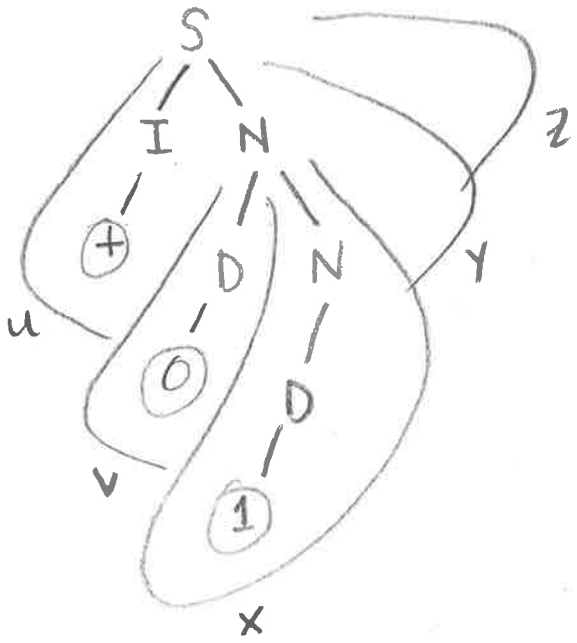
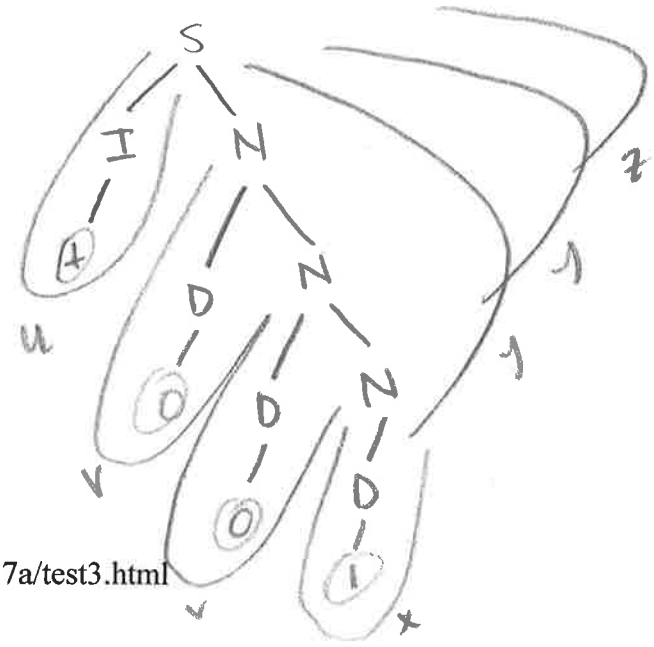
10|10

1. Illustrate the pumping lemma for context-free grammars by showing how it will represent the word w = +01 which is generated by the CFG with starting variable S and rules S → N, S → IN, I → +, I → −, 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.



u - +
v - 0
x - 1
y - ε
z - ε

uvvxyyz = +001



10|10

2. Prove that the language of all the words that have an equal number of a's and b's, and twice as many c's is not context-free.

$L = \{a^n b^n c^{2n}\}$

By contradiction lets assume is CFG. Then by Pumping lemma, there exists P such that every word w from L whose length is $\geq P$ can be represented as $w = uvxyz$, so that $len(vy) \geq P$, $len(vxy) \leq P$ and for all i $uv^i x y^i z \in L$. Lets take $w = a^P b^P c^{2P}$, its length is $4P \geq P$. So we can represent it as uvxyz a..ab..bC....C. The fragment vxy cannot contain a's, b's and c's because then it would have length $> P$ and $len(vxy) \leq P$. So we have the following options:

1) vxy is in a's
2) vxy is in a's and b's
3) vxy is in b's
4) vxy is in b's and c's
5) vxy is in c's

When we consider $uvvxyyz$ in case 1, we add a's but not b's or c's. So in uv^2xy^2z we have more a's than b's or c's. So $uv^2xy^2z \notin L$

In case 2 we are adding a's and b's but not c's. So in uv^2xy^2z we have more a's and b's than c's. So $uv^2xy^2z \notin L$

In case 3 we are adding b's but not a's or c's. So in uv^2xy^2z we have more b's than a's or c's. So $uv^2xy^2z \notin L$

In case 4 we are adding b's and c's but not a's. So in uv^2xy^2z we have more b's and c's than a's. So $uv^2xy^2z \notin L$

In cases we are adding c's but not a's or b's. So in uv^2xy^2z we have more c's than a's or b's. So $uv^2xy^2z \notin L$.

but by pumping lemma $uv^2xy^2z \in L$. So, we have a contradiction. so our assumption is is false and L is not a CFG

10/10

3. 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 = 1.

