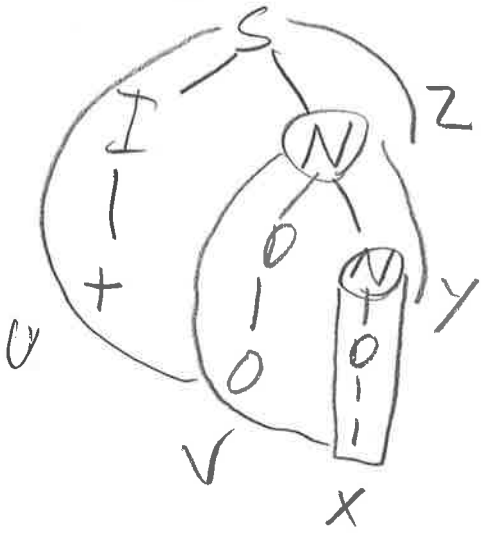
10|10

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.



$w = +01$

$u = +$

$v = 0$

$x = 1$

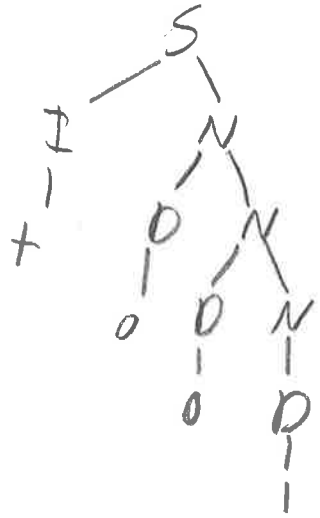
$y = \Lambda$

$z = \Lambda$

$len(vy) = len(0) = 1 > 0$ ✓

$len(vxy) = len(01\Lambda) = 2 \leq p$ (by assumption)

$uv^2xy^2z = +001\Lambda\Lambda\Lambda = +001$ which is generated as below



So $uv^2xy^2z \in L$

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.

The language $L=\{a^n b^n c^{2n}$ is not CFG.

Proof by contradiction:

Let's assume that L is a CFG. Then, by pumping lemma, there exists p such that every word from L whose length is $\geq p$ can be represented as $w=uvxyz$ so that $L(vy)>\emptyset$, $len(vxy)\leq p$, and for all i, $uv^ixy^iz \in L$. Let's take $w=a^p b^p c^{2p}$. Its length is $4p \geq p$, so we can represent it as $uvxyz$. a...b...bc...c...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 one of the following options:

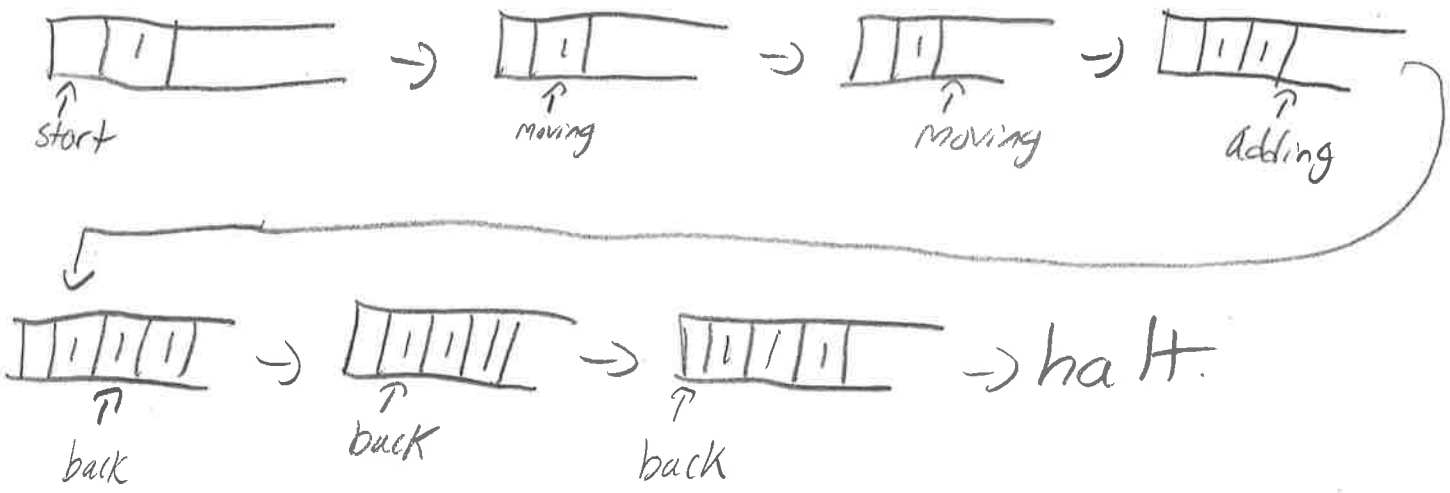
1) vxy is in a's
2) vxy is in a's and b's
3) vxy is in b's
4) vxy is in b's and c's
5) vxy is in c's

