Mahya Jamshidian
9525133

Homework 1

Problem 1

a) $L_1 = \{\ S_1\#S_2\#S_3 — S_1\ ,\ S_2\ ,\ S_3 \in \{0,1\}^*\ ,\ S_3$ is binary addition of $S_1\ ,\ S_2\}$

W.l.o.g. we assume S_1 , S_2 and S_3 least significant bits are at right-most cell and $|S_2| > |S_1|$. We can also assume S_1 , S_2 and S_3 have no leading zeros. The procedure given below is operated on one-tape TM with both read and write access. Inital State = q_0, head in Δ, TM M repeats steps below and changes its states according to given table untill it reaches one of the halt(accept or reject) states.

1. Read below head, put $\sqcup$ instead and change state on read cell and the table. Move R untill first # symbol.
2. Move R untill first Non-# symbol. read below head , put # instead and change state on read cell and the table.
3. Move R untill first # symbol and Move R untill first Non-# symbol. read below head and put # instead and change state on read cell and the table. Only move on to 4 if TM hasn't halted yet.
4. Move L untill first Non-# symbol. read below head; if read cell is $\sqcup$, Go to 5 ; O.W Go to 6.
5. Move R untill first Non-# symbol; read below head; if read cell is $\sqcup$, then change state to q_{accept} ,else change state to q_{reject}.
6. Move L untill first Non-# symbol; read below head; if read cell is $\sqcup$ then Go to 7; else Move L untill fist $\sqcup$ then Move R. Go to 1.
7. if current state is q_0 then change state to $q_{0,0}$; else change state to then $q_{1,0}$; then Go to 2.

state	*step*1	*state*	*step*2	*state*	*stpe*3	*state*
q_0	0	$q_{0,0}$	0	$q_{0,0,0}$	0	q_0
q_0	0	$q_{0,0}$	0	$q_{0,0,0}$	1	q_{reject}
q_0	0	$q_{0,0}$	1	$q_{0,0,1}$	0	q_{reject}
q_0	0	$q_{0,0}$	1	$q_{0,0,1}$	1	q_0
q_0	1	$q_{0,1}$	0	$q_{0,1,0}$	0	q_{reject}
q_0	1	$q_{0,1}$	0	$q_{0,1,0}$	1	q_0
q_0	1	$q_{0,1}$	1	$q_{0,1,1}$	0	q_1
q_0	1	$q_{0,1}$	1	$q_{0,1,1}$	1	q_{reject}
q_1	0	$q_{1,0}$	0	$q_{1,0,0}$	0	q_{reject}
q_1	0	$q_{1,0}$	0	$q_{1,0,0}$	1	q_0
q_1	0	$q_{1,0}$	1	$q_{1,0,1}$	0	q_0
q_1	0	$q_{1,0}$	1	$q_{1,0,1}$	1	q_{reject}
q_1	1	$q_{1,1}$	0	$q_{1,1,0}$	0	q_0
q_1	1	$q_{1,1}$	0	$q_{1,1,0}$	1	q_{reject}
q_1	1	$q_{1,1}$	1	$q_{1,1,1}$	0	q_{reject}
q_1	1	$q_{1,1}$	1	$q_{1,1,1}$	1	q_1

b) $L_2 = \{\ a^i b^j c^k$ — $i * j = k$ and $i, j, k > 1\ \}$

$\Sigma = \{$ a,b,c,d,# , $\sqcup$, Δ $\}$

1. Move R. Read below head; put $\sqcup$ instead. Move R untill first Non-a symbol; if read cell is b, put d instead; then Go to 2; else Go to 4.
2. Move R untill first Non-b and Non-# symbol. read below head. if read cell is c put # instead and Go to; else if it is $\sqcup$, Go to 6.
3. Move L untill first Non-# symbol. read below head. if it is b, Move L untill first d, Move R and put d under head and Go to 2. else if it is d, Move L untill first $\sqcup$; Move R and Go to 1.
4. Move R untill first Non-d symbol; read below head. if read cell is c, put # instead and Go to 5; else if it is $\sqcup$, Go to 6.
5. Move L untill first Non-# symbol; read below head; if read cell is d, Move L untill first b; Move R and put b under head and Go to 4; else if it is b, Move L untill first $\sqcup$; Move R and Go to 1.
6. Move L untill first Non-#; read below head. if first read cell is b, then Move L untill first Non-b symbol; read below head; if second read cell is $\sqcup$ change state to q_{accept}; else change state to q_{reject}. else if first read cell is d, then Move L untill first Non-d symbol; read below head; if second read cell is $\sqcup$ change state to q_{accept}; else change state to q_{reject}.

