# 1

## A

Prove if $L_1 \in P$ then $\bar{L_1} \in P$

### Proof

If $L_1 \in P$ then $\exists k$ s.t. $L \in TIME(n^k)$, that is, $L_1$ is decided by a deterministic machine in polynomial time.

Since $L_1 \in TIME(n^k)$, then the max of $t_m(n)$ of $x \in L_1$ and $t_m(n)$ of $x \notin L_1$ is $n^k$. This means a TM $M$ decides $L_1$ in polynomial time.

We can create a 3 tape machine $M'$ that flips the output of $M$ on $x$ by running $M$ on tape 2 and keeping track of the state on tape 3. Once $M$ finishes $M'$ accepts if

As discussed, running $M$ will be $O(n^k)$ time. Reading the final state of $M$ would be $O(n)$ time (where $n$ is length of tape 3). So final runtime is $O(n^k)$, thus $M' \in t_m(n^k)$.

We know $M'$ decides $\bar{L_1}$ because for $x \in \bar{L_1}$, $M$ rejects so $M'$ accepts and for $x \notin \bar{L_1}$, $M$ accepts so $M'$ rejects.

Thus since $M'$ decides $\bar{L_1}$ and $M' \in t_m(n^k)$ then $\bar{L_1} \in TIME(n^k)$ so $\bar{L_1} \in P$

## B

Explain why the techinque you used in part a fails to prove $L_1 \in NP \rightarrow \bar{L_1} \in NP$

The proof would need to prove there exists a verifier for $\bar{L_1}$ which is not present in part A. The techinque used in part A determined the runtime of our machine whereas this proof for $NP$ would need to show there is a verifier $V \in P$ for $\bar{L_1}$

## C

Prove $(L_1 \in P \wedge L_2 \in P) \rightarrow L_1 \circ L_2 \in P$

## Idea

If $L_1 \in P \wedge L_2 \in P$ then $\exists M_1, M_2$ that run in polynomial time that decide $L_1, L_2$.

Lets create a TM $M_3$ to decide $L_1 \circ L_2$ and prove that it decides the concatenation in polynomial time.

$M_3$ will be a 4 tape machine. Tape 1 contains the input string. Tape 2 is working space for $M_1/M_2$ and Tape 3 will contain the state of the current working machine. Tape 4 contains the length of the working substring (by storing the length as a number of 0's).

$M_3$ will count the number of 0's o tape 4 and copy that many characters from the start of Tape 1 onto Tape 2. We then simulate $M_1$ on this string, storing the state on Tape 3. If $M_1$ accepts, then we clear Tapes 2 and 3 then count the number of 0's on Tape 4 and skip that many characters on Tape 1 then copy remaining string to Tape 2. Then simulate $M_2$ on Tape 2 and store the state on Tape 3. If $M_2$ accepts then $M_3$ will accept. Otherwiseif $M_1$ or $M_2$ reject then we add a 0 to Tape 4 and repeat the above steps. If at any point the number of 0's exceed the length of the input string then $M_3$ rejects.

## Proof $M_3$ decides $L_1 \circ L_2$

If $w \in L_1 \circ L_2$ then $\exists x, y | w = xy \wedge x \in L_1 \wedge y \in L_2$.

> $M_3$ iterates through $w$ running $M_1$ on every possible $x$ in $w$. If $M_1$ accepts $x$, then $M_3$ runs $M_2$ on the remaining string ($y$) and will accept if $M_2$ accepts. Since we know there is an $x, y$ pair that are in their respective languages, we know $M_3$ will accept on this pair because it tests every possible $x, y$ combination.

If $w \notin L_1 \circ L_2$ then $\nexists x, y | w = xy \wedge x \in L_1 \wedge y \in L_2$.

> $M_3$ iterates through $w$ running $M_1$ on every possible $x$ in $w$. Since there is no $x, y$ pair that satisfy the concatenation, $M_1$ or $M_2$ will reject on $x, y$ for each iteration. Eventually once we have exhausted each $x, y$ pair, $M_3$ will reject once the number of 0's exceed the length of the input, that is $M_3$ rejects if we try to copy an $x$ where $|x| > |w|$. Thus, $M_3$ rejects if $w \notin L_1 \circ L_2$.

Therefore $M_3$ decides $L_1 \circ L_2$.

**Proof** $t_{M_3}(n) \in O(n^{k+1})$

For both $w \in L_1 \circ L_2 \land w \notin L_1 \circ L_2$, $M_3$ makes (at most) the following steps at each iteration:

- Copy $x$ onto Tape 2 -> $O(n)$
- Run $M_1$ on $x$ -> $O(n^k)$
- Read $M_1$ final state -> $O(1)$
- Clear Tape 2 and 3 -> $O(n)$
- Copy $y$ onto Tape 2 -> $O(n)$
- Run $M_2$ on $y$ -> $O(n^k)$
- Read $M_2$ final state -> $O(1)$
- Add another 0 to tape 4 -> $O(n)$

As we can see each of these steps are polynomial time and we can find the runtime of a single iteration as $O(n + n^k + 1 + n + n + n^k + 1 + n) = O(n^k)$. We do at most $n + 1$ iterations of this so $t_{M_3}(n) = O((n + 1)n^k) = O(n^k + 1)$ which is polynomial time.