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 add 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 add b's but not a's and 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 add 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 case 5, we add more c's but not a's or b's, so we have more than the twice amount of c's than a's and b's, so $uv^2xy^2z \notin L$. But by pumping lemma $uv^2xy^2z \in L$. We have a contradiction, so our assumption is false, 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, #) -> (moving, R)
(moving, 1) -> (moving, R)
(moving, #) -> (adding, 1, R)
(adding, #) -> (back, 1, L)
(back, 1) -> (back, L)
(back, #) -> halt



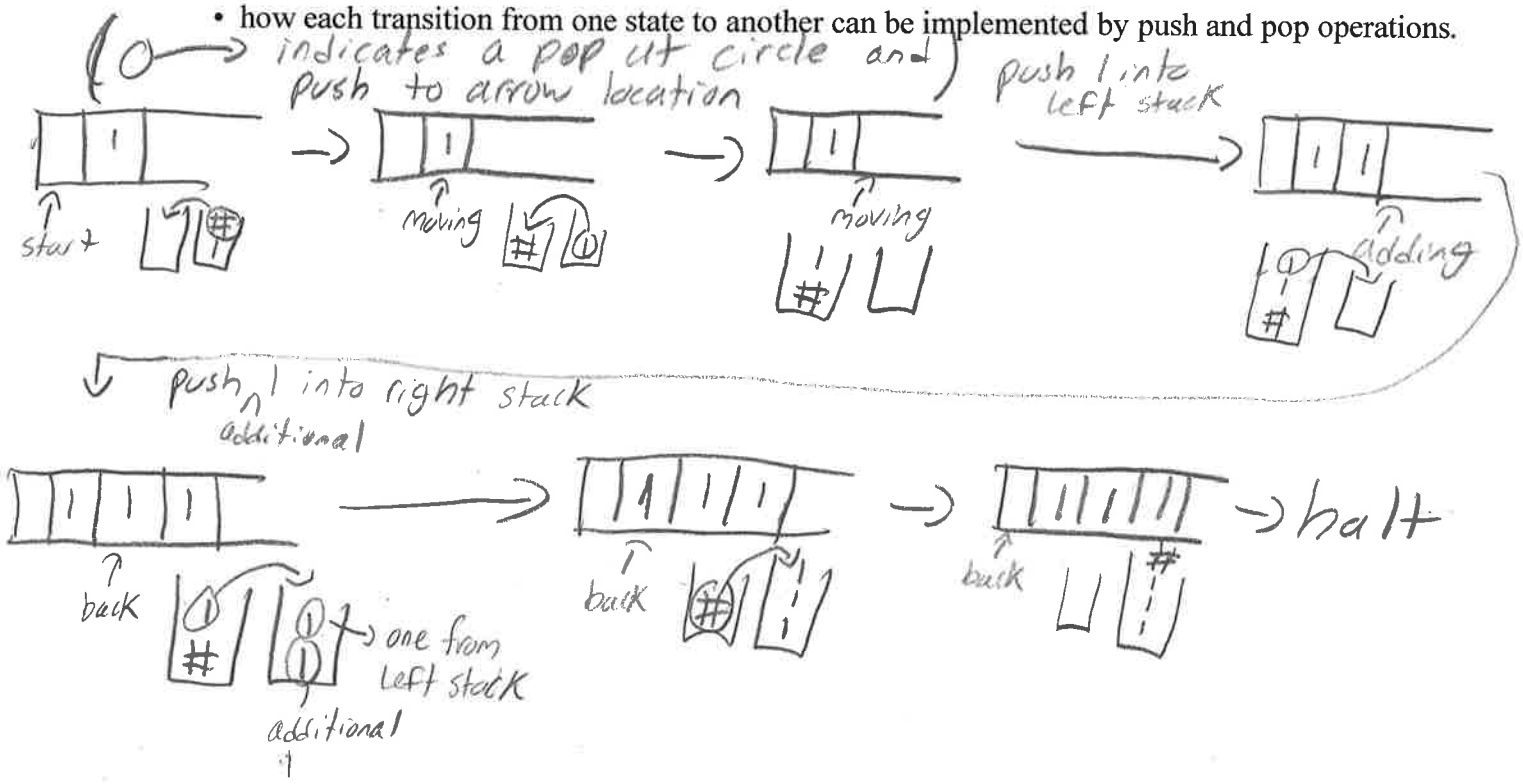
10/10

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.