(start, #) → (inNum, R)

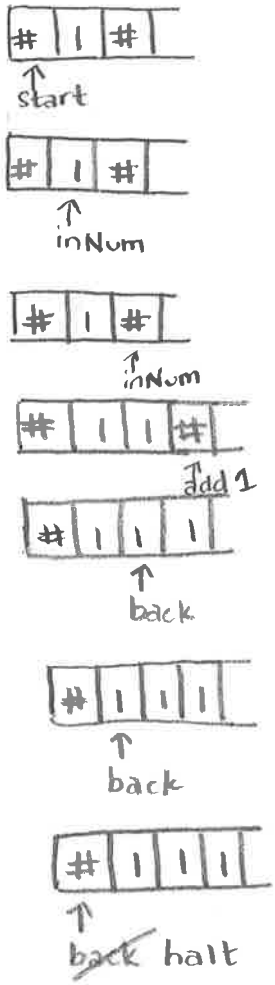
(inNum, 1) → (inNum, R)

(inNum, #) → (add1, 1, R)

(add1 #) → (back, 1, L)

(back, 1) → (back, L)

(back, #) → halt



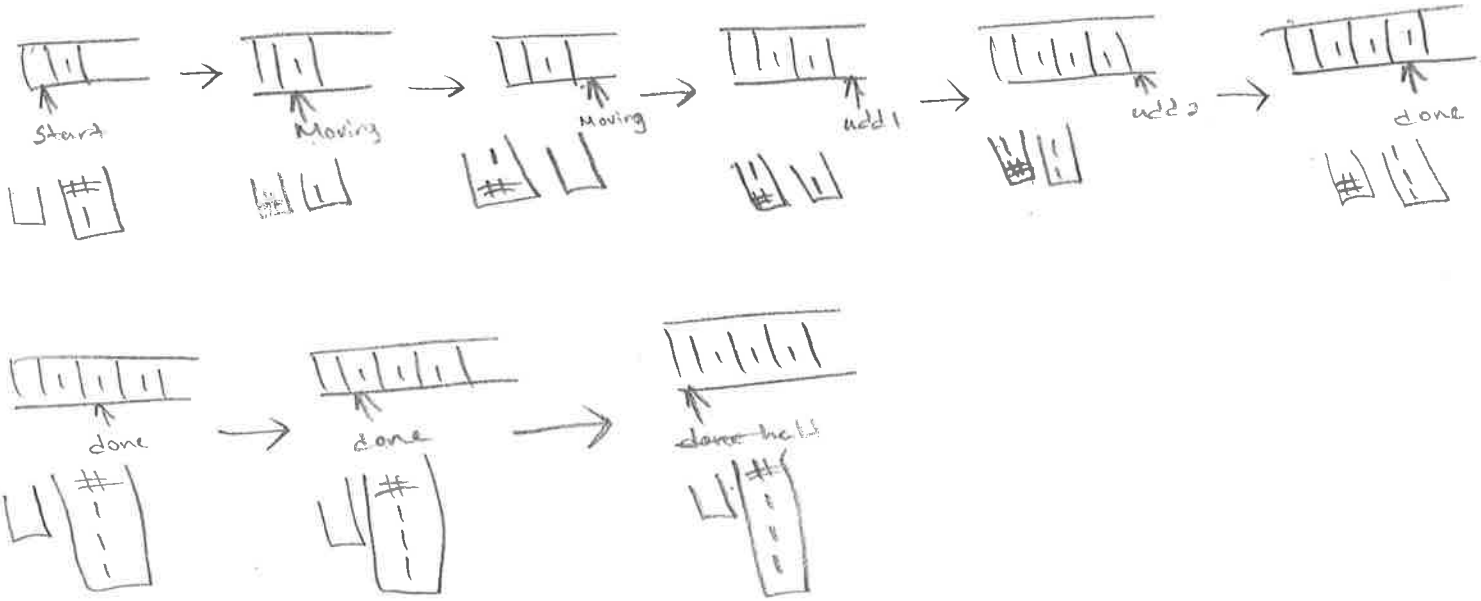
10/10

4. As we discussed in class, a Turing machine can be described as a finite automata with two stacks:

   - the right stack contains, on top, the symbol to which the head points; below is the next symbol to the right, then the next to next symbol to the right, etc.;
   - the left stack contains, on top, the symbol directly to the left of the head (if there is a one), under it is the next symbol to the left, etc.

