Computability and Complexity (Summer 2025) – Homework 2

Shir Altman, Yedidya Even-chen

August 2025

Problem 1

Let M be a poly-time DTM with $L = L(M)$, and M' be a poly-time NTM with $L' = L(M')$. For each of the following, determine whether it is true. If you cannot prove or disprove a statement, explain what unsolved problem is involved.

Part 1a

$L, L' \in P$

Solution: $L \in P$ is true by definition - a language is in P if there exists a poly-time DTM for it. However, $L' \in P$ is not necessarily true - the poly-time NTM only puts it in NP. L' might be in P, but until someone finds a poly-time DTM for it (or proves $P =_? NP$) this remains unknown. For example, if $L = 3SAT$, then there exists a poly-time NTM for it but as far as we know, 3SAT is not in P.

Part 1b

$L, L' \in NP$

Solution: This is true - $L' \in NP$ is true by definition of NP. And since $P \subseteq NP$, $L \in NP$ is also true.

Part 1c

$L, L' \in R$

Solution: This is true, as any NTM has an equivalent DTM, so $NP \subseteq R$. And since $P \subseteq NP$, we get $L, L' \in R$.

Part 1d

$L \cap L' \in P$

Solution: If $L' \in P$ then it is true, as P is closed under intersection. But until someone finds a poly-time DTM for L' or proves $P =_? NP$, then it is possible that $L' \in NP \setminus P$ and $P \neq NP$, in which case $L \cap L' \notin P$. This remains unknown.

Part 1e

If L' is NP-hard then there is a poly-time reduction from L to L'

Solution: This is true. If L' is NP-hard then by definition there is a poly-time reduction from any $L'' \in NP$ to L'. And since $P \subseteq NP$, there is a poly-time reduction from L to L'.

Part 1f

$L, L' \in PSPACE$

Solution: This is true. As stated in lecture 9, $P \subseteq NP \subseteq PSPACE$. $P \subseteq PSPACE$ follows from the fact that poly-time implies poly-space (because reaching a non-polynomial number of cells requires a non-polynomial number of moves, and thus non-polynomial time). And $NP \subseteq PSPACE$ follows from Savitch's Theorem, because transitioning from an NTM to a DTM requires only a quadratic growth in space.

Problem 2

Let $A \in DTIME(n^2)$, $A \in SPACE(n)$ be an algorithm for modular multiplication (i.e. $A(x, y, m) = xy \bmod m$ when $|x| = |y| = |m| = n$). Build a deterministic poly-time algorithm for modular exponentiation $B(x, z, m) = x^z \bmod m$, when $|x| = |z| = |m| = n$. What time and space complexity does it have? (Hint: use algorithm A as a black-box building block.)

Solution: First, we note the identity:

$$(a \cdot b) \bmod m = ((a \bmod m) \cdot (b \bmod m)) \bmod m$$

which can be generalized by induction to:

$$x^z \bmod m = \left(\prod_{i=1}^{z} x\right) \bmod m = \left(\prod_{i=1}^{z} (x \bmod m)\right) \bmod m$$

allowing us to use algorithm A to perform modular multiplication at each step instead of at the end. We also note that we can assume the input is given in binary notation, so z has n bits and can be written as a sum of powers of 2:

$$z = (a_{n-1}, a_{n-2}, \ldots, a_1, a_0) = \sum_{i=0}^{n-1} a_i 2^i$$

where $a_i \in \{0, 1\}$ for $0 \leq i < n$. Then we get the identity:

$$x^z = x^{\sum_{i=0}^{n-1} a_i 2^i} = \prod_{i=0}^{n-1} x^{a_i 2^i}$$

which we can use for algorithm B, and use algorithm A to calculate $x^z = \prod_{i=0}^{n-1} x^{a_i 2^i}$. We start with $result \leftarrow 1$, and then multiply by $x^{a_i 2^i}$ for $0 \leq i < n$. We note that we can square x each time and keep it in a variable *base*, so at iteration i, *base* is x^{2^i}. Then, if $a_i = 1$ then we multiply by x^{2^i}, and if $a_i = 0$ then $x^{a_i 2^i} = x^{0 \cdot 2^i} = x^0 = 1$.

Algorithm 1 Modular Exponentiation $B(x, z, m)$ using A

```
result ← 1
base ← A(x, 1, m)
while z > 0 do
    if z mod 2 = 1 then                ▷ check if least-significant bit of z is 1
        result ← A(result, base, m)
    z ← ⌊z/2⌋                          ▷ right shift z
    base ← A(base, base, m)            ▷ square mod m
return result
```

The loop iterates at most n times, and each time we perform:

1. check if the least-significant bit of z is 1: $O(n)$.
2. use algorithm A twice: $O(n^2)$.
3. right-shift z: $O(n)$.

