V22.0453-001: Honors Theory of Computation

Problem Set 4 Solutions

Problem 1

1. $L_1 = \{\langle M, w, t\rangle : M \text{ halts on } w \text{ in } t \text{ steps}\}$

   Answer: L_1 is decidable. The following TM decides L_1:

   On input $\langle M, w, t\rangle$:

   (a) Simulate M on w for at most t steps.

   (b) If M accepts, output ACCEPT. Else output REJECT.

2. $L_2 = \{\langle M\rangle : \varepsilon \in L(M)\}$

   Answer: L_2 is undecidable. To show this, we use Rice's theorem. Let TR denote the class of all Turing-recognizable languages. Let $\mathcal{P} = \{L \in \text{TR} : \varepsilon \in L\}$. Clearly $\mathcal{P} \neq \emptyset$ and $\overline{\mathcal{P}} \neq \emptyset$. Therefore, by Rice's Theorem, $L_2 = L_{\mathcal{P}} = \{\langle M\rangle : L(M) \in \mathcal{P}\}$ is undecidable.

3. $L_3 = \{\langle M\rangle : M \text{ halts on } \varepsilon\}$

   Answer: L_3 is undecidable. To show this, suppose that a TM D decides L_3. Then the following TM would decide the above L_2, that is, whether $\varepsilon \in L(M)$ for any given $\langle M\rangle$:

   On input $\langle M\rangle$:

   (a) Simulate D on $\langle M\rangle$.

   (b) If D rejects (i.e. if M does not halt on ε), output REJECT.

   (c) If D accepts, simulate M on ε.

   (d) If M accepts ε, output ACCEPT. If M rejects ε, output REJECT.

   However, L_2 is undecidable. Therefore L_3 is undecidable.

4. $L_4 = \{\langle M\rangle : M \text{ halts on some input}\}$

   Answer: L_4 is undecidable. We show that A_{TM} reduces to L_4. Since A_{TM} is undecidable, it follows that L_4 is undecidable.

   The reduction from A_{TM} to L_4 maps $\langle M, w\rangle$ to $\langle M'\rangle$ where M' is the following TM with M and w built in:

   On input x:

   (a) Simulate M on w.

   (b) If M accepts w, output ACCEPT. If M rejects w, output REJECT.

   If $\langle M, w\rangle \in \text{A}_{\text{TM}}$, that is, if $w \in L(M)$, then $L(M') = \Sigma^*$, so $\langle M'\rangle \in L_4$. If $\langle M, w\rangle \notin \text{A}_{\text{TM}}$, that is, if $w \notin L(M)$, then $L(M') = \emptyset$, so $\langle M'\rangle \notin L_4$. Therefore the above mapping is a reduction from A_{TM} to L_4.

5. L_5 $\{\langle M\rangle : L(M)$ is context-free$\}$

   Answer: L_5 is undecidable. To show this, we use Rice's theorem. Let TR denote the class of all Turing-recognizable languages. Let $\mathcal{P} = \{L \in \text{TR} : L \text{ is a CFL}\}$. Since every CFL is Turing-recognizable, and there exist Turing-recognizable languages that are not context-free, we have that $\mathcal{P} \neq \emptyset$ and $\overline{\mathcal{P}} \neq \emptyset$. Therefore, by Rice's Theorem, $L_5 = L_{\mathcal{P}} = \{\langle M\rangle : L(M) \in \mathcal{P}\}$ is undecidable.

Problem 2

1. $\exists$ constants $c < d$ such that $n^d = O(n^c)$

   Answer: FALSE. If $c < d$, then
   $$\lim_{n\to\infty} \frac{n^d}{n^c} = \infty,$$
   so $n^d \neq O(n^c)$.

2. $10^{10} \cdot n^{1000} = o(2^{0.001n})$

   Answer: TRUE. By elementary techniques in calculus, we have
   $$\lim_{n\to\infty} \frac{10^{10} \cdot n^{1000}}{2^{0.001n}} = 0,$$
   so $10^{10} \cdot n^{1000} = o(2^{0.001n})$.

