

Problem Set #4

- Rules of the game remain the same. Submissions must be single file in L^AT_EX format at the upload link set up in cse moodle page of the course.

1. (15 points) Argue whether the following functions qualify to be called as a *resource* according to Blum's resource axioms.

$$(a) \text{ (3 points) } \text{VALUE}(M, x) = \begin{cases} 0 & \text{if } M \text{ rejects } x, \\ 1 & \text{if } M \text{ accepts } x, \\ \text{undefined} & \text{if } M \text{ does not halt on } x \end{cases}$$

Answer

The above resource function satisfies Blum's first axiom i.e. it is defined only when M halts on x . However, given a M, x, k , it is not decidable to check if $\text{VALUE}(M, x) = k$. Hence it is not a resource.

Consider $R = \{(M, x, k) : \text{VALUE}(M, x) = k\}$

Suppose R was decidable via a total TM N , then MP can be decided by giving $(M, x, 1)$ as input to N . But, MP is not decidable and hence R is not decidable.

- (b) (5 points) A *turn* of tape head is defined as movement of tape head from $L \rightarrow R$ or $R \rightarrow L$.

$$\text{TURN}(M, x) = \begin{cases} \text{No. of turns that tape head makes when } M \text{ runs on } x & \text{if } M \text{ halts on } x \\ \text{undefined} & \text{otherwise} \end{cases}$$

Answer

The above resource function satisfies Blum's first axiom i.e. it is defined only when M halts on x . The above resource also satisfies Blum's second axiom, i.e. $R = \{(M, x, k) : \text{TURN}(M, x) = k\}$ is a decidable set.

Consider a total TM N corresponding to R . Let the description of this machine be as follows:

- Run M on x .
- If the machine makes more than k turns, reject and halt.
- Else, machine M will halt on x and hence if the number of turns made by the machine is k , accept. Else, reject.

Claim

Language accepted by a machine which makes finite number of turns is regular.

This is because, the crossing sequence at every index for this machine will be bounded by a fixed constant s , and from class we saw such machines accept only

regular languages.

From the claim above, the machine N will always halt and hence is total.

$$(c) \text{ (5 points) } \text{COUNT}(M, x) = \begin{cases} \text{No. of times } M \text{ visits a state } q & \text{if } M \text{ halts on } x \\ \text{undefined} & \text{otherwise} \end{cases}$$

Answer

The above resource function satisfies Blums first axiom i.e. it is defined only when M halts on x . However, given a M, x, k , it is not decidable to check if $\text{COUNT}(M, x) = k$. Hence it is not a resource.

Consider $R = \{(M, x, k) : \text{COUNT}(M, x) = k\}$

Suppose R was decidable, then we can use it as a sub routine to decide the HP as follows :

Construct a machine N as follows:

- Simulate M on x .
- If M halts on x , then have a transition to a special accept state.

This gives the reduction from R to HP .

2. (5 points) Show that if $\text{NTIME}(n) = \text{DTIME}(n)$ then $P = NP$. (Padding !!)

Answer

Let $\text{NTIME}(n) = \text{DTIME}(n)$.

Let $L \in NP$ via a non-deterministic machine M .

Consider $L_{pad} = \{x\#1^{|x|^c} : x \in L\}$

Consider a N which does the following on input y :

- Check if $y = x\#1^{|x|^c}$.
- Extract x
- Run machine M on x , to check if $x \in L$. If yes, accept. Else, reject.

The running time of the above machine is linear in its input size. Therefore, $L_{pad} \in \text{NTIME}(n)$.

From assumption, $L_{pad} \in \text{DTIME}(n)$. Let the deterministic machine accepting this be N' .

Consider a machine M' which does the following:

- Construct $y = x\#1^{|x|^c}$
- Check if $y \in L_{pad}$. If yes, accept. Else, reject.

$L(M') = L$. The time taken by this deterministic machine is polynomial in its input length. Hence, $L \in P$.