In total, we get a polynomial runtime of $O(n^3)$. The space complexity of each loop iteration is $O(n)$, because we only keep constants between uses of A, and each constant takes $O(n)$ space. Each use of A takes $O(n)$ space, but we can reuse the space for each use of A. Giving us:

$$B \in DTIME(n^3) \qquad B \in SPACE(n)$$

Problem 3

Prove or refute: $DTIME(n^2) \setminus DTIME(n^{1.9}) \neq \emptyset$. (here, "\" stands for set minus, "$\emptyset$" stands for empty set).

Solution: This statement is true. A corollary of the Time Hierarchy Theorem (from lecture 8) states that for all $a, b \in \mathbb{R}$ such that $1 \leq a < b$, $DTIME(n^a) \subsetneq DTIME(n^b)$ holds. Meaning there exists some x such that $x \in DTIME(n^2) \setminus DTIME(n^{1.9})$ so $DTIME(n^2) \setminus DTIME(n^{1.9}) \neq \emptyset$.

Problem 4

For each of the following languages, determine whether it is RE-complete, R-complete, NP-complete, P-complete, PSPACE-complete, or NL-complete. Explain why.

Part 4a

$3SAT = \{\langle\varphi\rangle \mid \varphi \text{ is a satisfiable 3CNF}\}$

Solution: From lecture 7, we know that 3SAT is NP-complete: We know it is NP because a boolean assignment is a poly-size witness, and verification of a satisfying assignment is poly-time. We also know it is NP-hard, because SAT is NP-hard (by the Cook-Levin Theorem) and we have seen the reduction for $SAT \leq_p 3SAT$ (splitting each clause into triplets).

Formally, the reduction $SAT \leq_p 3SAT$ is as follows: we will replace every clause $C = (\ell_1 \vee \ell_2 \vee ... \vee \ell_m)$ with a **clause gadget**, $g(C)$: If $m = 3$, then $g(C) \coloneqq C$. If $m < 3$, then we duplicate literals to complete to 3:

$$g(\ell_1) \coloneqq (\ell_1 \vee \ell_1 \vee \ell_1), \quad g(\ell_1 \vee \ell_2) \coloneqq (\ell_1 \vee \ell_2 \vee \ell_1)$$

If $m > 3$, then we need to split C into groups of 3. We introduce $m-3$ new variables: $y_1^C, y_2^C, ..., y_{m-3}^C$, and define the gadget:

$$g(C) \coloneqq (\ell_1 \vee \ell_2 \vee y_1) \wedge (\overline{y_1} \vee \ell_3 \vee y_2) \wedge (\overline{y_2} \vee \ell_4 \vee y_3) \wedge \cdots \wedge (\overline{y_{m-4}} \vee \ell_{m-2} \vee y_{m-3}) \wedge (\overline{y_{m-3}} \vee \ell_{m-1} \vee \ell_m)$$

Then, define $f(\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m) \coloneqq g(C_1) \wedge g(C_2) \wedge \cdots \wedge g(c_m)$. It remains to show that $\varphi \in SAT \iff f(\varphi) \in 3SAT$.

Assume $\varphi \in SAT$. Then there exists some satisfying assignment a for φ. Define an assignment a' for $f(\varphi)$: for every original variable x of φ, set $a'(x) \coloneqq a(x)$. We now define the assignment for the new y variables: As a satisfies φ, in every clause there was at least one satisfied literal, ℓ_i. Define: $a'(y_j^C) \coloneqq 1$ if $j < i$, and $a'(y_j^C) \coloneqq 0$ else. Then, All gadgets that appear before ℓ_i will be satisfied because if the y_j. The gadget with ℓ_i is satisfied because of ℓ_i. And the ones after are all satisfied because of the $\overline{y_j}$.