3. $n^{10} = o(2^{\log^2 n})$

   Answer: TRUE. Note that $n^{10} = 2^{10 \log n}$. Comparing the exponents of the two functions, we have that $10 \log n = o(\log^2 n)$. Therefore, $n^{10} = o(2^{\log^2 n})$.

4. $2^{\sqrt{\log n}} = o(\sqrt{n})$

   Answer: TRUE. Note that $\sqrt{n} = 2^{\frac{1}{2}\log n}$. Comparing the exponents of the two functions, we have that $\sqrt{\log n} = o(\frac{1}{2}\log n)$. Therefore, $2^{\sqrt{\log n}} = o(\sqrt{n})$.

5. $n^{\log n} = o(2^{\sqrt{n}})$

   Answer: TRUE. Note that $n^{\log n} = 2^{\log^2 n}$. omparing the exponents of the two functions, we have that $\log^2 n = o(\sqrt{n})$. Therefore, $n^{\log n} = o(2^{\sqrt{n}})$.

Problem 3

Solution: We show that **P** is closed under the star operation by dynamic programming. Let $L \in \mathbf{P}$, and let A be a polynomial-time algorithm that decides L. We construct a polynomial-time algorithm B that decides L^*, as follows. On input $w = w_1 \cdots w_n$, algorithm B constructs a table T such that $T[i, j] = 1$ if $w_i \cdots w_j \in L^*$, and $T[i, j] = 0$ otherwise. Details follow.

Algorithm B:

On input $w = w_1 \cdots w_n$,

1. If $w = \varepsilon$, output ACCEPT and halt.
2. Initialize $T[i, j] = 0$ for each $0 \leq i \leq j \leq n$.
3. For $i = 1$ to n: [1]
   (a) Run M on w_i to decide whether $w_i \in L$.
   (b) If $w_i \in L$, then set $T[i, i] = 1$.
4. For $\ell = 2$ to n: [2]
   For $i = 1$ to $n - \ell + 1$: [3]
   (a) Let $j = i + \ell - 1$. [4]
   (b) Run M on $w_i \cdots w_j$ to decide whether $w_i \cdots w_j \in L$.
   (c) If $w_i \cdots w_j \in L$, then set $T[i, j] = 1$.
   (d) For $k = i$ to $j - 1$: [5]
   If $T[i, k] = 1$ and $T[k, j] = 1$, then set $T[i, j] = 1$.
5. Output ACCEPT if $T[1, n] = 1$; else output REJECT. [6]

It is not hard to see that Algorithm B correctly decides L^* provided that Algorithm A correctly decides L. Moreover, Algorithm B runs in $O(n^3)$ stages, and each stage takes polynomial-time as A runs in polynomial-time. Therefore, Algorithm B is a polynomial-time algorithm that decides L^*.

Problem 4

Solution: Clearly Double-SAT $\in$ **NP**, since a NTM can decide Double-SAT as follows: On input a Boolean formula $\varphi(x_1, \ldots, x_n)$, nondeterministically guess 2 assignments and verfify whether both satisfy φ. To show that Double-SAT is **NP-Complete**, we give a reduction from SAT to Double-SAT, as follows:

On input $\varphi(x_1, \ldots, x_n)$:

1. Introduce a new variable y.
2. Output formula $\varphi'(x_1, \ldots, x_n, y) = \varphi(x_1, \ldots, x_n) \wedge (y \vee \overline{y})$.

If $\varphi(x_1, \ldots, x_n) \in SAT$, then φ has at least 1 satisfying assignment, and therefore $\varphi'(x_1, \ldots, x_n, y)$ has at least 2 satisfying assignments as we can satisfy the new clause $(y \vee \overline{y})$ by assigning either $y = 1$ or $y = 0$ to the new variable y, so $\varphi'(x_1, \ldots, x_n, y) \in$ Double-SAT. On the other hand, if $\varphi(x_1, \ldots, x_n) \notin SAT$, then clearly $\varphi'(x_1, \ldots, x_n, y) = \varphi(x_1, \ldots, x_n) \wedge (y \vee \overline{y})$ has no satisfying assignment either, so $\varphi'(x_1, \ldots, x_n, y) \notin$ Double-SAT. Therefore, SAT $\leq_P$ Double-SAT, and hence Double-SAT is **NP-Complete**.