3. (10 points) Space Hierarchy theorem implies the following: For any $k > 0$, There is a language in $\text{DSPACE}(n^{k+1})$ that is not in $\text{DSPACE}(n^k)$. Use this and a padding argument to show that: $P \neq \text{DSPACE}(n)$. (6pts)
(Note that we do not know the containment in either direction.)
You can do this in two steps:

- (a) For every language L define, $L_{pad} = \{x01^{|x|^2} : x \in L\}$.
Argue that L_{pad} is in $P \implies L \in P$.
- (b) Show an L_{pad} which is in $\text{DSPACE}(n)$ but whose corresponding L is not in $\text{DSPACE}(n)$.

Answer

(a) For every language L , let $L_{pad} = \{x01^{|x|^2} : x \in L\}$.

Let $L_{pad} \in P$ via a deterministic machine M running in polynomial time.

Given an input y , M does the following

- Checks if y is of the form $x01^{|x|^2}$.
- Extracts x out of it, and checks if $x \in L$.

Since, M runs in polynomial time, it does steps 1 and 2 in polynomial time each. Given an x , checking if $x \in L$ can be done in polynomial time by a deterministic machine. Hence, $L \in P$.

(b) From space hierarchy theorem, $\exists A$ such that, $A \in \text{DSPACE}(n^2)$ but $A \notin \text{DSPACE}(n)$. Consider $A_{pad} = \{x01^{|x|^2} : x \in A\}$

Consider a machine N accepting A_{pad} . It does the following

- Checks if the input y is of the form $x01^{|x|^2}$.
- It extracts x from the input and checks if $x \in A$.

This machine takes space $|x|^2$. That is $\text{DSPACE}(n)$. (Since input size is $|x|^2$). Hence, (b) is also true.

Now, it remains to show that (a) and (b) $\implies P \neq \text{DSPACE}(n)$.

This can be shown by contradiction. Suppose $P = \text{DSPACE}(n)$. Then $A_{pad} \in \text{DSPACE}(n) \implies A_{pad} \in P \implies A \in P \implies A \in \text{DSPACE}(n)$.

But this is a contradiction, because A belongs to $\text{DSPACE}(n^2)$ and not $\text{DSPACE}(n)$.

Therefore, (a) and (b) $\implies P \neq \text{DSPACE}(n)$.

4. (5 points) Show that if $\text{SAT} \in \text{NP} \cap \text{coNP}$ then $\text{NP} = \text{coNP}$. (Definitions !)

Answer

SAT is $\text{NP} - \text{Complete}$. Hence, for every language L in NP , there is a polynomial time reduction to SAT . And, $\text{SAT} \in \text{NP} \cap \text{coNP} \implies L \in \text{NP} \cap \text{coNP} \implies \text{Every Language in}$

NP belongs to $\text{NP} \cap \text{coNP} \implies$ Every Language in NP belongs to coNP ... (1)

Consider a language L' in coNP $\implies \overline{L'} \in \text{NP} \implies \overline{L'} \in \text{NP} \cap \text{coNP} \implies \overline{L'} \in \text{coNP} \implies L' \in \text{NP}$.. (2)

From (1) and (2), $\text{NP} = \text{coNP}$.

5. (5 points) If L, L' are in NP, then show that $L \cup L', L \cap L'$ are in NP. (Definitions !)

Answer

From defn, $L \in \text{NP} \Leftrightarrow$ there exists a polynomial length certificate c_1 which can be verified in polynomial time via a machine M_1 .

Similarly, $L' \in \text{NP} \Leftrightarrow$ there exists a polynomial length certificate c_2 which can be verified in polynomial time via a machine M_2 .

$L \cap L'$, construct a new certificate $c_1 \# c_2$. This is of polynomial size. To verify if $x \in L \cap L'$, we need to construct a polynomial time running machine which verifies c_1 and c_2 . Construct a new machine N which simulates first simulates M_1 on input c_1 and verifies if its the correct certificate of L . If yes, it proceeds to simulating M_2 on c_2 . Else rejects. Similarly if c_2 is verified as the correct certificate of L' by simulation of M_2 then it will accept. Else reject. Since simulation of both M_1 and M_2 takes polynomial time in the input length, hence N also runs in polynomial time.

