Problem 1

For each of the following problems, say whether it is decidable or not. Justify your answer by describing an appropriate Turing Machine, or by reducing from \( \text{ALL}_{\text{CFG}} \) which was shown undecidable in class. Assume that the alphabet of CFG \( G \) contains the symbol \( a \).

(a) \( L_1 = \{ \langle G \rangle : \text{CFG } G \text{ generates at least one string that starts with } a \} \).

(b) \( L_2 = \{ \langle G \rangle : \text{CFG } G \text{ generates all strings that start with } a \} \).

Solution

(a) Decidable. To decide \( L_1 \), we want to determine if there is a derivation that begins with the start variable and produce a string that starts with \( a \). It will be convenient to convert the CFG in Chomsky Normal form. Since we only care whether the first symbol is an \( a \), we may disregard all terminal productions of the form \( A \rightarrow x \) where \( x \) is anything other than \( a \). For the same reason, we can replace every nonterminal production \( A \rightarrow BC \) by \( A \rightarrow B \); this does not affect the first symbol of derivations. After we do these simplifications, we end up with a very simple CFG. For example, if \( G \) is the CFG

\[
S \rightarrow AA \mid BA \\
A \rightarrow BB \mid a \\
B \rightarrow AB \mid b
\]

the following CFG \( G' \) preserves the property that “at least one string starts with \( a \):

\[
S \rightarrow A \mid B \\
A \rightarrow B \mid a \\
B \rightarrow A
\]

Now we can represent this CFG by a directed graph whose nodes are the terminals, as well as the nonterminal \( a \), and there is a directed edge \( A \rightarrow x \) for every such production. If there is a path from \( S \) to \( a \) in this graph, then \( G' \) has a string that starts with \( a \), and so does \( G \); if not, this cannot be the case.

Here is a Turing Machine that decides \( L_1 \). In this Turing Machine description the conversion of \( G \) into \( G' \) and the construction of the graph happen in one step.
On input \( \langle G \rangle \):
- Convert \( G \) to Chomsky Normal Form.
- Construct the following graph \( H \):
  - The nodes of \( H \) are the variables of \( G \) plus the terminal \( a \).
  - For every rule of the form \( A \rightarrow BC \) in \( G \), put an edge from \( A \) to \( B \).
  - For every rule of the form \( A \rightarrow a \) in \( G \), put an edge from \( A \) to \( a \).
- If \( H \) has a path from \( S \) to \( a \), accept. Otherwise reject.”

(b) **Undecidable.** To show \( L_2 \) is undecidable, we reduce from \( ALL_{CFG} \). In other words, let us assume that \( L_2 \) is decidable by a TM \( M \). We then show how to construct a TM that decides \( ALL_{CFG} \), contradicting the fact that the latter is undecidable.

To achieve this plan, we need to find a way to convert a \( G \) into \( G' \) so that \( G' \) generates all strings that start with \( a \) if and only if \( G \) generates all strings. We do so by first copying all the rules of \( G \); then we introduce a new start variable \( S' \) and add the rule \( S' \rightarrow aS' \), where \( S \) is the start variable of \( G \).

Notice that \( G' \) then generates all strings in \( G \) preceded by the symbol \( a \). So if \( G \) generates all strings, \( G' \) will generate all strings that start with \( a \). Conversely, if \( G \) fails to generate some string \( w \), then \( G' \) will fail to generate \( aw \).

Let us now write down this reduction formally. Assume that \( M \) is a TM that decides \( L_2 \). Then the following TM will decide \( ALL_{CFG} \):

On input \( \langle G \rangle \):
- Construct the following CFG \( G' \):
  - Copy all the rules of \( G \).
  - Add a new start variable \( S' \) and add the production \( S' \rightarrow aS' \)
  - Run \( M \) on input \( G' \) and return its answer.

