CS 332 – Spring 2021, Assignment 9

Colaborators: None

Answer 1

(a) There is no language satisfying this property. If I can poly-reduce A to SAT, and since we know from class that $SAT \in NP$, then I know that I can reduce A to SAT in polynomial time, and then utilize the polynomial time verifier for SAT on the resulting mapping, thus being able to polynomially verify A, which would mean that $A \in NP$, regardless of the actual language of A.

(b) There is no language satisfying this property. We know that SAT is both $NP - Hard$, and by the definition of $NP - Hard$, $NP - Complete$. From here, we know that, according to theorem 7.36 in Sipser, that if $SAT \leq_p B$ and $SAT \in NP - Complete$, then B must be $NP - Complete$, if $B \in NP$ *(this is true because we know that every problem in $NP - Complete$ is reducible to SAT, which in turn is reducible to B, thus meaning that every problem in $NP - Complete$ is reducible to B, as long as $B \in NP$).*

Thankfully, we don't actually need to prove this last part to establish our goal of proving that $B \in NP - Hard$. By definition, all problems in $NP - Hard$ are at least as difficult as the hardest problems in NP *(ie: the $NP - Complete$ problems).* Since we already know that $SAT \leq_p B$ and that $SAT \in NP - Complete$, then we can never have that B is "easier" to solve than SAT, otherwise $SAT \in P$, something we know not to be true since $NP \neq P$. Thus, if B is equally hard as SAT, then $B \in NP - Complete$, otherwise it'll be harder, which means $B \in NP - Hard$. However, we also know that every problem in $NP - Complete$, then regardless of the "difficulty" of B, $B \in NP - Hard$.

(c) The Halting Problem is an example of a language C where $SAT \leq_p C$ and $C \notin NP - Complete$. As per theorem 7.36 in Sipser, if $SAT \leq_p C$, then since $SAT \in NP - Complete$, then $C \in NP - Complete$ if $C \in NP$. However, we know that the Halting Problem is not decidable in polynomial time, nor can it be verified with a polynomial time verifier, thus it isn't in NP. You can also reduce an instance of SAT to an instance of the Halting Problem in polynomial time by transforming it into a description of a Turing Machine which tries all truth value assignments and halts if it finds one that satisfies the formula, looping infinitely otherwise. Thus, we have that $SAT \leq_p C$, and $C \notin NP - Complete$ *(but $C \in NP - Hard$)* if $C = Halting$.

(d) There are no languages that satisfy this properties. We know from theorem 7.16 of Sipser that all context-free languages are in P, thus all regular languages are also in P. However, we know that if a language $A \in NP - Complete$, and $A \in P$, then $NP = P$. Since we know that $NP \neq P$, then if P is regular, then $P \notin NP - Complete$.

Answer 2

(a) $(i) =$ a possible list T of foods.

(b) $\langle S_1, ..., S_n, k \rangle$

Answer 3

We know that $SAT \in NP - Complete$, furthermore, we know from Theorem 7.36 of Sipser, that if $SAT \leq_p XSAT$ then $XSAT \in NP - Complete$ if and only if $XSAT \in NP$. We already know that $XSAT \in NP$, so we are left with proving that $SAT \leq_p XSAT$.

To reduce SAT to $XSAT$, we can use the TM below:

"$M =$ On input $\langle \varphi_0 \rangle$:

1. Build φ_1 and φ_2, with $\varphi_1 = \varphi_0$ and $\varphi_2 = 0$.
2. Return $\langle \varphi_1, \varphi_2 \rangle$."

This algorithm will run in polynomial time, since for an input φ_0 of size n, it takes exactly $O(n)$ to build φ_1 and φ_2 and return them. Furthermore, since φ_2 will always be unstasfiable, then if when running $XSAT$ on input $\langle \varphi_1, \varphi_2 \rangle$, we get an "accept", then we know that there must be some set of conditions that satisfy φ_1, thus also satisfying φ_0, and solving SAT.

As such, we now know that $SAT \leq_p XSAT$, thus proving that $XSAT \in NP - Complete$.

Answer 4

(a) $BROOT$ has a poly-time deterministic verifier V, which when taking $\langle p, c\rangle$, returns whether or not $p \in BROOT$. Here we have that our certificate c will be a set of assignments $(b_1, ..., b_n) \in \{0, 1\}^n$. With this input, we have that V is as below:

"$V =$ On input $\langle p, c\rangle$:

1. Solve $p(c_1, ..., c_n)$.
2. If the solution is equal to 0, accept, else reject."

Since this verifier runs in polynomial time, then $BROOT \in NP$.

(b) To transform an instance of $3SAT$ into an instance of $BROOT$, we can use the TM below:

"$T =$ On input $\langle\varphi\rangle$:

1. Split φ into its n different $(u \vee v \vee w)$ clauses *(let's call each of these c_i)*.
2. For each c_i, create the polynomial $p_i = p(u, v, w) = (1 - v) \times (1 - w) \times (1 + u) + (1 - u) \times (1 - w) \times (1 + v) + (1 - u) \times (1 - v) \times (1 + w)$.
3. Combine all p_i into $p_{final} = p(p_1, ..., p_n) = \sum_{i=1}^{n} x_i$.
4. Return p_{final}."

Since each p_i is already a polynomial, then the addition of the n polynomials, will also be a polynomial, which can then be ran through $BROOT$. Furthermore, T itself runs in polynomial time since all it's doing is for an input of size n, it constructs a polynomial, which runs in linear time, and then it's going through all n polynomials, and adding them together, which would take $O(n)$ time. Finally, we can say that if $BROOT$ accepts, then we know that there is some combination of assignments that satisfies $3SAT$, thus $3SAT \leq_p BROOT$.