Problem 1

a) TRUE.

$L(D_1) = L(D_2) \Rightarrow \overline{L(D_1)} \cap L(D_2) = \phi$ (can be computed in linear time)

and $L(D_1) \cap \overline{L(D_2)} = \phi$

- $L(D_1) \cap \overline{L(D_2)} = \phi$: can be computed in quadratic time
- $\overline{L(D_1)} \cap L(D_2) = \phi$: checking emptiness can be done via breath-first-search.

b) False. ϕ and Σ^* $\notin$ NL-complete

but ϕ and $\Sigma^* = L \subseteq NL$.

c) True. Proof by contradiction.

For each B we need a different computable function f_B for the reduction.

there are uncountably many B but only countably many computable functions.

d) False.

Let $A_2 = \{ w \in \{0,1\}^* \mid w \text{ starts with } 1 \}$

$A_1 = \{ w \in \{0,1\}^* \mid |w| \text{ is even} \}$

clearly $A_1, A_2 \in L$ so $A_1 \leq_L A_2$.

But $A_2 \in \text{TIME}(1)$ while $A_1 \in \text{TIME}(n^2)$ $\notin \text{TIME}(o(n))$.

It is not enough to say that $A_1 \leq_L A_2$ implies that $A_1 \in \text{TIME}(t(n) \cdot n^{O(1)})$ because it could exist a direct algorith for A_1 that is faster than using the reduction. You need to give a counter-example!

Problem 1 (cont)

e) TRUE. Call the language A.

$\overline{A} \in NL$.

For each final state $q \in F$ { (initial state)
  if there exists path (in the DFA) from s to q then accept ⇐ = PATH.
}
reject

So $\overline{A} \leq_L$ PATH ⇒ $A \leq_L \overline{\text{PATH}}$ ⇒ $A \in$ co-NL = NL.

Now we show that $A \in$ NL-complete via $\overline{\text{PATH}} \leq_L A$

Take the graph to be the DFA. Take s to be the initial state and t to be the only final state. Compute the maximum degree d of the graph. Since $d \leq$ number of nodes -1 we can store d (and compute it) in log space.
Let $\Sigma = \{1, 2, \ldots, d\}$ and label the edges from a node v in order from 1 to the degree of v. If degree$(v) \leq d-1$ then add transitions from v to a new dead state with symbols degree$(v)+1$, degree$(v)+2$, ..., d.
this can also be done in log-space.

Problem 2:

SUBSET-SUM

ELEMENTS $\{z_1, z_2 \ldots z_n\}$

FIND SUBSET THAT SUMS TO Y.

$$A = \begin{bmatrix} z_1 & z_2 & \ldots & z_n \\ -z_1 & -z_2 & \ldots & -z_n \end{bmatrix}$$

0-1 INT. PROGRAMMING

$$b = [Y \quad -Y]$$

Problem 3

a) Let $w_1, w_2, w_3 \ldots$ be an ordering of all strings in Σ^*.

for each $i = 1, 2, \ldots$

run M_1 and M_2 for each $w_1, w_2, \ldots w_i$ up to i steps.

If M_1 and M_2 simultaneously accept at least k such strings => accept.

b) Reduce

$\overline{A_{TM}} \leq_M \overline{A}$ which implies that $\overline{A}$ is not Turing-recognizable and A is not Turing-decidable.

On input $\langle M, w \rangle$ to $\overline{A_{TM}}$

construct machines M_1 and M_2 as follows

M_1 accept all languages in Σ^*.

M_2, on input x;
accepts iff M accepts w.

So, if M accepts w => $L(M_2) = \Sigma^*$ and $|L(M_1) \cap L(M_2)| \geq k$
$\langle M, w \rangle \notin \overline{A_{TM}}$ and $\langle M_1, M_2, k \rangle \notin \overline{A}$