   On the example a Turing machine from Problem 3, show, step-by-step:

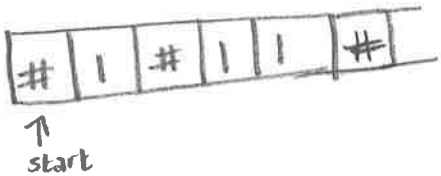
   - how each state of the corresponding Turing machine can be represented in terms of two stacks, and
   - how each transition from one state to another can be implemented by push and pop operations.

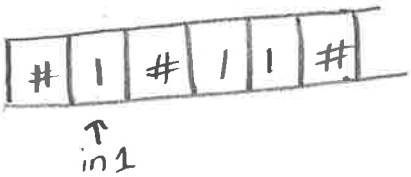


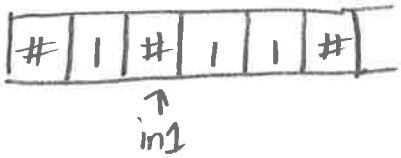
10/10

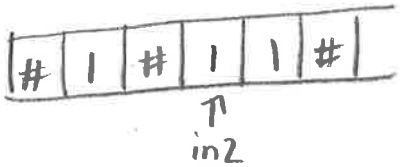
5. Design a Turing machine that, given two unary numbers, computes their sum. Test it, step-by-step, on the example of 1 + 2.

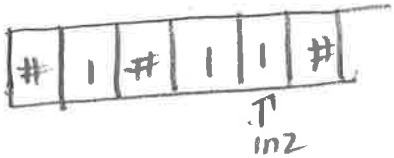
(start, #) → (in1, R)
(in1, 1) → R
(in1, #) → (in2, R)
(in2, 1) → R
(in2, #) → Ready, L
(ready, 1) → (#, L, wait)
(wait, 1) → L
(wait, #) → 1, L, back
(back, 1) → L
(back, #) → halt
(ready, #) → L, back

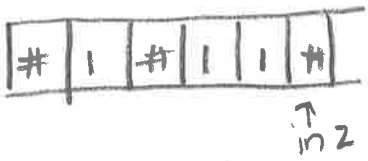


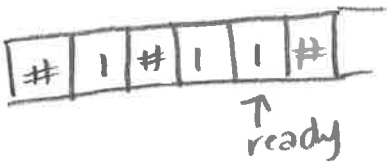


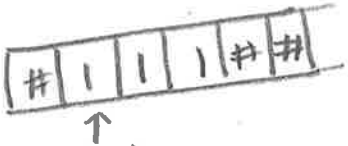


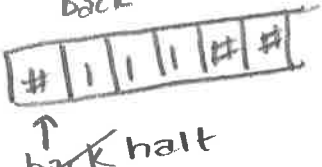












10/10

6. Give two examples:

- an example of an computation time which makes an algorithm feasible according to the formal definition but not practically feasible, and
- an example of a computation time for which the corresponding algorithm is practically feasible, but not feasible according to the formal definition.

These examples should be different from what you learned in class -- a minor difference is OK.