Problem 2

(i) Design $M_1(\llcorner n \lrcorner_2) = \llcorner n \lrcorner_1$

least significant bit of $\llcorner n \lrcorner_2$ is at its leftmost cell. M has 2 tapes: one input/work tape and one output tape. head of both tapes are on their leftmost cell. Initial state = g_0

1. Move R; read below $head_1$; If read cell is *0*, Move R untill fist Non-*0* symbol; read below head; if read cell is $\sqcup$, change state to q_{halt}; else Move L untill Δ then Move R.
2. Move R for $head_2$; write 1;
3. read below $head_1$:
   - if read cell is *0* and state is q_0 : write *1* ; change state to q_0; Move R and Go to 3;
   - if read cell is *1* and state is q_0 : write *0* ; change state to q_1; Move R and Go to 3;
   - if read cell is *0* and state is q_1 : write *1* ; change state to q_0; Move R and Go to 3;
   - if read cell is *1* and state is q_1 : write *1* ; change state to q_1; Move R and Go to 3;
   - if read cell is $\sqcup$, change state to q_0; Move L untill Δ and Go to 1;

(ii) Design M_2 for L = $\{\ \sigma \in \{0,1\}^* : |\sigma|$ is prime$\}$

M_2 has 2 tapes. one for input/work and one work. both heads on Δ.

1. Change all *0*s in first tape to 1 and put head on first *1*.
2. Move $head_1$ R; Read below head; If read cell is $\sqcup$, then change state to q_{reject}; else Move $head_1$ L.
3. Put *11* on second tape and put head on first *1*.
4. Change *1* to *0* under both heads; then read below heads:
   - if under $head_1$ is $\sqcup$ and under $head_2$ is $\sqcup$, change state to q_{reject};
   - if under $head_1$ is $\sqcup$ and under $head_2$ is not $\sqcup$, Go to 5;
   - if under $head_1$ is not $\sqcup$ and under $head_2$ is $\sqcup$, Go to 5
5. change all the *0*s on both tapes to *1* and put another *1* at the end of second tape. put both heads on first 1;
6. If $tape_1$ and $tape_2$ are equal then change state to q_{accept}; else put both heads on first *1* on each tape and Go to 4;

(iii) Design an M_3 for PRIMES = $\{\sigma \in \{0,1\}^* \mid \sigma \text{ } is \text{ } prime \text{ } \}$
$M_3(\sigma) = M_2(M_1(\sigma))$

Problem 3

Design a NDTM M for COMPOSITE = $\{\sigma \in \{1\}^* \mid \mid \sigma \mid is \text{ } composite \text{ } \}$
let the certificate of this language to be a pair (x,y) s.t x.y = z and (x,y) , z are the inputs to M. let the certificate input to be in {0,1}*; $Mhasoneinputtapeforz \in$
$\{1\}^* andoneinputtapeforcertificate(x, y) \in$
{0,1*$(becauseoftwotransitionfunctionandNondeterministicMachine) and 3worktape.$

1. The procedure given below verifies the certificate for being exactly 2 number and non-zero numbers.
   (a) we map the certificate into alphabet {0,1,#} and write it on first work tape.
   (b) we check if there is exactly one # symbol on first worktape; O.W. chane state to q_{reject}.
   (c) we check if the # symbol is not on the fist cell or last Non-blank cell(non-zero input - first check); O.W. chane state to q_{reject}.
   (d) we check if first number and second number have at least one 1 symbol(non-zero input - second check); O.W. chane state to q_{reject}.
   (e) we map these binary representation inti alphaber 1 using the method explained in Problem 2 (i) and write first number unary representation on second worktape and second number unary representation on third worktape.

2. The procedure given below checks if the given certificate is halting on yes.

3. we write the unary representation on input(z) on firs work tape.

4. check if z has atleast length of 2; O.W. change state to q_{reject}.
   (a) Put all three heads of worktapes on first 1.
   (b) Move $head_{\text{W3}}$ R.
   (c) Move $head_{\text{W1}}$ and $head_{\text{W2}}$ R. If:
      - below $head_{\text{W1}}$ is $\sqcup$ and (below $head_{\text{W2}}$ is not $\sqcup$ or below $head_{\text{W3}}$ is not $\sqcup$) : change state to q_{reject}
      - below $head_{\text{W1}}$ is not $\sqcup$ and below $head_{\text{W3}}$ is $\sqcup$: change state to q_{reject}
      - below $head_{\text{W1,3}}$ is not $\sqcup$ and below $head_{\text{W2}}$ is $\sqcup$: Put $head_{\text{W2}}$ on first 1; Go to b.
      - below $head_{\text{W1,2,3}}$ is $\sqcup$: change state to q_{accept}
      - below $head_{\text{W1,2,3}}$ is not $\sqcup$: Go to c.

