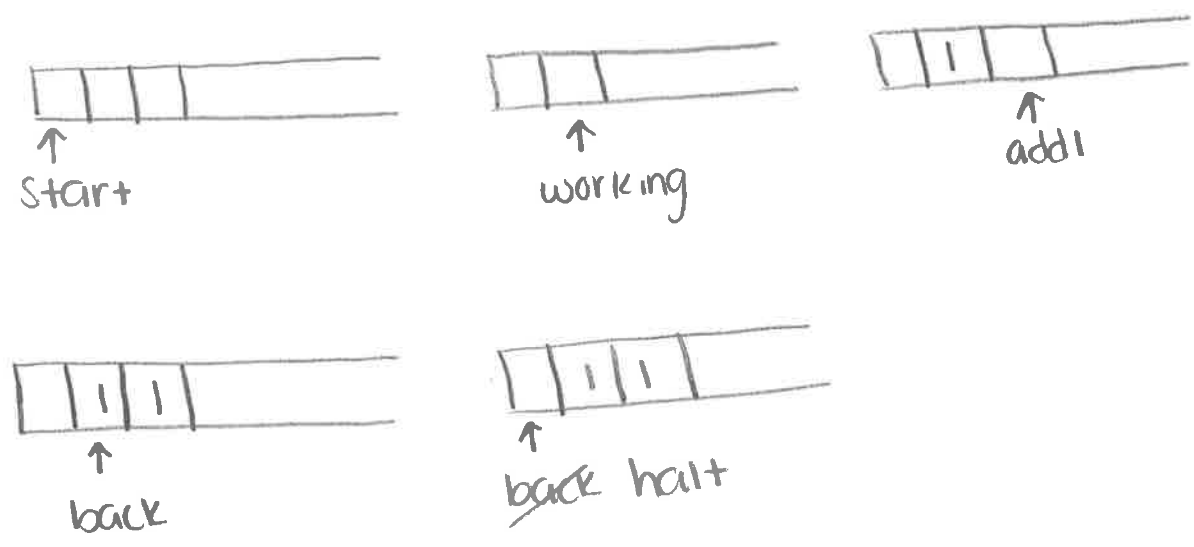
working, ⊔ → add1, 1, R

add1, ⊔ → back, 1, L

back, 1 → back, L

back, ⊔ → halt

n = 0



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.