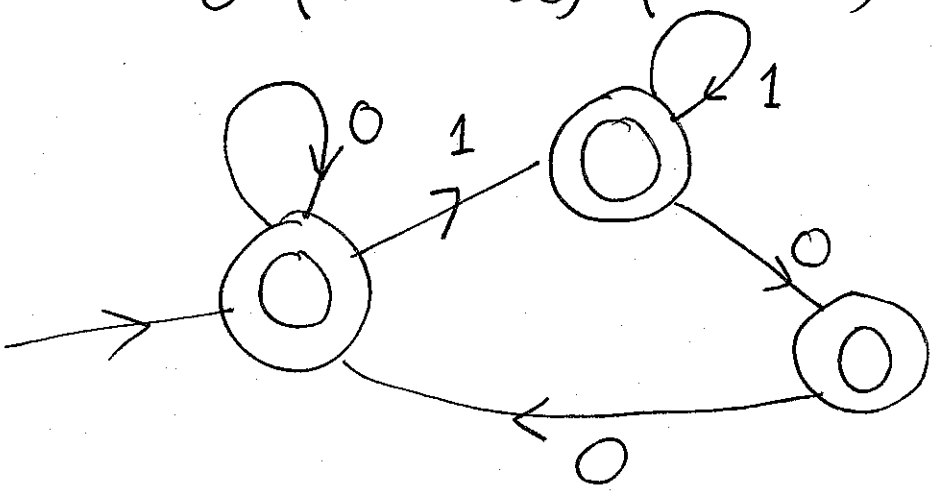
if M does not accept w => $L(M_2) = \phi$ => $|L(M_1) \cap L(M_2)| = 0$
$\langle M, w \rangle \in \overline{A_{TM}}$ and $\langle M_1, M_2, k \rangle \in \overline{A}$

Problem 4:

Take a string like that and split it into pieces with form 10^* plus a prefix of 0's.

Each piece 10^* must not be 10 with the exception of the last piece.

Answer: $0^*(1 \cup 1000^*)^*(10 \cup \varepsilon)$



The DFA above is the minimum DFA for this language. We took each solution and used a program to generate the minimum DFA and compared to the DFA above. You got credit if the minimum DFA for your regular expression matched the DFA above.

Problem 5:

a) Consider pairs of states (q, q') where q is state of D_1 and q' " " " D_2'.

Put edge from (q, q') to (p, p') if there is a transition from q to p in D_1 and from q' to p' in D_2'. If $L(D_1) \neq L(D_2')$ then there exists a path from (q_0, q_0') to a state (q, q') where either q or q' (but not both) is a final state.

(q_0: initial state of D_1; q_0': initial state of D_2')

Clearly, such a path must have length $\leq m_1 m_2'$ which gives a string in $\left(L(D_1) \cup L(D_2')\right) \setminus \left(L(D_1) \cap L(D_2')\right)$.

b) We show that the language is in NPSPACE = PSPACE.

Note that we can simulate an NFA in PSPACE since it suffices to keep track of the states the NFA is in

We cannot convert it to a DFA since this may use exponential space.

Also, we cannot initially guess the string that distinguishes N_1 and N_2, since the string may be exponentially large. But we can guess the string one symbol at a time. By Lemma (a), string has at most $2^{m_1} 2^{m_2} = 2^{m_1 + m_2}$ symbols, which can be stored in poly space.

For each $i = 1, 2, \ldots 2^{m_1+m_2}$ {
  guess i-th symbol and update current states of N_1 and N_2.
  If either N_1 or N_2 but not both is in an accepting configuration => accept.
}
reject.

(m_1 = # states of N_1; m_2 = # " " N_2).

Problem 6:

$A \in NL$.

Linear grammars have only one non-terminal at the right-hand-side of each production. this means that, during the derivation, at any time, there exists only one variable to expand. We then guess the derivation of that variable at each step.

Let $w = w_1 w_2 \cdots w_n$. We keep two pointers start and end which marks the # of symbols matched for the beginning and ending of w.

If the variable to be derived is X and we guess the rule $X = aYb$ where $a, b \in \Sigma^*$ we do

if $a = w_{start+1} w_{start+2} \cdots w_{start+|a|}$

and $b = w_{n-end-|b|+1} \cdots w_{n-end}$ then

start := start + |a|

end := end + |b|

derive Y

otherwise reject (wrong guess).

the pointers start and end, and the pointer to the variable to be derived require only log-space.

Problem 6 cont'd :

$A \in NL$-complete reduce from PATH.

Let $\langle G, s, t \rangle$ be an instance for PATH.

Each node of G is a variable of the grammar.

s is the start variable.

For each edge $(U \to v)$ add the rule $U \to V$ to the grammar and add the rule $t \to w$.