Assume $f(\varphi) \in 3SAT$. Then there exists some satisfying assignment a' for $f(\varphi)$. Define an assignment a for φ: for every original variable x of φ, set $a(x) \coloneqq a'(x)$. We prove that this is a satisfying assignment: assume for a contradiction that it is not, meaning there is some clause $C = (\ell_1 \vee \ell_2 \vee ... \vee \ell_m)$ such that $\ell_1 = \ell_2 = \cdots = \ell_m = 0$. If $m \leq 3$ then they are the same variables, so if C is unsatisfied then so is $g(C)$, contradiction. If $m > 3$, consider $g(C)$: all the original variables of C are 0, but $g(C) = 1$, so each clause of $g(C)$ must be satisfied because of the y literal. This means that the underlined literals must get 1:

$$g(C) \coloneqq (\ell_1 \vee \ell_2 \vee \underline{y_1}) \wedge (\overline{y_1} \vee \ell_3 \vee \underline{y_2}) \wedge (\overline{y_2} \vee \ell_4 \vee \underline{y_3}) \wedge \cdots \wedge (\overline{y_{m-4}} \vee \ell_{m-2} \vee \underline{y_{m-3}}) \wedge (\overline{y_{m-3}} \vee \ell_{m-1} \vee \ell_m)$$

note that there is no other option: $\ell_1 = \ell_2 = 0$, so y_1 must be 1. Then $\overline{y_1} = 0$, so y_2 must be 1... etc. We get to y_{m-3} which must be 1 so $\overline{y_{m-3}} = 0$, leaving the last clause unsatisfied. Contradiction.

Part 4b

$EQ_{TM} = \{\langle M, M' \rangle \mid L(M) = L(M')\}$

Solution: As we previously showed in Homework 1, EQ_{TM} is not even in RE, so it cannot be RE-complete (or any of the other, stricter classes listed here). We recount the proof from HW 1:

$$\mathrm{EQ_{TM}} \notin RE$$

To see why it's not in RE, we reduce from $\overline{A_{\mathrm{TM}}}$, which is known not to be in RE. Given an input $\langle M, w \rangle$, we build two machines M_1 and M_2:

- M_2: Rejects every input, so $L(M_2) = \emptyset$.
- M_1: Ignores its input and runs M on w. If M accepts w, M_1 accepts; otherwise, it rejects.

If M does not accept w, then M_1 also rejects every input, so $L(M_1) = \emptyset$ and $L(M_1) = L(M_2)$. In this case, $\langle M_1, M_2 \rangle \in \mathrm{EQ_{TM}}$.

If M does accept w, then M_1 accepts every string ($L(M_1) = \Sigma^*$), which is different from $L(M_2) = \emptyset$, so $\langle M_1, M_2 \rangle \notin \mathrm{EQ_{TM}}$.

This means:

$$\langle M, w \rangle \in \overline{A_{\mathrm{TM}}} \quad \iff \quad \langle M_1, M_2 \rangle \in \mathrm{EQ_{TM}}$$

Since $\overline{A_{\mathrm{TM}}} \notin \mathrm{RE}$, $\mathrm{EQ_{TM}}$ cannot be in RE either, and therefore it cannot be RE-complete.

Part 4c

$PATH = \{\langle G, s, t \rangle \mid G \text{ is a directed graph, and there is a path from } s \text{ to } t\}$

Solution: As shown in lecture 10, PATH is NL-complete.

First, $PATH \in NL$ as we have shown a nondeterministic logarithmic TM which decides PATH:

Algorithm 2 Determining if there exists a path from s to t in directed graph G

```
1: a ← ⟨s⟩
2: count ← 0
3: while a ≠ ⟨t⟩ and count < |V(G)| do
4:     Non-deterministically, choose b ∈ V(G) such that (a, b) ∈ E(G)
5:     a ← ⟨b⟩
6:     count ← count + 1
7: if a = ⟨t⟩ then
8:     accept
9: else
10:    reject
```

On any branch, we only keep a pointer to a vertex and a counter. Those require $O(\log |V(G)|)$ space.

Next, $PATH \in NL\text{-}hard$. We show that for any $A \in NL$, $A \leq_L PATH$ holds. Let N be an $O(\log n)$-space NTM deciding A. We define the reduction. Given input x for N, we build $\langle G, s, t\rangle$:

- Every configuration of N defines a vertex of G. There are $O(n^2)$ configurations.
- Define $s \coloneqq c_{start}, \quad t \coloneqq c_{accept}$
- For every c_i, c_j such that c_i yields c_j, we add an edge (c_i, c_j) to $E(G)$.

Then, if $x \in A$, it means there is a series of movements between configurations that start with c_{start} and ends with c_{accept}, which exactly corresponds to a path from s to t in G. And a path $s \rightsquigarrow t$ describes a series of configurations $c_{start} \rightsquigarrow c_{accept}$.
This reduction is also log-space, because we just need to keep the configuration number (the number of configurations is polynomial, so this needs only log-space) and we can check the movements between configurations in log-space. And of this happens in poly-time.

