Attach a fully filled-in cover sheet to the front of your printed homework. Your name should not appear anywhere except on the cover sheet; each individual page of the homework should include your banner ID instead.

While collaboration is encouraged in this class, please remember not to take away notes from any collaboration sessions. Also please list the names and logins of any of your collaborators at the beginning of each homework.

Please monitor Piazza, as we will post clarifications of questions there. You should hand in your solutions by 12:55 to the CSCI 51 bin on the second floor of the CIT. Late homeworks are not accepted.

Problem 1

**Note:** This problem uses material that will be presented on Tuesday.

For each of the following $2 \times 3$ windows, decide whether it is:

(a) A legal window in a tableau regardless of the Turing Machine’s $\delta$ function;

(b) A legal window in a tableau for certain $\delta$ functions; or

(c) An illegal window in a tableau regardless of the Turing Machine’s $\delta$ function.

Assume that $\Gamma = \{a, b, c, \omega\}$

For example, the following window is completely illegal:

```
<table>
<thead>
<tr>
<th>q0</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>b</td>
<td>q1</td>
</tr>
</tbody>
</table>
```

because the tape head moves more than one space.
Problem 2

In this problem we will see how slight changes to NP-hard languages can create languages that are in the class $P$.

a. We define the language

$\text{k-Colorability} = \{ \langle G \rangle \mid G$ is a graph with vertices $V$ and edges $E$ and there exists some $k$-partitioning of $V$, $V = \{ V_1, V_2, \ldots, V_k \}$, such that for all $(u, v) \in E$ $u$ and $v$ belong to different elements of $V$ $\}$.

In class, we will show that 3-Colorability (i.e. $k = 3$) is NP-Complete. Prove that 2-Colorability is in $P$.

b. We will also see that $\text{SubsetSum} = \{ \langle S, t \rangle \mid S$ is a set of numbers with a subset that sums to $t$, where $S$ and $t$ are both encoded in binary.$\}$

is NP-Complete. However, the representation of the numbers matters! Show that $\text{UnarySubsetSum}$ is in $P$, where

$\text{UnarySubsetSum} = \{ \langle S, t \rangle \mid S$ is a set of numbers with a subset that sums to $t$, where $S$ and $t$ are both encoded in unary.$\}$
Problem 3

Given an undirected graph $G = (V, E)$ and $a, b \in V$, a simple path from $a$ to $b$ is a path containing no repeated vertices. Let

$SPATH = \{(G, a, b, k) \mid G \text{ contains a simple path from } a \text{ to } b \text{ of length } \leq k\}$

$LPATH = \{(G, a, b, k) \mid G \text{ contains a simple path from } a \text{ to } b \text{ of length } \geq k\}$.

a. Show that $SPATH \in P$.

b. Show $LPATH$ is $NP$-complete. (Hint: $\text{HAM回TONIANPATH}$ is $NP$-complete.)
The following questions are lab problems.

Lab Problem 1

Recall that the language SAT is defined as

\[ \text{SAT} = \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable boolean formula} \} \]

where \( \langle \phi \rangle \) is a description of a boolean formula over some alphabet (say \( \Sigma = \{0, 1\} \)). In Homework 1 you learned about CNF (conjunctive normal form) for a boolean formula, where a formula is expressed as the conjunction (AND) of a number of clauses, and a clause is the disjunction (OR) of literals (go back and check out Homework 1 if you don’t remember what this is supposed to mean). In this problem we will use boolean formulas in a new form, namely 3CNF (3-conjunctive normal form). Unsurprisingly, 3CNF is similar to CNF, but with every clause having exactly three distinct literals. This allows us to define the language 3SAT as follows:

\[ \text{3SAT} = \{ \langle \phi \rangle \mid \phi \text{ is in 3-CNF and } \phi \text{ is a satisfiable boolean formula} \} \]

In this problem you will work with a reduction from SAT to 3SAT. If this task seems daunting, have no fear! We will provide you with most of the reduction, leaving the completion and proof of it to you. The reduction will consist of three steps: **Formula to circuit**, **Circuit to clauses**, and **Clauses to 3CNF**. Here comes the description of the first two:

The reduction

**Formula to circuit**: The idea is to first construct a binary “parse tree” for the input formula \( \phi \), with the literals as leaves and the connectives as internal nodes. This is done by parenthesizing the expression so that each parenthesis contains a Boolean operation of two elements, either clauses or literals. This is best shown by example: The clause \( x_1 \land x_2 \land x_3 \land x_4 \) can be parenthesized as follows:

\[ x_1 \land x_2 \land x_3 \land x_4 = (((x_1 \land x_2) \land x_3) \land x_4) \]

Now the formula \( \phi \) can also be viewed as a Boolean circuit (remember those from CS022?) where each internal node is a logic gate (AND,OR,NOT). So this step outputs a circuit \( C \) expressing a Boolean formula \( \phi \).
Circuit to clauses: Introduce a variable \( y_i \) for the output wire of each internal gate. A circuit \( C \) with \( m \) gates is represented using \( m + 1 \) clauses; one for each gate and an additional one for the output wire. We will denote these clauses by \( \varphi_i \) with \( 1 \leq i \leq m + 1 \), and the output of this step will be denoted \( \varphi \).

The form of a clause depends on the type of the gate it is representing.

- If \( y_i \) is the variable corresponding to the output wire of an AND gate with input variables \( x_j \) and \( x_k \) corresponding to the input wires, then we write \( y_i \leftrightarrow (x_j \land x_k) \) as the \( i \)th clause, i.e., \( \varphi_i = y_i \leftrightarrow (x_j \land x_k) \).

- If \( y_i \) is the variable corresponding to the output wire of an OR gate with input variables \( x_j \) and \( x_k \) corresponding to the input wires, then we write \( y_i \leftrightarrow (x_j \lor x_k) \) as the \( i \)th clause, i.e., \( \varphi_i = y_i \leftrightarrow (x_j \lor x_k) \).

- If \( y_i \) is the output variable corresponding to the output wire of a NOT gate with input variable \( x_j \) on the input wire, then we write \( y_i \leftrightarrow \overline{x_j} \) as the \( i \)th clause, i.e., \( \varphi_i = y_i \leftrightarrow \overline{x_j} \).

For example, if one of your clauses from the parenthesized expression of \( \phi \) were \( (x \land z) \) and the outgoing wire from the corresponding gate in the circuit were called \( y_i \), you would now have a clause that reads \( (y_i \leftrightarrow (x \land z)) \). Note that each clause has at most three literals. The \( m + 1 \)th clause is just the variable on the output wire of the circuit \( C \), i.e., \( \varphi_{m+1} = y_{m+1} \), assuming \( y_{m+1} \) is the name of the variable you assigned to the output wire. We take the disjunction (OR) of all clauses \( \varphi_i \) to get the output \( \varphi \) of this step:

\[
\varphi = \varphi_1 \land \varphi_2 \land \ldots \land \varphi_{m+1}
\]

In case this sounds like Klingon to you, let us illustrate with an example. Let \( \psi \) be a clause in CNF that reads

\[
\psi = (x_1 \land x_2) \lor (x_1 \lor x_3 \lor x_4 \lor x_5)
\]

\( \psi \) is transformed into the parenthesized \( \phi \) by the first step:

\[
\phi = ((x_1 \land x_2) \lor ((x_1 \lor x_3) \lor x_4) \lor x_5))
\]
Below is a depiction of the circuit $C$ corresponding to $\phi$:

By applying the rules from the second step, you derive the clauses $\varphi_i$ and the whole boolean formula $\varphi$.

$$
\varphi_1 = (y_1 \leftrightarrow (x_1 \land x_2)) \\
\varphi_2 = (y_2 \leftrightarrow (y_1 \lor y_3)) \\
\varphi_3 = (y_3 \leftrightarrow \overline{y_4}) \\
\varphi_4 = (y_4 \leftrightarrow (y_5 \lor x_5)) \\
\varphi_5 = (y_5 \leftrightarrow (x_4 \lor y_6)) \\
\varphi_6 = (y_6 \leftrightarrow (x_1 \lor x_3)) \\
\varphi_7 = y_2 \\
\varphi = \varphi_1 \land \varphi_2 \land \varphi_3 \land \varphi_4 \land \varphi_5 \land \varphi_6 \land \varphi_7
$$

Your Lab Problem 1 Mission: to explore strange new worlds, to seek out new life and new civilizations, and to boldly:

(a) Complete the reduction by transforming each clause $\varphi_i$ into 3CNF.