$L \cup L'$, construct a new certificate $c_1 \# c_2$. This is of polynomial size. To verify if $x \in L \cup L'$, we need to construct a polynomial time running machine which verifies c_1 and c_2 . Construct a new machine N which simulates first simulates M_1 on input c_1 and verifies if its the correct certificate of L . If no, it proceeds to simulating M_2 on c_2 . Else accepts. If c_2 is verified as the correct certificate of L' by simulation of M_2 then it will accept. Else reject. Since simulation of both M_1 and M_2 takes polynomial time in the input length, hence N also runs in polynomial time.

6. (5 points) If L, L' are in $\text{NP} \cap \text{coNP}$, then show that $L \oplus L'$ defined as

$$L \oplus L' = \{x : x \text{ is in one of } L \text{ or } L' \text{ but not both.}\}$$

is in $\text{NP} \cap \text{coNP}$. (Definitions !)

Answer

If $L \in \text{NP} \cap \text{coNP}$ implies $\overline{L} \in \text{NP}$. Similarly, $\overline{L'} \in \text{NP}$.

Let c_1, c_2, c_3, c_4 be the polynomial size certificates verifiable in polynomial time corresponding to language $L, \overline{L}, L', \overline{L'}$ respectively.

Similar to the construction above we have to check if c_1 and c_3 are the correct certificates or c_2 and c_4 are correct certificates for the respectively languages $L \cap \overline{L'}$ and $\overline{L} \cap L'$. Hence, $L \oplus L' \in \text{NP}$.

Similarly, we have to check if c_1 and c_4 are the correct certificates or c_2 and c_3 are correct certificates for the respectively languages $\overline{L} \cap \overline{L'}$ and $L \cap L'$. Hence, $L \oplus L' \in \text{coNP}$.

Therefore, $L \oplus L' \in \text{NP} \cap \text{coNP}$

7. (15 points) Consider the following language: $\text{PRIMES} = \{n \mid n \text{ is a prime}\}$ where the input n is in binary. Without using the known result that PRIMES is in P, solve the following:

- (a) (5 points) Show that PRIMES is in coNP.

Answer

To show that $\text{PRIMES} \in \text{coNP}$, we have to show that $\overline{\text{PRIMES}} \in \text{NP}$.

$\overline{\text{PRIMES}} = \{n \mid \exists p, q \text{ s.t. } p, q \neq 1 \text{ and } p * q = n\}$

Given p, q as a certificate (each of them need $\log n$ bits respectively and hence polynomial in the size of input n), we can verify if $p * q = n$ in polynomial time as follows:

Calculate n/p using long-division. Since each step of long division eliminates one digit from the dividend (and every step is polynomial in dividend length), hence it runs for polynomial time in the length of dividend. Now check if the obtained quotient is equal to q and the obtained remainder is equal to 0 (This can be done in linear time with respect to length of q).

Therefore, $\overline{\text{PRIMES}} \in \text{NP}$ and hence $\text{PRIMES} \in \text{coNP}$.

- (b) (10 points) Here is Lucas test for primality (you don't need prove it) : n is prime if and only if there is an integer $a \in \{2, \dots, n-1\}$ with $a^{n-1} \equiv 1 \pmod n$, and for every prime factor q of $n-1$: $a^{\frac{n-1}{q}} \not\equiv 1 \pmod n$. Use this test to show that PRIMES is in NP.

Answer

Given a and the prime factorization of $n-1$ as a certificate, it is easy to verify the above condition of Lucas in $O(\log^3(n))$ time. Modular exponentiation takes $O(\log n)$ time and at each stage multiplication takes $O(\log^2(n))$ steps. Now we have to show that $|a| + |\text{prime factorization of } n-1|$ is a polynomial in $\log n$.