It is obvious that if such a pair exists, Our NDTM will accept the input and if no such a pair exists, it will reject on all possible certificates.

Problem 4

W.l.o.g, we can assume that M' that is the non-oblivious TM deciding L has One tape and running time of $T'(n)$. Now we design M with 3 tapes : one input tape, one tape for keeping i and one Counter. first tape has one additional symbol $\hat{a}$ for every symbol a in alphabet wich marks the head of M' (we are simulating M' using M);

- at first $\hat{}$ is on first cell of first tape. The second tape is equal to one and the third tape is equal to zero.

- To simulate step i of the M', M makes to pass from cell 1 to i and back to cell 1. it uses tape three as a counter to know when to turn around.

- on the way left, M $head_1$ reads below head of M' and on the way back it operates according to M' transition function. even if the M' is to move R, we set the leftward sweep of M on first tape to be two consecutive L and one R.

- after going back to cell #1, the number on second tape in incremented.

- Note that number on the third tape is incremented on the way to ith position and decremented on the way back.

- Clearly M will simulate M', and clearly M is oblivious; because the head movements are governed solely by the number i. Simulating step i of M' takes $O(i)$ time, so the total time to simulate all $T'(n)$ steps is $(T'(n))^2$, still a polynomial.

Problem 5

Prove that PRIMES NP; using : A number n is prime iff for every prime factor q of $n-1$, there exists a number $a \in \{2, ..., n-1\}$ satisfying $a^{\text{n-1}} = 1\ (modn)$ but $a^{(\text{n-1})/\text{q}} \neq 1\ (modn)$. Use this fact and induction to guess certificates for prime factors of n - 1.

Given such an a and the prime factorization of $n-1$, it's simple to verify the above conditions quickly: we only need to do a linear number of modular exponentiations, since every integer has fewer prime factors than bits, and each of these can be done in polynomial.

However, it is possible to trick a verifier into accepting a composite number by giving it a "prime factorization" of $n1$ that includes composite numbers.

by setting $n = 85$ and $a = 4$ and $q = 14, 6$:

- 4 is coprime to 85 : $4^{85\,-\,1} = 1(mod85)$
- $4^{(85\,-\,1)/14} = 16(mod85)$ and $4^{(85\,-\,1)/6} = 16(mod85)$
- therefore 85 is prime.

A solution to this issue is to give primality certificates for each of the prime factors of $n-1$ as well, which are just smaller instances of the original problem. We continue recursively in this manner until we reach a number known to be prime, such as 2. We end up with a tree of prime numbers, each associated with an a.

example, $n = 23$ and $a = 6\ 28 = 2.11$

- 2 is known prime
- $n = 11$ and $a = 10\ 10 = 2.5$
  - 2 is known prime
  - $n = 5$ and $5 = 2^2$
    - 2 is known prime

Proof by induction can show that this proof three has at most $4\,\text{logn}$ values. we set base to be greater than 3 and :

- for p > base and $p_1\ p_2\ p_3\ ...\ p_k = p - 1$.
- each p_i as root has at most $4 \log p_i$ children.
- $\Sigma\ _{\text{i= 1:k}}\ 4\text{log}\, p_i = 4\,\text{logp}_1\ p_2\ p_3\ ...\ p_k < 4 \log p$

(pratt 2nd theorem)

Problem 6

Let L be an NP-complete unary language. Since L is NP-complete, there is a polytime reduction f from SAT to L.

Since f is polytime, $|f(x)| \leq C|x|^d$ for some C,d. We will now describe a polytime algorithm for SAT.

Denote the input by ϕ, a formula on the n variables $x_1, ..., x_n$; we assume that $|\phi| \geq n$. The algorithm proceeds in n stages, creating a sequence of lists $L_n, ..., L_0$. The list L_k consists of a list of pairs $(f(\psi_i), \psi_i)$, where ψ_i is a formula resulting from substituting values for $x_{\text{k+1}}, ..., x_n$ in ϕ and simplifying. We maintain the invariant that ϕ is satisfiable if and only if one of the ψ_i is satisfiable.