By the above reasoning, \( G' \) generates all strings that start with \( a \) if and only if \( G \) generates all strings. So if \( M \) decides \( L_2 \), this TM will decide \( ALL_{CFG} \) which is impossible, since \( ALL_{CFG} \) is undecidable. Therefore \( L_2 \) must be undecidable as well.

**Problem 2**

For each of the following problems, show that it is NP-complete (namely, (1) it is in NP and (2) some NP-complete language reduces to it.) When showing NP-completeness, you can start from any language that was shown NP-complete in class or tutorial.

(a) \( L_1 = \{ \langle \phi \rangle : \phi \) is a boolean formula that has at least two satisfying assignments\}.

(b) \( L_2 = \{ \langle G, k \rangle : G \) is a graph that contains a clique of size \( k \) or an independent set of size \( k \} \).
Solution

(a) To show $L_1$ is in NP, we give a verifier for $L_1$: On input $⟨φ, a, b⟩$, where $φ$ is a boolean formula and $a$, $b$ are candidate assignments, reject if $a = b$. If $a$ is a satisfying assignement for $φ$ and $b$ is also a satisfying assignment for $φ$, accept, otherwise reject. The running time of this verifier is polynomial, as it merely needs to tell if two strings are equal and if each of them is a satisfying assignment for $φ$.

To show $L_1$ is NP-hard, we give a reduction from SAT. We need a way to turn a boolean formula $φ$ into a new boolean formula $φ’$ so that if $φ$ is satisfiable, then $φ’$ has two satisfying assignments and if not, then $φ’$ has less than two satisfying assignments. For example, the following $φ’$ is like that:

$$φ’ = (φ \land y) \lor (φ \land \overline{y})$$

where $y$ is a new variable that does not appear in $φ$. If $φ$ is satisfiable and $a$ is its satisfying assignment, then $φ’$ has at least two satisfying assignments: one is $a$, $y = false$, the other one $a$, $y = true$. If $φ$ is not satisfiable, then for any assignment $φ$ will evaluate to false, and so will $φ’$ (regardless of what is assigned to $y$).

The Turing Machine that on input $φ$, outputs $(φ \land y) \lor (φ \land \overline{y})$ implements this reduction. Since it runs in polynomial time, $L_1$ is NP-hard.

(b) To show $L_2$ is in NP, we give a verifier for $L_2$: On input $⟨G, k, S⟩$, where $S$ is a set of vertices of $G$, if $S$ has size less than $k$ reject. Otherwise, for every pair of vertices in $S$ check if there is an edge. If all the edges are there (so $G$ has a $k$-clique), accept. If none of the edges are there (so $G$ has an independent set), accept. Otherwise reject. This verifier needs to perform $O(k^2)$ checks on pairs of vertices, and so it runs in polynomial time.

We now argue that $L_2$ is NP-hard. There are two natural problems we can try to reduce from: clique or independent set. Let’s try CLIQUE. So given $⟨G, k⟩$, we want to come up with $⟨G’, k’⟩$ so that (1) if $G$ has a $k$-clique then $G’$ has a $k’$-clique or a $k’$-independent set, and (2) if $G$ has no $k$-clique, then $G’$ has neither.

Condition (1) is easy to satisfy; we can just take $G’ = G$ and $k’ = k$. But then condition (2) may be false: $G$ could have no $k$-clique but it could well have a $k$-independent set, in which case so will $G’$.

So we have to do something that will “kill” all independent sets in $G’$ of size $k’$, but make sure that cliques of size $k$ in $G$ become cliques of size $k’$ in $G’$. Here is one way to do it. Suppose $G$ has $n$ vertices. Then $G’$ will have $2n + 1$ vertices. The first $n$ of these will be a copy of $G$, and the other $n + 1$ will be connected to all the other vertices (the first $n$ as well as among themselves). Now set $k’ = k + n + 1$.