Clearly, since $a < n$, therefore it can be represented using $\log(n)$ bits. Now we have to check

- 1) prime factorization of $n-1$ can be represented in poly in $\log(n)$ bits.
- 2) Each of the prime factors should be further verified as a prime.

Let $n-1$ be factorized as $p_1^{a_1} * p_2^{a_2} * \dots * p_k^{a_k}$.

This product is greater than $2^{a_1} * 2^{a_2} * \dots * 2^{a_k}$. Therefore, $a_1 + a_2 + \dots + a_k < \log(n)$. Implies the number of prime factors of $n-1$ is utmost $\log(n)$.

For each of p_1, p_2, \dots, p_k , there will be another set of certificates. Each of these certificates is a polynomial in $\log(n)$ and verifiable in polynomial in $\log(n)$ (From strong

induction, because each of p_1, p_2, \dots, p_k is strictly smaller than $n-1$). Hence, the total time for verifying the prime factorization for $n-1$ is (polynomial in $\log(n)$) * (polynomial in $\log(n)$), which is polynomial in $\log(n)$.

And since each of the prime factors can be $\log(n)$ length and there are $\log(n)$ of them, the total length is a polynomial in $\log(n)$.

Therefore, the certificate a , prime factors of $(n-1)$ is a polynomial size in input length verifiable in polynomial time in input length. Therefore, PRIMES \in NP.

Hence conclude that PRIMES is in $\text{NP} \cap \text{coNP}$.

From (a),(b), PRIMES is in $\text{NP} \cap \text{coNP}$.

8. (10 points) Prove that reachability in undirected forests (a possibly disconnected acyclic undirected graph) can be solved in log-space. That is, given (T, s, t) where T is an undirected forest, it can be tested in log-space whether s is connected to t by a path.
9. (18 points) Let $E = \bigcup_{c \geq 0} \text{DTIME}(2^{cn})$ and $NE = \bigcup_{c \geq 0} \text{NTIME}(2^{cn})$. A set A is called *sparse* if there is a polynomial p , such that $|\{x \in A : |x| = n\}| \leq p(n)$. A set A is called *tally set* if $A \subseteq \{1\}^*$. Prove that following are equivalent.
 1. Restricted to tally sets $\text{NP} = \text{P}$. That is all tally sets in NP are in P .
 2. Restricted to sparse sets $\text{NP} = \text{P}$. That is all sparse sets in NP are in P .
 3. $E = NE$

Hence conclude that $E \neq NE \implies \text{P} \neq \text{NP}$.

Hint : Try for $(b) \implies (a) \implies (c) \implies (b)$. For the second implication : consider the language $L_{\text{tally}} = \{1^{2^{|x|}} : x \in L\}$. This will not work, but a slight modification of this language which includes some more information about x will work !.

For the third implication, consider the language

$$L_{\text{order}} = \{(k, i, c) : \text{the } i^{\text{th}} \text{ bit of the } k^{\text{th}} \text{ string (in lex order) in } L \text{ is } c\}$$

10. (7 points) Imagine a world in which $\text{P} = \text{NP}$. Now show that there is a polynomial time algorithm which given a Boolean formula ϕ produces a satisfying assignment for ϕ if ϕ is satisfiable. (Hint : Use queries to SAT).

Answer

We can get the assignments for x_1, x_2, \dots, x_n in $n+1$ queries to SAT as follows.

- Firstly, make a query to SAT to find if ϕ is satisfiable. If no, immediately return.
- If ϕ is satisfiable, construct new expressions, $\phi_1, \phi_2, \dots, \phi_n$, where, $\phi_i = \phi \wedge x_i$.
If, ϕ_i is satisfiable assign $x_i = 1$. Else, assign $x_i = 0$.

Since, $\text{P} = \text{NP}$, therefore there is a polynomial time algorithm to SAT. And since, the number of queries to SAT is in $O(n)$, therefore, above algorithm runs in polynomial time.