The initial list L_n consists of the pair $(f(\phi), \phi)$. Given the list L_k, we construct the list $L_{\text{k-1}}$ in two steps:

1. For each f$(\psi), \psi) \in L_k$,add to $L_{\text{k-1}}$the two pairs(f$(\psi x_k = T)$,$\psi x_k = T)$ and $(f(\psi x_k = F)$,$\psi x_k = F)$; after substituting a value for x_k, we simplify the resulting formula.
2. For each m, out of all pairs of the form $(1^m, \psi)$ verify only one(if exists).

The final set L_0 could contain $(f(T), T)$ and $(f(F), F)$; the formula is satisfiable if and only if it contains the former.

Each list L_k has size at most $C|\phi|^d$, and so the algorithm runs in polynomial time(because of the bound on the size of fs output.).

Problem 7

We provide a chain of reductions from 3SAT to NAE 4-SAT (NAE is the Boolean operation that evaluates to true if and only if not all of its inputs are equal.) to NAE 3-SAT and to MAX-CUT.

We will give a polynomial-time algorithm A that given a 3-SAT instance constructs an equivalent NAE 4-SAT instance. Given a 3-SAT instance ϕ, the algorithm A constructs a NAE 4-SAT instance ϕ' = A(ϕ) by adding a variable z to every clauses. (The variable z is distinct from the variables that appear in ϕ.) For example, the 3-SAT clause $[x_1 \bigvee x_3 \bigvee \neg\, x_4]$. would be replaced by the NAE 4-SAT clause NAE(x_1, x_3, $\neg\, x_4$, z)

$$\text{3-SAT} \leq_{\text{p}} \text{NAE 4-SAT}$$

We are to show that ϕ is satisfiable if and only if ϕ is satisfiable. If x_1, . . . , x_n is a satisfying assignment for ϕ, then the same assignment satisfies ϕ when we choose z = 0.

The reason is

$$a \bigvee b \bigvee c = \text{NAE(a, b, c, 0)}$$

To show the other direction, suppose x_1, . . . , x_n, z is a satisfying assignment of ϕ. It is obvoius that ¬ x_1, . . . ,¬ x_n,¬z is also a satisfying assignment of ϕ (because NAE(a, b, c, d) = NAE(¬a,¬b,¬c,¬d)). In one of these two assignments, the value assigned to the variable z is 0. This assignment corresponds to a satisfying assignment for ϕ.

$$\text{NAE 4-SAT} \leq_{\text{p}} \text{NAE 3-SAT}$$

it is obvious that by a 4-SAT (a, b, c, d) we can construct a 3-SAT in polynomial time. we put two of the literals in each clause C_i into another clause with on additional literal w_i s.t. $NAE(a, b, w_i)$ = $NAE(c, d, \neg w_i)$ and two generated clause for 3-SAT are (a, b, w_i) and $(c, d, \neg w_i)$.

$$\text{NAE 3-SAT} \leq_{\text{p}} \text{MAX CUT}$$

Given a NAE 3-SAT instance ϕ, we will construct an equivalent MAX-CUT instance (G, c). For every variable x_i of ϕ, we will add two vertices to G labeled by G_i and $\neg x_i$ and we will connect the two vertices by an edge. We assign capacity M = 10 * m to each of these variable edges. (Here, m is the number of clauses in ϕ and n is the number of variabes.) For every clause C in ϕ, we will add a clause triangle between the vertices corresponding to the terms in C. We assign capacity 1 to each of these clause edges.

Suppose ϕ is satisfiable and consider any satisfying assignment. This assignment corresponds to a cut in G. (One side of the cut consists of all vertices labeled by terms that evaluate to 1 in the assignment. The other side of the cut consists of all vertices labeled by terms that evaluate to 0 in the assignment.) Since exactly one of terms x_i and $\neg x_i$ evalute to 1 in an assignment, all variable edges go across the cut, which contributes n * M to the capacity of the cut. Since the assignment satisfies ϕ, exactly two edges in every clause triangle go across the cut, which contributes 2 m to the capacity of the cut. In total the capacity of the cut is equal to n * M + 2 * m.