Practically feasible but not theoretically

- $2^{(10^{-11}n)}$

Theoretically feasible but not practically

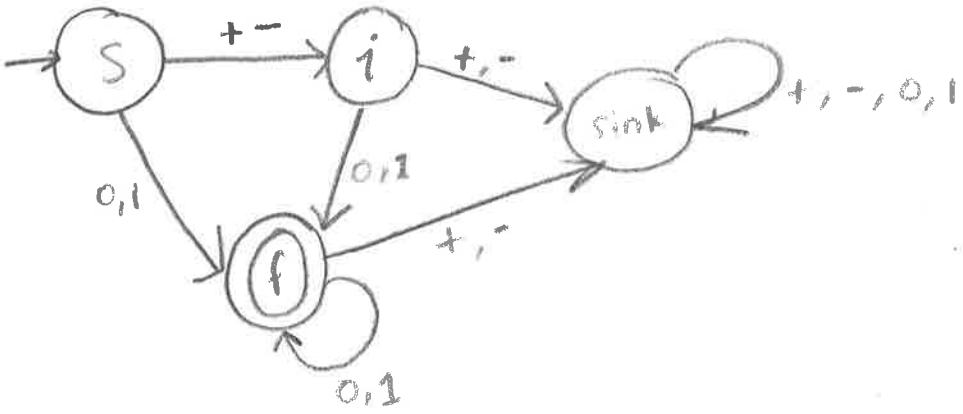
- $10^{110}n$

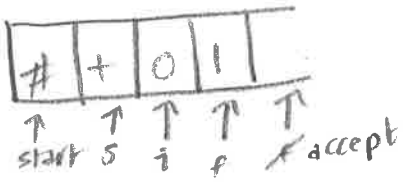
10/10

7. Use the general algorithm to transform the following finite automaton for recognizing possibly signed binary integers to a Turing machine. 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 − lead to sink;
- from f, 0 or 1 lead to f, while + or − lead to sink.

Show, step-by-step, how your Turing machine will accept the word +01.





(Start, #) → (s, R)

(s, +/−) → (i, R)

(s, 0/1) → (f, R)

(i, +/−) → (sink, R)

(i, 0/1) → (f, R)

(f, 0/1) → (f, R)

(f, +/−) → (sink, R)

(sink, +/−/0/1) → (sink, R)

(sink, #) → reject

(s, #) → reject

(i, #) → reject

(f, #) → accept

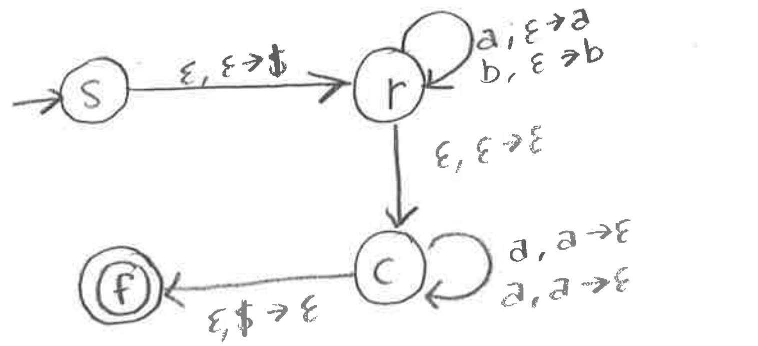
10/10

8. Use the general algorithm to transform the following PDA into a two-tape Turing machine. This PDA recognizes words of the type ww^R, where w is any word, and w^R means the same word reversed.

For example, if w = ab, then w^R = ba, and ww^R = abba.

The pushdown automaton has 4 states:

- the starting state s,
- the reading state r,
- the checking state c, and
- the final state f.



The transitions are as follows:

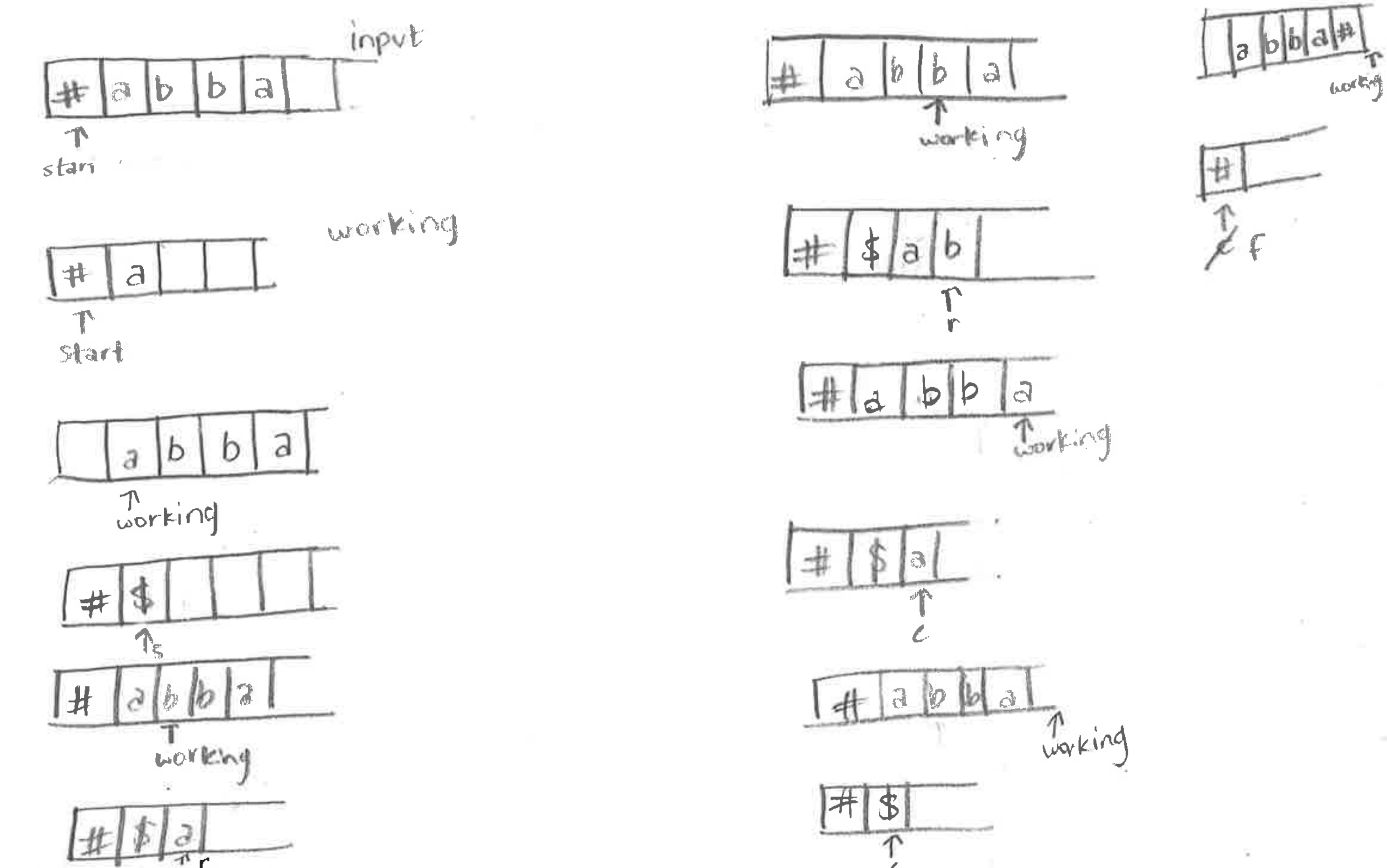
- From s to r, the transition is:
  - ε, ε → $;
- From r to r, the transitions are:
  - a, ε → a;
  - b, ε → b;
- From r to c, we have a jump ε, ε → ε.
- From c to c, the transitions are:
  - a, a → ε
  - b, b → ε
- From c to f, the only transition is:
  - ε, $ → ε.

(start, start2, #, #2) → (working, R1, R2, $2, S)

(working, S, a1) → (R1, R2, a2)

(working, S, b1) → (R1, R2, b2)

Show, step-by-step, how the resulting Turing machine will recognize the sequence *abba*.



10/10

9. What is NP? What is P? What is NP-complete? What is NP-hard? Give brief definitions. Give an example of an NP-complete problem. Is P equal to NP?

P - class of languages decidable in polynomial time on a deterministic single tape

NP - Class of languages that have polynomial time verifiers

NP-hard A problem x is NP-hard if there is an NP-complete problem y such that y is reducible to x in polynomial time

NP-complete - A language B is NP complete if it satisfies 2 conditions:

1) B is in NP

2) Every A in NP is polynomial time reducible to B

Example: Hamiltonian path - whether input graph contains a path s to t that goes through every node exactly once.

It is not known whether P=NP, it is still an unsolved question.

10. Formulate Church-Turing thesis. Is it a mathematical theorem? Is it a statement about the physical world?

Anything that can be computed on any physical device can also be computed on a turing machine. It is NOT a mathematical theorem, it IS a statement about the physical world