Problem 5

Show that if $L, L' \in NL$ then $L^* = \{xy \mid x \in L, y \in L'\} \in NL$.

Solution: $L, L' \in NL$ implies the existence of log-space NTM's M, M' which decide L, L' respectively. We use these NTM's to define a log-space NTM M^* deciding L^*: given input w, let $n = |w|$ and non-deterministically choose an index $i \in \{0, ..., n\}$ such that $x = w[0 : i]$ and $y = w[i : n]$. The index needs $O(\log n)$ space. Run $M(x)$, reject if it rejects. If it accepts, then erase that work tape and simulate $M(y)$. Accept if $M(y)$ accepts. Each branch is independent and can reuse the same space, so M^* is log-space and thus $L^* \in NL$.

Problem 6

Prove that if L is PSPACE-hard then it is NP-hard.

Solution: Recall from lecture 9 that the space complexity of an NTM is bounded by its longest branch, so $NP \subseteq NPSPACE$. And Savitch's Theorem gives us $NPSPACE = PSPACE$. So $NP \subseteq PSPACE$. Recall the definition of PSPACE-hardness: L is PSPACE-hard if $L' \leq_p L$ (a poly-time reduction, which is also poly-space) for any $L' \in PSPACE$. And recall the definition of NP-hardness: L is NP-hard if $L' \leq_p L$ for any $L' \in NP$. Let some $L' \in NP$. By $NP \subseteq PSPACE$, we get $L' \in PSPACE$. From the fact that L is PSPACE-hard, we get $L' \leq_p L$. Implying that L is NP-hard.

Problem 7

An LBA (linearly bounded automaton) is a single-tape DTM with linearly bounded tape size (the number of cells equals cn for some constant c and input length n). To what time and space complexity classes does the following language belong?

$$A_{LBA} = \{\langle M, x\rangle \mid M \text{ is an LBA accepting } x\}$$

Solution: We assume that the language refers to pairs $\langle M, x\rangle$ where M is known to be an LBA, and the unknown is whether M accepts x. (otherwise, it is undecidable because as long as $M(x)$ doesn't stop, we can never know if it will eventually accept x). To check if an LBA M accepts x, We want to simulate $M(x)$, keeping track of the number of steps. Let N be the number of possible configurations of M. If $M(x) = 1$ then accept, If $M(x) = 0$ then reject. After $N + 1$ steps, we know that some configuration has repeated, so we are in a loop and we can reject. The simulation of $M(x)$ requires poly-space and $|\Gamma|^{O(n)}$ time, so $A_{LBA} \in PSPACE$ and $A_{LBA} \in EXPTIME$. Specifically, the space required is: cn for the tape contents, $O(1)$ for the state, $O(\log(cn))$ for the head position, $O(\log(N))$ for the counter. It remains to find a way to keep track of how many configurations we've seen and count to N. We know that $N = |Q| \cdot cn \cdot |\Gamma|^{cn}$. We can allocate $k := \lceil \log_2 N \rceil$ tape-places for a counter which we start at 0 and increment with every step. When that counter overflows, we know we have made at least $N + 1$ steps. It remains to calculate k in poly-space. We have:

$$k = \lceil \log_2 N \rceil \leq \lceil \log_2 |Q| \rceil + \lceil \log_2 cn \rceil + cn \lceil \log_2 |\Gamma| \rceil$$

$|Q|, |\Gamma|$ are constants, we can write them in binary and count the number of bits used. We have n, and we can multiply in poly-space to get cn which we can write and count bits to get $\lceil \log_2 cn \rceil$. We can then add and multiply everything as needed to get k. So the total space needed is $O(n) + O(k)$. And since k depends on M (how many states, alphabet size) then the space is $O(|x| + \log |M|) \subseteq O(|\langle M, x\rangle|)$ which is linear in the input size. So $A_{LBA} \in SPACE(m)$ for input size m.

Problem 8

Prove that

$$L = \{\langle M, x, t\rangle \mid \exists w : \text{ a DTM } M(x, w) \text{ stops in } \leq t \text{ steps}\}$$

is NP-hard.

Solution: We show a reduction $SAT \leq L$: Given a CNF formula φ, build a poly-time verifier M_φ for it. Set t to be the number of steps needed to verify. This number depends on the encoding of φ and the verifier, but can always be calculated and is polynomial in φ. M_φ gets φ, t, and an assignment w, and halts iff $\varphi = T$ under w. Then:

$$\varphi \in SAT \iff \exists w : M_\varphi(\varphi, w) = 1 \iff \langle M_\varphi, \varphi, t\rangle \in L$$