Problem Set 9 Solutions

Problem One: 4-Colorability (24 Points)

i. Prove that *4COLOR* ∈ **NP** by designing a polynomial-time NTM for it. Prove that your NTM is correct, then justify informally why it runs in polynomial time.

Let M be the machine

> M = "On input $\langle G \rangle$, where G is a graph:
> Nondeterministically guess an assignment C of nodes to one of four colors.
> For each edge in G, check if the endpoints are assigned the same color by C.
> If any edge is assigned endpoints of the same color, reject.
> Otherwise, accept."

Note that there is some series of choices such that M accepts $\langle G \rangle$ iff there is a way to assign each node in G one of four colors such that no edge has endpoints which are the same color iff G is 4-colorable. Guessing such an assignment takes polynomial time, because we need to make one of four choices for each of the polynomially many nodes, and checking such an assignment takes polynomial time because there are only polynomially many checks to make, each of which takes at most polynomial time.

ii. Prove that *4COLOR* is **NP**-hard by proving *3COLOR* $\leq_P$ *4COLOR*. That is, show how to take an arbitrary graph G and construct (in polynomial time) a graph G' such that graph G is 3-colorable iff graph G' is 4-colorable.

For simplicity, you do not need to formally prove that your reduction is correct and runs in polynomial time. Instead, briefly answer each of the following questions about your reduction (two or three sentences apiece should be sufficient):

1. If the original graph G is 3-colorable, why is your new graph G' 4-colorable?
2. If your new graph G' is 4-colorable, why is the original graph G 3-colorable?
3. Why can your reduction be computed in polynomial time?

Let the graph G' be defined as follows: take G, add in a new node v, then add an edge from every node in G to v.

1. If G is 3-colorable, we can 4-color G as follows: color all the nodes in G the original colors, then color the new node the fourth color. Any edge between nodes either is between two nodes in G, in which case the old coloring ensures the endpoints have different colors, or is between a node in G and the new node v, in which case the endpoints can't have the same colors since v is given a unique color.
2. If G' is 4-colorable, first we claim that no node in G' besides v can be the same color as v. This is because there is an edge from every node in G to v. This means that of the four colors used in G', there must be one color that doesn't appear in any of the nodes in G, since it's reserved for v. Thus the remaining nodes in G must be colored with just three colors, so G is 3-colorable.
3. This can be computed in polynomial time since we add only one new node and polynomially many edges, each of which can be constructed in polynomial time.

Problem Two: $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ (32 Points)

1. There is a **P** language that can be decided in polynomial time.

Neither. All languages in **P**, by definition, can be decided in polynomial time.

2. There is an **NP** language that can be decided in polynomial time.

Neither. $\mathbf{P} \subseteq \mathbf{NP}$, so all languages in **P** are also languages in **NP**. Therefore, we already know of infinitely many languages in **NP** that can be decided in polynomial time.

3. There is an **NP**-***complete*** language that can be decided in polynomial time.

$\mathbf{P} = \mathbf{NP}$. As mentioned in lecture, if an **NP**-complete languages belongs to **P**, then $\mathbf{P} = \mathbf{NP}$.

4. There is an **NP**-***hard*** language that can be decided in polynomial time.

$\mathbf{P} = \mathbf{NP}$. Any **NP**-hard language that can be decided in polynomial time belongs to **P**, and therefore to **NP**. Consequently, an **NP**-hard language decidable in polynomial time is an **NP**-complete language decidable in polynomial time. Thus $\mathbf{P} = \mathbf{NP}$.

5. There is an **NP** language that ***cannot*** be decided in polynomial time.

$\mathbf{P} \neq \mathbf{NP}$. This would mean there is a language in $\mathbf{NP} - \mathbf{P}$, so $\mathbf{P} \neq \mathbf{NP}$.

6. There is an **NP**-***complete*** language that ***cannot*** be decided in polynomial time.

$\mathbf{P} \neq \mathbf{NP}$. This is just a special case of the above.

7. There is an **NP**-***hard*** language that ***cannot*** be decided in polynomial time.

Neither. A_{TM} is **NP**-hard. We can reduce any language L in **NP** to A_{TM} as follows: let M be a polynomial-time NTM for L and let $f(w) = \langle M, w \rangle$. This can be computed in polynomial time. However, $A_{TM} \notin \mathbf{R}$, so it can't be decided in polynomial time.

8. There is ***some*** **NP**-complete language that can be decided in $O(2^n)$ time.

Neither. This would not prove that an **NP**-complete language belongs to or does not belong to **P**.