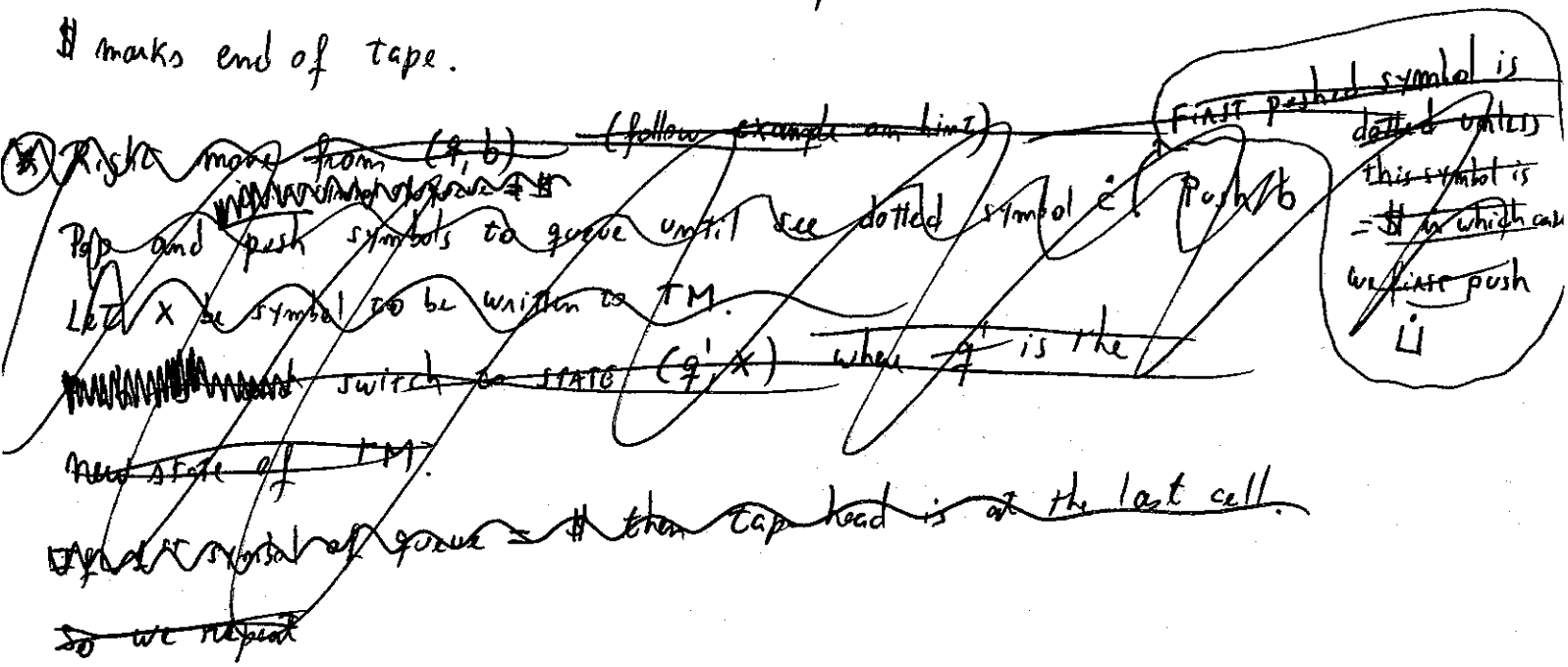
Then, a path from s to t can be used to derive w and, if there exists no path from s to t, the grammar accepts no language.

Problem 7:

the element at the tape head is always marked with a dot in the queue

the element to the left of the tape head is recorded in the state as a buffer (element b is the hint). If tape head is the left most symbol then buffer = \$.

\$ marks end of tape.



(*) Left move from (q, b)

If $b = \$$ we are in left most entry of cell.

If $\neq \$$: Pop and push symbols to the queue remembering last symbol popped until dotted symbol $\dot{c}$ is seen. Let y be remembered symbol.
Push $\dot{b}$ and X, where X is the symbol to be written to tape head.
switch to (q', Y), where q' is new state of TM.

If $b = \$$ we do the same but $Y = \$$ and we do not push $\dot{b}$.

(*) Right move from (q, b)

Pop and push until see dotted symbol, let x be symbol to be written to tape.
Push b. Pop next symbol. If symbol = \$, push $\dot{\sqcup}$ and \$.
otherwise, push $\dot{y}$ (symbol = y)

switch to (q', X)