Chapter 4 : Hierarchy theorems and intractability : Solutions of the exercises

Section 4.1

1. $[2n^2 + 3n + 4]$ First obtain the binary representation of n by counting the input size using a binary counter. This requires $O(n \log n)$ time and $O(\log n)$ space. Once n is available, computing the expression $2n^2 + 3n + 4$ can be done in $O(\log^2 n)$ time and $O(\log n)$ space.

$[2^n]$ Write 1 to the output. For each input symbol append a 0. In $O(n)$ time and $O(n)$ space we end up with the binary representation of 2^n.

$[3^n]$ Count n in binary in $O(n \log n)$ time and $O(\log n)$ space. Raise 3 to the exponent n using the conventional square-and-multiply algorithm. The exponentiation requires $O(\log n)$ multiplications and squarings, each on operands of size $O(\log 3^n)$, i.e., $O(n)$. Using the high-school quadratic multiplication routine yields a running time of $O(n^2 \log n)$ and a space requirement of $O(n)$.

$[5^{n^2}]$ Similar to 3^n, except that we have to compute n^2 from n, a process that can be done using $O(\log^2 n)$ time and $O(\log n)$ space.

2. $\mathrm{TIME}(n^k) \subseteq \mathrm{SPACE}(n^k/\log n)$ by the given assertion. Since $n^k \log n$ is $o(n^k)$ and n^k is space constructible, by the space hierarchy theorem $\mathrm{SPACE}(n^k/\log n) \subsetneqq \mathrm{SPACE}(n^k)$.

We may have two infinite sequences $T_1, T_2, \ldots$ and $S_1, S_2, \ldots$ of sets with each $T_k \subsetneqq S_k$ and, at the same time, with $\bigcup_{k\in\mathbb{N}} T_k = \bigcup_{k\in\mathbb{N}} S_k$. For example, one may take $T_k := \{1, 2, \ldots, k\}$ and $S_k := \{1, 2, \ldots, 2k\}$. One may easily construct an example in which each T_k and S_k are infinite. (Just add the negative integers to each T_k and S_k in the above example.)

3. Obviously $\mathrm{NTIME}(n^k) \subseteq \mathrm{NSPACE}(n^k)$. By Savitch's theorem $\mathrm{NSPACE}(n^k) \subseteq \mathrm{SPACE}(n^{2k})$. Finally, by the space hierarchy theorem $\mathrm{SPACE}(n^{2k}) \subsetneqq \mathrm{SPACE}(n^{2k+1}) \subseteq \bigcup_{i\in\mathbb{N}} \mathrm{SPACE}(n^i) = \mathrm{PSPACE}$.

This does not imply $\mathrm{NP} = \bigcup_{k\in\mathbb{N}} \mathrm{TIME}(n^k) \subsetneqq \mathrm{PSPACE}$, since a sequence of proper subsets of PSPACE may have a union which is the full of PSPACE.

4. We may assume that $f(n)$ is positive for all $n \geqslant 0$. Let M be a recognizer for L, such that for all n every accepting branch of M takes $\leqslant d f(n)$ steps on an input of size n. I want to construct an $O(f(n) \log f(n))$-time decider M' for L. M' is designed as a two-track machine. By time constructibility M' first computes the value $d f(n)$ in binary and stores it in a counter in the second track. M' then simulates M on the first track and decrements the counter by 1 after every step of M. The counter is kept close to the head of M'; whenever M's head moves, the counter is also shifted. If M halts before the counter reaches zero, M' echoes M's decision and halts. If M does not halt in $d f(n)$ steps (i.e., the counter attempts to become negative), M' rejects and halts.

Clearly, M' simulates M correctly. Since the binary counter on the second track is of size $O(\log f(n))$, decrementing and relocating it requires $O(\log f(n))$ steps of M', i.e., M' simulates each single step of M in $O(\log f(n))$ time, i.e., M' has a running time of $O(f(n) \log f(n))$.

The last statement of the exercise follows from the fact that if $f(n)$ is $O(n^k)$, then $f(n) \log f(n)$ is $O(n^{k+1})$. Thus a poly-time recognizable language is poly-time decidable too. The converse implication is obvious.

Section 4.2

1. The following is a poly-time alternating algorithm for UNSAT. Compare this algorithm with the nondeterministic algorithm for SAT.

> **Input:** $\langle \phi \rangle$.
> 1. Universally select an assignment of the variables of ϕ.
> 2. Evaluate ϕ at the selected assignment.
> 3. If the evaluation outcome is 0, *accept*, else *reject*.

In general, let $L \in \mathrm{NP}$ have nondeterministic poly-time decider N. I convert N to an alternating poly-time decider $\bar{N}$ of $\bar{L}$. $\bar{N}$ is identical to N, except that N's accepting state is rejecting for $\bar{N}$, N's rejecting state is accepting for $\bar{N}$, and every other state of N is a universal state for $\bar{N}$. It is easy to prove by the recursive construction of trees that $\mathcal{L}(\bar{N}) = \bar{L}$. Moreover, $\bar{N}$ produces identical computation trees as N (except that the labels are changed from (invisible) $\vee$ to $\wedge$ at the nodes); so $\bar{N}$ runs in poly-time too.