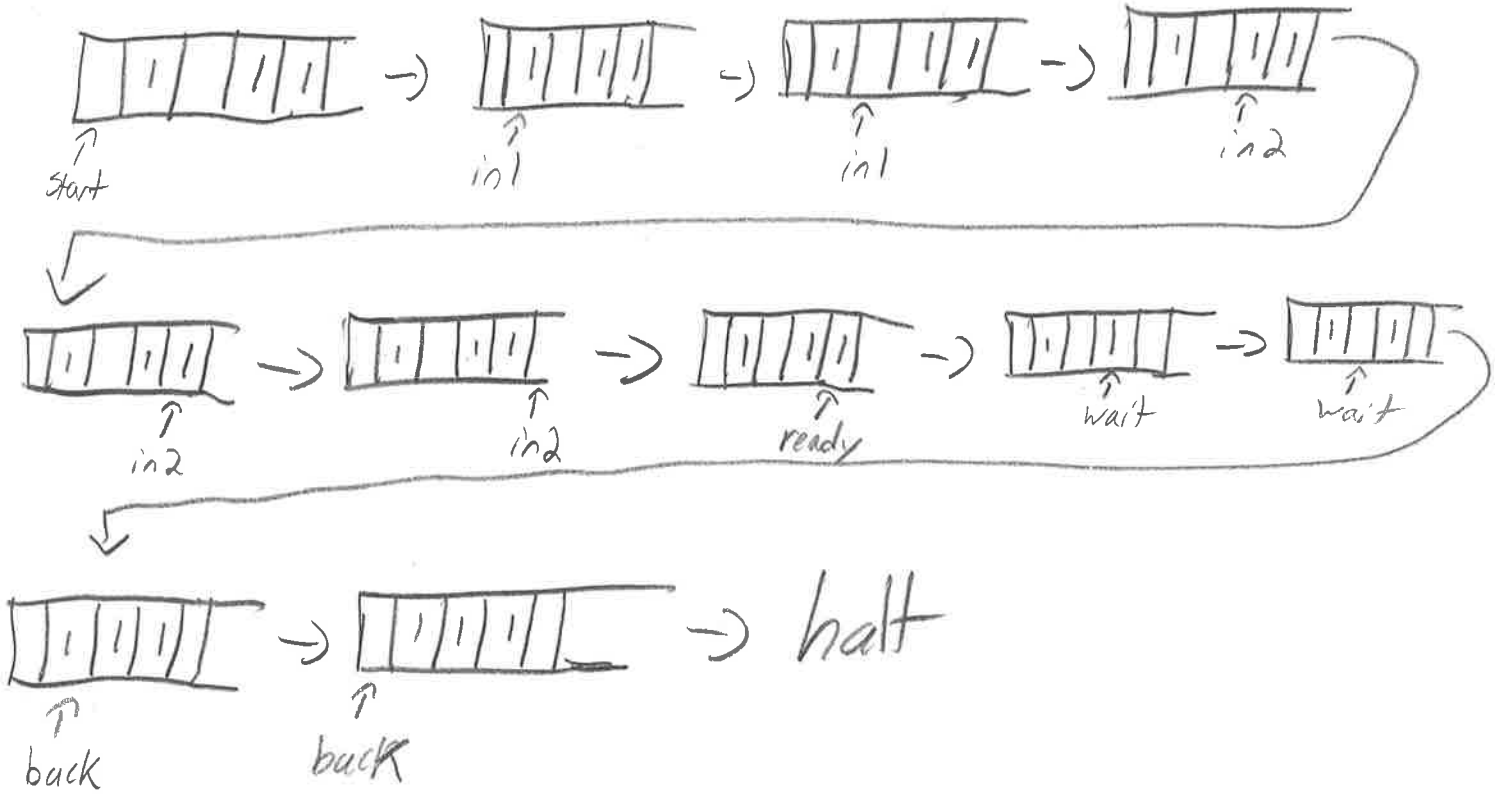
10|10

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) → (in1, R)
(in1, #) → (in2, R)
(in2, 1) → (in2, R)
(in2, #) → (ready, L)

(ready, 1) → (wait, #, L)
(ready, #) → (back, L)
(wait, 1) → (back, L)
(wait, #) → (back, 1, L)
(back, 1) → (back, L)

(back, #) → halt



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.

feasible but not practically feasible;

$10^{2^{10}} \cdot n$

practically feasible but not feasible;

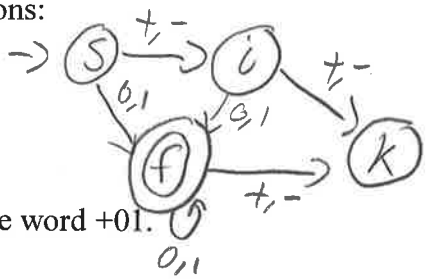
$2^{(10-2^{10})n}$ good

10^{-1000}

10/10

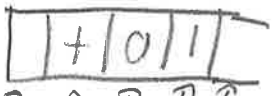