Rules

Start, $\sqcup \to$ num1, R

num1, 1 $\to$ num1, R

num1, $\sqcup \to$ num2, R

num2, 1 $\to$ num2, R

num2, $\sqcup \to$ erase, L

erase, 1 $\to$ $\sqcup$, add, L

erase, $\sqcup \to$ back, L

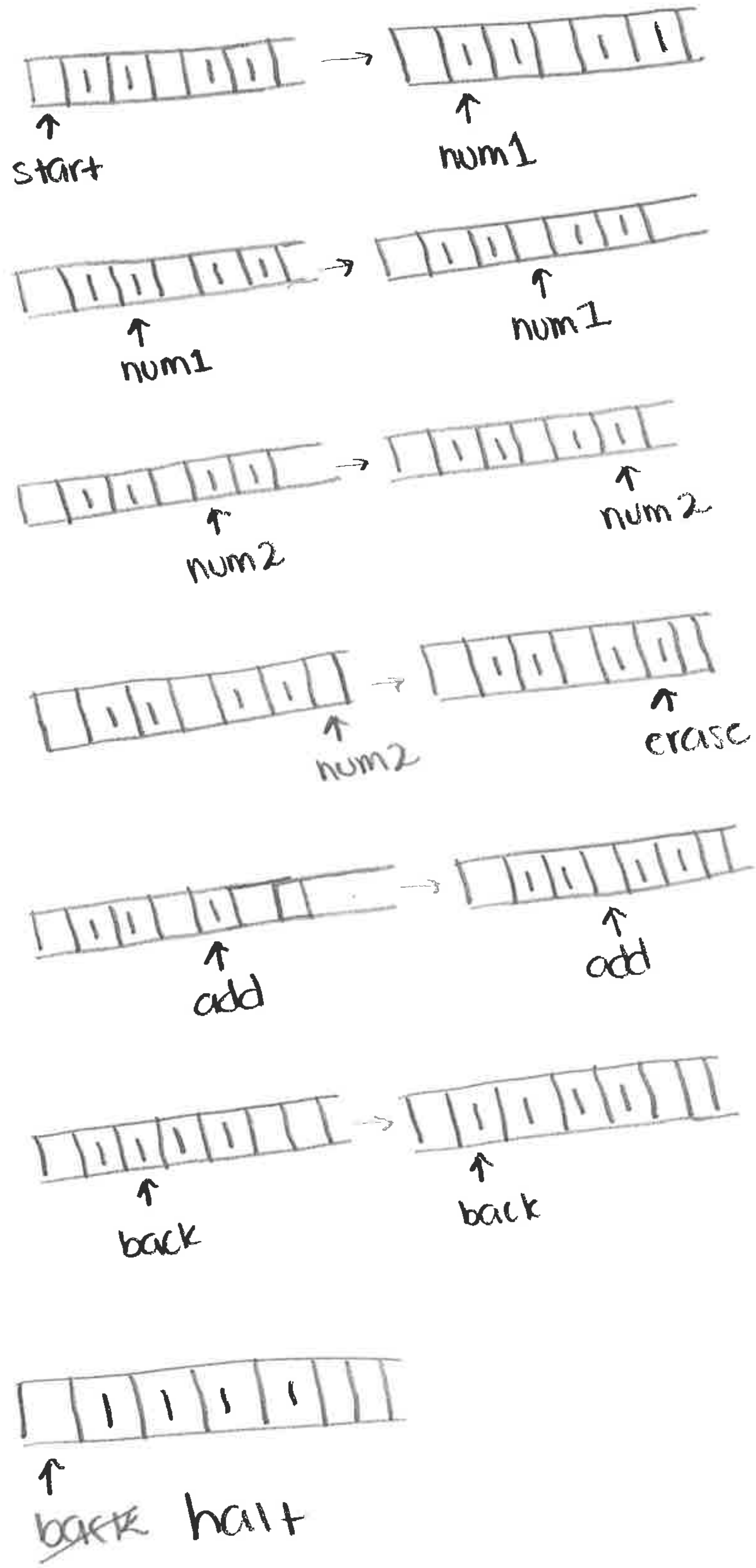
add, 1 $\to$ add, L

add, $\sqcup \to$ 1, back, L

back, 1 $\to$ back, L

back, $\sqcup \to$ halt

2 + 2

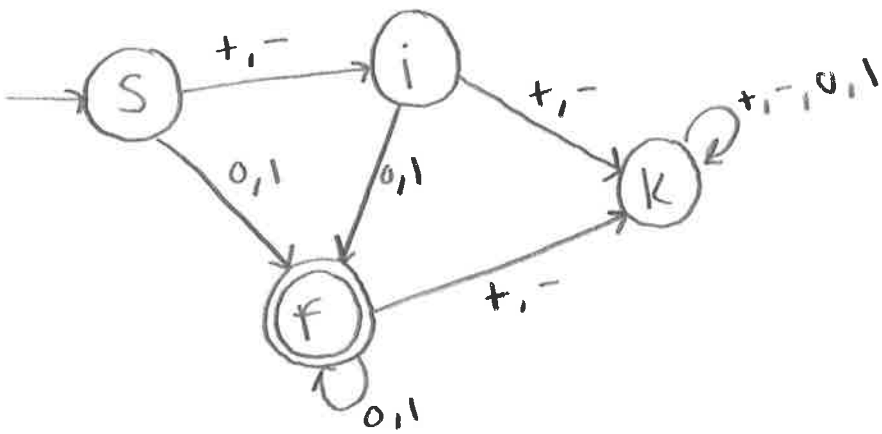


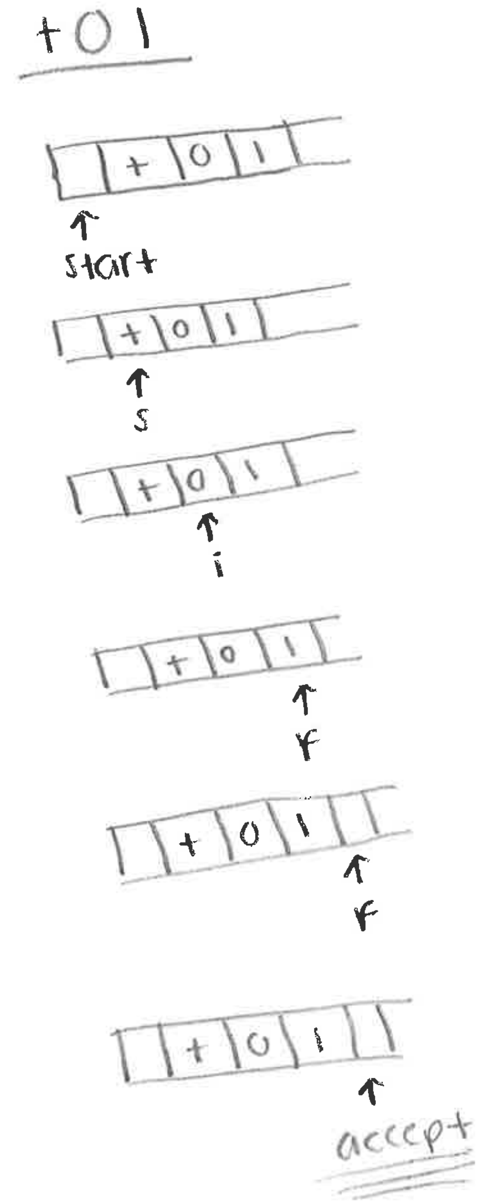
10/10

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.





Start, $\sqcup$ → S, R
S, + → i, R
S, – → i, R
S, 0 → F, R
S, 1 → K, R
i, 0 → F, R
i, 1 → F, R
i, + → K, R
i, – → K, R

K, + → K, R
K, – → K, R
K, 0 → K, R
K, 1 → K, R
F, 0 → F, R
F, 1 → F, R
F, + → K, R
F, – → K, R
S, $\sqcup$ → reject
i, $\sqcup$ → reject
k, $\sqcup$ → reject
F, $\sqcup$ → accept