2. The following poly-time alternating algorithm decides HALFCYCLE:

> **Input:** $\langle G \rangle$, where G is a directed graph.
> 1. Let $m := \lfloor n(G)/2 \rfloor$.
> 2. Existentially select m vertices $u_1, \ldots, u_m$ of G.
> 3 If $(u_1, \ldots, u_m)$ is not a cycle in G, *reject*.
> 4. Universally select an integer k in the range $m < k \leqslant n(G)$.
> 5. Universally select k vertices $v_1, \ldots, v_k$ of G.
> 6. If $(v_1, \ldots, v_k)$ is a cycle in G, *reject*, else *accept*.

Compare this algorithm with the algorithm of Exercise 5 of the Midsem test-paper.

3. The following construction proves all the assertions of this exercise. Let N be an alternating TM accepting language L. I design an alternating TM $\bar{N}$ to accept $\bar{L}$ as follows. $\bar{N}$ is identical to N with the exceptions:

- N's accepting state is rejecting for $\bar{N}$.
- N's rejecting state is accepting for $\bar{N}$.
- N's existential states are universal in $\bar{N}$.
- N's universal states are existential in $\bar{N}$.

A proof that $\mathcal{L}(\bar{N}) = \bar{L}$ follows from the recursive construction of trees. N and $\bar{N}$ have identical running times.

4. [if] $\mathrm{P} = \mathrm{PH}$ implies $\mathrm{NP} = \Sigma_1\mathrm{P} \subseteq \mathrm{PH} = \mathrm{P}$.

[only if] Given that $\mathrm{P} = \mathrm{NP}$, I inductively demonstrate that $\Sigma_i\mathrm{P} = \Pi_i\mathrm{P} = \mathrm{P}$ for all $i \geqslant 1$. For $i = 1$, we have $\Sigma_1\mathrm{P} = \mathrm{NP} = \mathrm{P}$ by hypothesis, whereas $\Pi_1\mathrm{P} = \mathrm{coNP} = \mathrm{P}$, since P is a deterministic class and so closed under complementation. Now assume that we have proved $\Sigma_i\mathrm{P} = \Pi_i\mathrm{P} = \mathrm{P}$ for some $i \geqslant 1$. Take a language $L \in \Sigma_{i+1}\mathrm{P}$. Consider a poly-time ATM N for L that makes at most $i+1$ nondeterministic choices. Let α be an input for N and let c be a configuration of N immediately after it makes the first non-deterministic choice (existential). Since N runs in poly-time, c is of length bounded by a polynomial in $|\alpha|$. We run, with $\langle N, c \rangle$ as input, an ATM N' that, given the encoding of an ATM N and a configuration c of N on some input, simulates N starting from the configuration c. Clearly, N' accepts $\langle N, c \rangle$ for some c, if and only if N accepts α. Moreover, N' makes $\leqslant i$ choices and so decides a $\Pi_i\mathrm{P}$ language. By induction, $\mathcal{L}(N') = \mathcal{L}(M)$ for a poly-time DTM M. But then we can run M on $\langle N, \alpha \rangle$ to know the same decision of N on α. Since M makes no nondeterministic choices, it follows that $L \in \Sigma_1\mathrm{P} = \mathrm{NP} = \mathrm{P}$. Thus $\Sigma_{i+1}\mathrm{P} \subseteq \mathrm{P}$. The reverse inclusion is obvious. Since $\mathrm{co}\Sigma_{i+1}\mathrm{P} = \Pi_{i+1}\mathrm{P}$, it follows that $\Pi_{i+1}\mathrm{P} = \mathrm{P}$ too.

5. I prove part (a) only, the proof for the other part being analogous. Let L be a PH-complete language. Since $\mathrm{PH} = \bigcup_{i \in \mathbb{N}} \Sigma_i\mathrm{P}$, there exists i_0 for which $L \in \Sigma_{i_0}\mathrm{P}$. Let N be a poly-time $\Sigma_{i_0}\mathrm{P}$ ATM for L. Now choose any $A \in \Sigma_i\mathrm{P}$ for some $i \geqslant i_0$. By completeness of L there exists a poly-time reduction from A to L. This reduction followed by a simulation of N implies that $L \in \Sigma_{i_0}\mathrm{P}$.

6. The choice of i is questionable in the given proof. There may exist some language $L \in \mathrm{AP}$ for which the number of nondeterministic choices increases monotonically with the input size, i.e., the value of i goes unbounded as the input size goes to infinity, i.e., i cannot be chosen in an input-independent manner as described in the proof. $\Sigma_i\mathrm{P}$ and $\Pi_i\mathrm{P}$ refer to the classes, where at most i choices are sufficient, irrespective of the length of the input. Thus AP may potentially contain languages that are in neither of the classes $\Sigma_i\mathrm{P}$ or $\Pi_i\mathrm{P}$ for any $i \in \mathbb{N}$.