9. There is ***no*** **NP**-complete language that can be decided in $O(2^n)$ time.

$\mathbf{P} \neq \mathbf{NP}$. All polynomials are $O(2^n)$, so if an **NP**-complete language cannot be decided in $O(2^n)$, then that **NP**-complete language is not in **P**. Thus $\mathbf{P} \neq \mathbf{NP}$.

10. There is a polynomial-time ***verifier*** for every language in **NP**.

Neither. This is the definition of **NP**.

11. There is a polynomial-time ***decider*** for every language in **NP**.

P = NP. This means every language in **NP** belongs to **P**, so **NP** $\subseteq$ **P** and therefore **P** = **NP**.

12. There is a language $L \in$ **P** where $L \leq_P$ 3SAT.

Neither. All languages in **NP** are polynomial-time reducible to all **NP**-complete languages, and 3SAT is **NP**-complete.

13. There is a language $L \in$ **NP** where $L \leq_P$ 3SAT.

Neither. Same as above.

14. There is a language $L \in$ **NPC** where $L \leq_P$ 3SAT.

Neither. Same as above.

15. There is a language $L \in$ **P** where 3SAT $\leq_P L$.

P = **NP**. This means 3SAT $\in$ **P**, and since 3SAT is **NP**-complete, **P** = **NP**.

16. There is a language $L \in$ **P** where 3SAT $\leq_M L$.

Neither. We can reduce 3SAT to the regular language `0*` by using the function

$f(\langle\varphi\rangle) = 0$ if φ is satisfiable

$f(\langle\varphi\rangle) = 1$ otherwise

Since 3SAT is decidable, this function is computable (run an NTM for φ, outputting 0 if the NTM accepts and 1 otherwise). We thus have a mapping reduction from 3SAT to a language in **P**.

17. All languages in **P** are decidable.

Neither. **P** $\subset$ **R**.

18. All languages in **NP** are decidable.

Neither. **NP** $\subset$ **R**.

19. There is a polynomial-time algorithm that correctly decides SAT for all strings of length ***at most*** 10^{100}, but that might give incorrect answers for longer strings.

Neither. We can already design such an algorithm by having the algorithm see if the string has length at most 10^{100}. If so, it checks all possible variable assignments to the formula and sees if any are satisfying assignments. Otherwise, it immediately rejects. This algorithm runs in time $O(n)$, since as n gets "large" (length at least 10^{100}), the runtime depends only on n. However, this algorithm doesn't say anything about whether **P** = **NP**.

20. There is a polynomial-time algorithm that correctly decides SAT for all strings of length ***at least*** 10^{100}, but that might give incorrect answers for shorter strings.

P = **NP**. We can use this algorithm as a subroutine in a polynomial-time algorithm for SAT as follows: if the string has length less than 10^{100}, then check whether the formula is satisfiable using a brute-force solution. Otherwise, run the polynomial-time algorithm. Since as n gets "large" (length at least 10^{100}) the runtime is polynomial, this algorithm runs in polynomial time. Thus SAT ∈ **P**, so **P** = **NP**.

Problem Three: The Big Picture (24 Points)

The following solutions are just examples of correct answers; there are infinitely many correct answers for each problem, so don't worry if yours doesn't exactly match ours!

1. One regular language is `0*`. We can prove that it is regular by building a DFA, NFA, or regular expression for it.
2. One CFL that is not regular is $\{ \texttt{0}^n\texttt{1}^n \mid n \in \mathbb{N} \}$. We can show it to be context-free by designing a CFG or PDA for it. We can prove it is nonregular using the pumping lemma.
3. Any regular language belongs to **P**, so Σ* is one example. We know it's in **P** because every regular language is in **P**.
4. One language in **NP** suspected not to be in **P** is *INDSET*. We can show it is in **NP** by designing a polynomial-time verifier for it or building a nondeterministic polynomial-time algorithm for it. We suspect it is not in **P** because it is **NP**-complete by a reduction from 3SAT.
5. One language in **RE** that is not in **R** is A_{TM}. We can prove that it is in **RE** by showing that it is the language of the universal Turing machine. We can prove that it is not in **R** by reducing $\overline{L_D}$ to it; since $\overline{L_D}$ is not R, A_{TM} is not **R** either.
6. One language in co-**RE** that is not in **R** is L_D. We can prove that it is not in **R** by contradiction; if L_D were in **R**, then it would be in **RE**, but we have a diagonal argument that proves that it cannot be **RE**.
7. One language that is neither **RE** nor co-**RE** is EQ_{TM}. We can show this by reducing L_D and its complement to $REGULAR_{TM}$.