On the other, suppose that G contains a cut with capacity at least n * M + 2 * m. First, we claim that all variable edges go across this cut. The reason is that any cut that misses at least one of the variable edges has capacity at most (n - 1) * M + 3 * m = n * M + 3 * m - 10m, which is strictly smaller than n * M + 2m. Next, we claim that exactly two edges of every clause triangle go across the cut. The reason is that no cut can separate three edges of a triangle and therefore if a cut separates fewer than two edges in one of clause triangles, then its capacity is strictly smaller than n * M + 2m. Since all variable edge go across, this cut corresponds to an assignment for ϕ. Furthermore, since the cut separates exactly two edges per clause triangle, the corresponding assignment satisfies all clauses of ϕ.

Problem 8

NP = CoNP $\rightarrow$ 3SAT and $TAUTOLOGY$ is polynomial-time reducible to one another.

- SAT and 3SAT are polynomial-time reducible to each other
- we know $SAT \in NP \rightarrow \overline{SAT} \in CoNP$
- $NP = CoNP \rightarrow \overline{SAT} \in NP$
- $\overline{TAUT} \in NP$: certificate is the assignment and we are looking for one assignment that makes TAUT(ψ) = 0 $\rightarrow TAUT \in CoNP$
- $TAUT \in NP$
- SAT is $NP - Complete \rightarrow \overline{TAUT} \leq_{\text{ p}} SAT$
- it is easy to show that $SAT \leq_{\text{ p}} \overline{TAUT}$: a CNF like ψ is satisfiable iff its complement is not a tautology.
- CNF ψ is satisfiable $\rightarrow$ there exists an assignment for its variable which makes it True and that assignment makes its complement False (it is not a tautology)
- $\overline{\psi}$ is not a tautology $\rightarrow$ there exists an assignment for its variable which makes it False and that assignment makes its complement True (it is satisfiable)
- now that NP = CoNP and TAUT is CoNP-Complete ($\overline{TAUT}$ is NP-Complete) both SAT and TAUT are NP-Complete and therefore polynomial-time reducibel to one another.

3SAT and $TAUTOLOGY$ is polynomial-time reducible to one another $\rightarrow$ NP = CoNP

- we showed that TAUT is CoNP-Complete and we know SAT is NP-Complete.
- if TAUT and SAT are polynomial-time reducible to one another, then every Language in CoNP is polynomial-time reducible to SAT and every Language in NP is polynomial-time reducible to TAUT
- by defenition, we conclude that NP=CoNP

Problem 9

In any sufficiently powerful axiomatic system, for any input α and x we can write a mathematical statement $\psi(\alpha, x)$ that is true iff $HALT(\alpha$,x) = 1. Now if the system is complete, it must prove at least one of $\psi(\alpha$x) or $\overline{\psi(\alpha}$x), and if it is sound it cannot prove both. So if the system is both complete and sound, the following algorithm for the Halting problem is guaranteed to terminate in finite time for all inputs.

Given input (α, x), start enumerating all strings of finite length, and check for each generated string whether it represents a proof in the axiomatic system for either $\psi(\alpha$x) or $\overline{\psi(\alpha\text{x})}$. If one waits long enough, a proof of one of the two statements will appear in the enumeration, at which point the correct answer 1 or 0 is revealed, which you then output.

Note that this procedure implicitly uses the simple fact that proofs in axiomatic systems can be easily verified by a Turing machine, since each step in the proof has to follow mechanically from previous steps by applying the axioms.

Now we sketch the construction of the desired statement $\psi(\alpha$x). Assume the axiomatic system has the ability to express statements about the natural number using the operators plus (+) and times (), equality and comparison relations $(=, >, <)$, and logical operators such as $AND(\bigwedge)$, $OR(\bigvee)$, and $NOT(\neg)$.

The language also includes the quantifiers for-all ($\forall$) and exists ($\exists$) and the constant 1 (we can get any other constant c by adding 1 to itself c times). For example, the formal expression for x divides y will be $DIVIDES(x, y) = \exists k : y = xk$, and the expression for y is prime will be $PRIME(y) = \forall x(x = 1)(x = y)DIVIDES(x, y)$ (where $DIVIDES(x, y)$ is shorthand for the corresponding expression).

We can encode strings (and hence also Turing machines and their inputs and tapes) as numbers. Then one notes that a basic operation of the Turing machine only influences one (or a few, if the machine has mutiple tapes) of bits on its tape, which can be viewed as a simple arithmetic operation on the string/number representing the tape contents.

With some work one obtains an expression $\psi(\alpha$x)(t) that is true if and only if the TM M halts on input x within t steps. Hence, Mα halts on x if and only if $\exists$t *s.t.* $\psi(\alpha$x)(t) is true, which is the desired mathematical statement.