[1] Test each substring of length 1
[2] ℓ is the length of the substring
[3] i is the start position of the substring
[4] j is the end position of the substring
[5] k is the split position
[6] $T[1, n] = 1$ if and only if $w = w_1 \cdots w_n \in L^*$.

Problem 5

Solution:

(a) In an $\neq$-assignment to φ, each clause has at least one satisfied literal and one unsatisfied literal. The negation of an $\neq$-assignment preserves this property, and thus is an $\neq$-assignment too.

(b) Let φ be any 3-CNF formula. Let φ' be the 3-CNF formula which is the output of the given reduction on input φ. We show that $\varphi \in$ 3-SAT if and only if $\varphi' \in \neq$SAT, and therefore the reduction is correct.

Suppose that $\varphi \in$ 3-SAT, that is, φ is satisfiable. Then we can obtain an $\neq$-assignment to φ' by extending a satisfying assignment to φ in such a way that we assign 1 to z_i if both literals y_1 and y_2 in clause C_i are unsatisfied[7]; else we assign 0 to z_i. Finally we assign 0 to b. It is clear that the extended assignment satisfies φ', and moreover it is an $\neq$-assignment to φ'. Therefore, $\varphi' \in \neq$SAT.

Suppose that $\varphi' \in \neq$SAT, that is, φ' has an $\neq$-assignment. Then we can obtain a satisfying assignment to φ as follows. By part (a), we may assume without loss of generality that the $\neq$-assignment assigns 0 to b, for otherwise, simply negate the assignment. This $\neq$-assignment cannot assign 0 to all of y_1, y_2 and y_3 as doing so would force one of the two clauses, $(y_1 \vee y_2 \vee z_i)$ and $(\overline{z_i} \vee y_3 \vee b)$, to have all 0's. Hence restricting this assignment to the variables of φ yields a satisfying assignment to φ.

(c) Clearly, $\neq$ SAT $\in$ **NP**, as it is easy to verify whether an assignment is an $\neq$-assignment. Thus from part (b) it follows that $\neq$ SAT is **NP-Complete**.

Problem 6

Clearly MAX-CUT is in **NP**. One can guess the partition of the graph into two parts and verify that the number of edges cut is at least k.

Now we show that MAX-CUT is **NP-complete** by showing that $\neq$ SAT $\leq_P$ MAX-CUT.

Let n be the number of variables and c be the number of clauses in the $\neq$ SAT instance ϕ. Follow the construction as described in the book. Let G be the resulting graph. For every literal z, G contains $3c$ nodes each labeled as z (let's call this "block" of nodes corresponding to z). Add all $(3c)^2$ edges between the block of z and block of $\overline{z}$. For every clause, there is a triangle between three nodes that are labeled by the three literals that appear in that clause. We do not use the same node in a block for more than one clause triangles.

Note that G now has $6cn$ nodes and $(3c)^2n + 3c$ edges. Set $k = (3c)^2n + 2c$.

We show that $\neq$ SAT has a $\neq$-assignment iff G has a cut of size at least k.

For the forward direction, take a $\neq$-assignment and place all nodes labeled by a TRUE literal on one side of the cut and all nodes labeled by a FALSE literal on the other side of the cut. This cuts all $(3c)^2n$ edges between the blocks. Also, since every clause gets a TRUE and a FALSE literal, for every triangle, two of the three edges are cut. Thus overall $(3c)^2n + 2c$ edges are cut.

For the backward direction, prove that any partition that cuts at least k edges must (1) place every block on one side of the partition entirely (2) place blocks corresponding to complementary literals on opposite sides (3) therefore the partition defines an assignment to literals (4) and then every clause must have a TRUE as well as a FALSE literal so that two edges in that clause triangle get cut.

[7] Note that in this case, the literal y_3 must be satisfied.