Section 4.3

1. **(a)** Take a poly-time DTM M for L and replace every oracle call of L by the simulation of M. This implies $\mathrm{P}^L \subseteq \mathrm{P}$. The reverse inclusion is obvious.

(b) P^L is closed under complementation, so that $\mathrm{coNP} = \mathrm{P}^L = \mathrm{NP}$.

2. It is easy to see that the language

$$\mathrm{BIGCYCLE} := \{\langle G\rangle \mid G \text{ is a directed graph having a cycle of length} > n(G)/2\}$$

is in NP. Let f be a poly-time reduction from BIGCYCLE to SAT. We then have the following nondeterministic poly-time algorithm for HALFCYCLE that uses the SAT oracle.

Input: $\langle G\rangle$ for a directed graph G.
1. Let $m := n(G)/2$.
2. Nondeterministically select m vertices $u_1, \ldots, u_m$ of G.
3. If $(u_1, \ldots, u_m)$ is not a cycle in G, *reject*.
4. Use the poly-time reduction f to obtain $\alpha := f(\langle G\rangle)$.
5. Query the SAT oracle about α.
6. If the oracle answers YES, *reject*, else *accept*.

3. (a) Since $\mathrm{SAT} \in \mathrm{NP}$, $\mathrm{P}^{\mathrm{SAT}} \subseteq \bigcup_{A\in\mathrm{NP}} \mathrm{P}^A = \mathrm{P}^{\mathrm{NP}}$. Now let $L \in \mathrm{P}^{\mathrm{NP}}$, i.e., $L \in \mathrm{P}^A$ for some $A \in \mathrm{NP}$. Let N be a poly-time TM for L, that uses the oracle for A. Since SAT is NP-complete, one may first convert in poly-time an instance for A to an instance for SAT and then ask the SAT oracle, instead of asking the oracle for A straightaway. Though this replacement of A by SAT may make the query inefficient, it retains the polynomial behavior of M. Thus $L \in \mathrm{P}^{\mathrm{SAT}}$ too, i.e., $\mathrm{P}^{\mathrm{NP}} \subseteq \mathrm{P}^{\mathrm{SAT}}$.

Since UNSAT is coNP-complete, we analogously have $\mathrm{P}^{\mathrm{UNSAT}} = \mathrm{P}^{\mathrm{coNP}}$. Finally, note that $\mathrm{P}^{\mathrm{SAT}} = \mathrm{P}^{\mathrm{UNSAT}}$, since an answer of the SAT oracle is readily interpreted as an answer for the UNSAT oracle, and vice versa.

(b) As in Part (a) NP-completeness of SAT implies $\mathrm{NP}^{\mathrm{NP}} = \mathrm{NP}^{\mathrm{SAT}}$. Thus it suffices to show that $\mathrm{NP}^{\mathrm{SAT}} = \Sigma_2\mathrm{P}$.

$\left[\mathrm{NP}^{\mathrm{SAT}} \subseteq \Sigma_2\mathrm{P}\right]$ Let $L \in \mathrm{NP}^{\mathrm{SAT}}$ and let M be a nondeterministic poly-time decider of L relative to the SAT oracle. I first convert M to a nondeterministic poly-time decider M' of L, that makes only *one* query of the SAT oracle and accepts if and only if the answer is NO. M' can nondeterministically guess the correct answers to the oracle queries. However, these guesses must be validated for correctness. If a query about ϕ made by M returns YES, then ϕ is satisfiable and M can replace the oracle call by a non-deterministic guess for a satisfying assignment. Now assume that $\phi_1, \ldots, \phi_k$ are all the oracle queries receiving the response NO. This happens, if and only if the formula $\phi_1 \vee \cdots \vee \phi_k$ is unsatisfiable. Thus instead of making individual queries about $\phi_1, \ldots, \phi_k$, M makes a single query about $\phi_1 \vee \cdots \vee \phi_k$ at the end.

In the second step I convert M' to a $\Sigma_2\mathrm{P}$ decider M'' for L. Note that M' makes existential choices followed by an oracle call that receives the answer NO in case of acceptance. Thus this oracle call can be replaced by a universal branching on all possible truth assignments of the variables in $\phi_1 \vee \cdots \vee \phi_k$ and accepting if and only if the assignments are all non-satisfying.

$\left[\Sigma_2\mathrm{P} \subseteq \mathrm{NP}^{\mathrm{SAT}}\right]$ Let N be a $\Sigma_2\mathrm{P}$ decider for a language L. Let c be the configuration of N immediately after it makes the (first) nondeterministic choice. Then the machine N' that simulates N starting from c accepts $\langle N, c\rangle$ for some c if and only if N accepts α. Also $\mathcal{L}(N') \in \Pi_1\mathrm{P} = \mathrm{coNP}$. Thus the computation of N' can be replaced by a reduction to a Boolean formula, calling the SAT oracle on this formula and accepting if and only if the oracle returns NO.