Therefore $L_1 \circ L_2 \in P$ because we created a machine that decides the language in polynomial time.

### D

Prove $(L_1 \in NP \land L_2 \in NP) \rightarrow L_1 \circ L_2 \in NP$

## 2

### A

Prove that $SUBGRAPHISOMORPHISM \in NP$ (show $\exists V \in P$)

### B

Prove $HAMILTONIAN \leq_p SUBGRAPHISOMORPHISM$

## 3

Prove MAJORITY-SAT is NP-complete

**Prove** $3-SAT \leq_p MAJORITY-SAT$

We can map each clause in $3-SAT$ which consists of literals $a, b, c$ to a new clause in MAJORITY-SAT which consists of literals $a, b, c, x, y$, where $x, y$ are new literals not in our $3-SAT$ problem.

Since $x, y$ are new literals, they can be $T/F$ unlike $a, b, c$ which are $T$ iff they are $T$ in $3-SAT$.

**Proof:**

If $F \in 3-SAT$ then $f(F) \in MAJORITY-SAT$

> If $F \in 3-SAT$ then $\exists$ assignments where each clause is true.
>
> This means for all clauses in $F$, at least one literal is true.
>
> Then our CNF for $MAJORITY-SAT$ $(f(F))$ has at least one literal from all clauses in $F$ true.
>
> This means $f(F) \in MAJORITY-SAT$ because all clauses in $F$ have at least one true literal, so each clause in $MAJORITY-SAT$ have at least one true literal besides $x, y$. If $x, y$ are true then that clause will have majority true because we have one true literal from $F$ and $x, y$ are true.
>
> Therefore, if $F \in 3-SAT$ then $f(F) \in MAJORITY-SAT$

If $F \notin 3-SAT$ then $f(F) \notin MAJORITY-SAT$

> If $F \notin 3-SAT$ then there exists at least one clause in $F$ that is false. That means this clause has all false for the literals.
>
> For $f(F)$ that means that this clause will have 3 false literals from $F$ and $x, y$ which can be true or false. Even if both $x, y$ are true, we will not have a majority true for this clause. Meaning $f(F) \notin MAJORITY-SAT$
>
> Therefore if $F \notin 3-SAT$ then $f(F) \notin MAJORITY-SAT$

As such, our clause in $3-SAT$ is true iff our clause in $MAJORITY-SAT$ is true. So $3-SAT \leq_p MAJORITY-SAT$.

## 4

Prove the 0-1 integer programming problem is NP-complete

**Prove** $SAT \leq_p INT$

For our function, each literal in $SAT$ matches to one $x$ variable in $INT$ and each clause in $SAT$ matches to one linear inequality in $INT$.

Our function $f$ makes a set of linear equations mapping each variable in $F$ to the variabls in our system of linear inequalities. Then all $b_1 ... b_m$ are equal to $-1$. For coeffecients, $a_{i,j} = -1$ if $l_j$ is a literal in the $i$th clause, and $a_{i,j} = 0$ otherwise. For each negation of a literal $l_j$ in a clause $(i)$, we increase $b_i$ by 1 and flip the sign of $a_{i,j}$ (make $a_{i,j} = 1$)

**Proof:**

If $F \in SAT$ then $f(F) \in INT$

If $F \in SAT$ then there is an assignment to the literals where each clause is true.

If each clause is true, then at least one literal is true. This carries over to our system of inequalities $f(F)$. If all literals in a clause are simply the variable, this means for our inequality to be true at least one $x_i$ is equal to 1, so the sum is at most $-1 * 1 \leq -1$ so this inequality is true. If the clause contains negations of a variable, then the inequality will still hold because we incremented $b_j$ by 1 so that way the inequality accepts when that variable is 0 (if the negated variable is true then the inequality requires one other non negated variable to be 1 in order for the inequality to hold, this way we do not accept when we should not). Thus if $F \in SAT$ the same assignment that makes $F$ true can be used to make $f(F) \in INT$ true.

If $F \notin SAT$ then $f(F) \notin INT$

If $F \notin SAT$ then there is no assignment to the literals where each clause is true. This means at least one clause is false in $F$, that is all literals are false for this clause. In $f(F)$, there must also exist an inequality that is false. If there are no negations of variables in our clause $c$, then the inequality would contain all $x_i = 0$ so $0 + 0 + ... + 0 \leq -1$. For each negation, $b_c$ is incremented by 1, so if there are $y$ negations we get the inequality $y + 0 + 0 + ... + 0 \leq y - 1$ which is false. Thus if $F \notin SAT$ then $f(F) \notin INT$

Therefore, $F \in SAT \leftrightarrow f(F) \in INT$ so $SAT \leq_p INT$.