(b) Prove that the reduction is correct.

(c) Show that all the steps of the reduction can be computed in polynomial time.

---

1… go where multiple men and women have been before, such as, but not limited to, your loving TAs.
Lab Problem 2

Lok Durd is running his android prototype through some higher-level logic testing by asking it to determine whether certain operations are closed on \( NP \) and \( NP \)-complete languages. Unfortunately, he doesn’t know the correct answers! Help him out, so that he can create a fully-functional android.

For the purposes of this problem, only consider languages over the alphabet \{0, 1\}.

a. Let \( L_1 \) and \( L_2 \) be two languages in \( NP \).

   (i) Must \( L_1 \cup L_2 \) be in \( NP \)?

   (ii) Must \( L_1 \cap L_2 \) be in \( NP \)?

b. Now, suppose \( L_1 \) and \( L_2 \) are both \( NP \)-complete.

   (i) Must \( L_1 \cup L_2 \) be \( NP \)-complete?

   (ii) Must \( L_1 \cap L_2 \) be \( NP \)-complete?

Prove your answers.

Lab Problem 3

Consider the following three languages:

\[
\text{PRIMES} = \{ p \mid p \text{ is a binary representation of a prime integer} \}
\]

\[
\text{COMPOSITES} = \{ n \mid n \text{ is a binary representation of a composite integer} \}
\]

\[
\text{COMPOSITESWithHint} = \{ (n, a, b) \mid n \in \text{COMPOSITES}, \text{ and there exists } p \in \text{PRIMES} \text{ such that } a \leq p \leq b \text{ and } p \text{ divides } n \}.
\]

In 2002, Agrawal, Kayal and Saxena (AKS) proved that the language \text{PRIMES} is in \( P \).

For this problem, you may use, without proof, the fact that arithmetic operations (multiplication, division, addition and subtraction) and comparisons (equal, less than) on binary integers take polynomial time in the size of the binary representations of these integers.
(a) Show that it follows from the AKS result that \text{Composites} \in P.

(b) Even though the languages \text{Primes} and \text{Composites} are both in P, there is no known polynomial-time algorithm that finds the prime factorization of an input integer \( n \). (Define that the \textit{prime factorization} of \( n \) to be a list of prime numbers, possibly with repetitions, \( p_1, \ldots, p_k \) such that \( n = \prod_{i=1}^{k} p_i \) and \( p_i \leq p_j \) for all \( i \leq j \).)

This is where the language \text{CompositesWithHint} is important. Prove that (1) if \text{CompositesWithHint} \in P, then there is a polynomial-time algorithm that finds the prime factorization of an input \( n \); and (2) if there exists a polynomial-time algorithm that finds the prime factorization of an input \( n \), then \text{CompositesWithHint} \in P.

(c) Prove that \text{CompositesWithHint} \in \text{NP} \cap \text{coNP} (recall that the class \text{coNP} consists of the languages whose complement is in \text{NP}).

(d) Show that if \text{CompositesWithHint} is \text{NP}-complete, then \text{Tautology} \in \text{NP}, where the language \text{Tautology} is defined as follows:

\[
\text{Tautology} = \{ \langle \phi \rangle \mid \phi \text{ is a Boolean formula that evaluates to TRUE on all possible assignments} \}.
\]

It is widely conjectured that\(^2\) \text{Tautology} is not in \text{NP} (Do you have any intuition as to why?), and therefore \text{CompositesWithHint} is not believed to be \text{NP}-complete. However, it is also widely conjectured that \text{CompositesWithHint} is \text{not} in P. Thus \text{CompositesWithHint} is an example of a language in \text{NP} widely believed not to be in \( P \), and not to be \text{NP}-complete.

---

\(^2\)This is just a nice way of saying “Without any proof whatsoever, most mainstream theoretical computer scientists, including Anna, are convinced that..."