7. Use the general algorithm to transform the following finite automaton for recognizing possibly signed binary integers to a Turing machine. This automaton has a starting state s, an intermediate state i, a final state f, and a sink state k, and the following transitions:

- 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.

rules: (start, #) → (s, R)
(s, ±) → (i, R)
(s, 0/1) → (f, R)
(i, 0/1) → (f, R)
(i, ±) → (k, R)
(f, 0/1) → (f, R)
(f, #) → accept
(k, ±/0/1) → (k, R)
(i, #) → reject
(k, #) → reject
(s, #) → reject
(f, ±) → (k, R)



start s i f f → accept

so +01 is accepted.

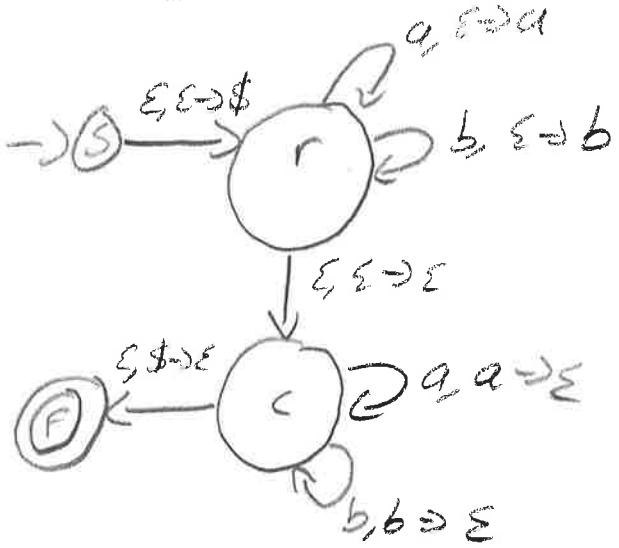
10|10

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:

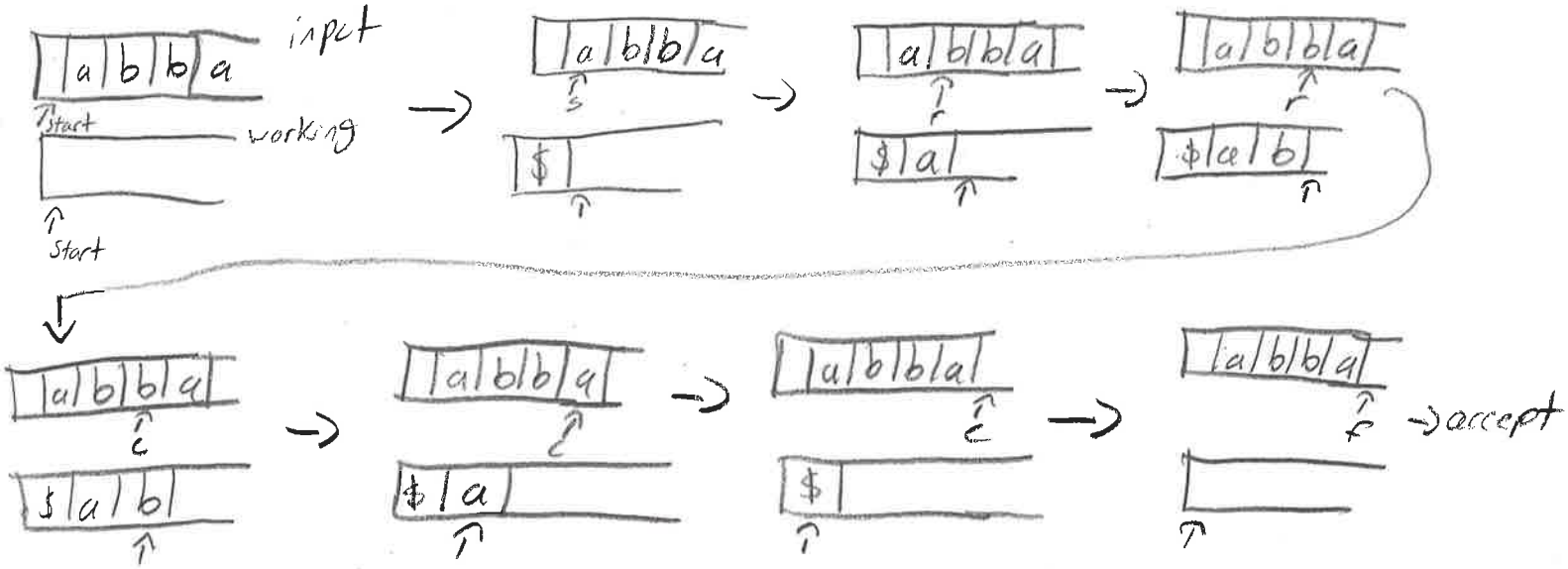
- the starting state s,
- the reading state r,
- the checking state c, and
- the final state f.



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.

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 problems that can be solved in polynomial time.

NP: class of problems for which, once you have a candidate for a solution you can feasibly check whether it is a solution

NP-hard: Every problem from class NP can be reduced to this problem. That is, problems harder than those of NP.

ex. SAT

NP-complete: A problem is NP complete iff A is in NP and for all NP problems B, B is polynomial-time may-one reducible to A.

ex. subset sum

$P \stackrel{?}{=} NP$: P equal to NP has not been proven to be true or proven to be false.

10/10

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

- Thesis: Anything that can be feasibly computed on any physical device can also be feasibly computed on a Turing Machine.

- It is not a mathematical theorem, it is a statement about the physical world.