Let us describe $G$ a bit more formally. If the vertices of $G$ are $\{v_1, \ldots, v_n\}$, then the vertices of $G’$ will be $\{v_1, \ldots, v_n, w_1, \ldots, w_{n+1}\}$. For every edge $\{vi, vj\}$ in $G$ the same edge will be present in $G’$, but $G’$ will have the additional edges from every $w_i$ to every $v_j$ and for every $w_i$ to every $w_j$.

If $G$ has a clique $S$ of size $k$, then $G’$ will have a clique of size $k’ = k + n + 1$, namely the clique $S \cup \{w_1, \ldots, w_{n+1}\}$. 
If \( G \) has no clique of size \( k \), then \( G' \) cannot have a clique of size \( k + n + 1 \): Any such clique must contain at least \( k \) of the vertices \( \{v_1, \ldots, v_n\} \), and the restriction to those vertices would be a clique of size \( k \) in \( G \).

Moreover, \( G' \) cannot have an independent set of size \( k' = k + n + 1 \) because any set of \( k' \) vertices must contain one of the vertices \( w_i \), and this vertex is connected to everything so it cannot be part of an independent set.

The construction of \( G' \) and \( k' \) from \( G \) and \( k \) requires building a graph with \( O(n) \) vertices \( O(n^2) \) edges, so it can be done in polynomial time. Therefore \( \text{CLIQUE} \) reduces to \( L_2 \) in polynomial time and so \( L_2 \) is NP-hard.

**Problem 3**

Throughout the semester, we looked at various models of computation and we came up with the following hierarchy:

\[
\text{regular} \subseteq \text{context-free} \subseteq \mathbb{P} \subseteq \text{NP decidable} \subseteq \text{recognizable}
\]

We also gave examples showing that the containments are strict (e.g., a context-free language that is not regular), except for the containment \( \mathbb{P} \subseteq \text{NP} \), which is not known to be strict.

There is one gap in this picture between NP languages and decidable languages. In this problem you will fill this gap.

(a) Show that \( 3\text{SAT} \) is decidable, and the decider has running time \( 2^{O(n)} \). (Unlike a verifier for \( 3\text{SAT} \) which is given a 3CNF \( \phi \) together with a potential satisfying assignment for \( \phi \), a decider for \( 3\text{SAT} \) is only given a 3CNF but not an assignment for it.)

(b) Argue that for every language \( L \) in NP there is a constant \( c \) such that \( L \) is decidable in time \( 2^{O(n^c)} \). (Use the Cook-Levin Theorem.)

(c) Let \( D \) be the following Turing Machine:

\[
D: \text{ On input } \langle M, w \rangle, \text{ where } M \text{ is a Turing Machine,} \\
\quad \text{Run } M \text{ on input } \langle M, w \rangle \text{ for at most } 2^{|w|} \text{ steps.} \\
\quad \text{If } M \text{ accepts } \langle M, w \rangle \text{ within this many steps, reject;} \\
\quad \text{Otherwise (if } M \text{ rejects or hasn’t halted), accept.}
\]

Show that the language decided by \( D \) cannot be decided in time \( 2^{O(n^c)} \) for any constant \( c \), and therefore it is not in NP.

**Hint:** Assume that \( D \) can be decided in time \( 2^{O(n^c)} \). What happens when \( D \) is given input \( \langle D, w \rangle \), where \( w \) is a sufficiently long string?
Solution

(a) To decide 3SAT, we do the following. Given a 3CNF formula \( \phi \) with \( k \) variables, we list all possible \( 2^k \) assignments to \( \phi \). If any one of these assignments satisfies \( \phi \) we accept, otherwise we reject. The running time of this algorithm is \( O(n2^k) = 2^{O(n)} \), where \( n \) is the length of \( \phi \).

(b) Let \( L \) be any language in NP. By the Cook-Levin theorem, there is a reduction from \( L \) to 3SAT that runs in time \( O(n^c) \) for some constant \( c \). Consider the following TM for \( L \): On input \( x \), first reduce \( x \) to an instance \( \phi \) of 3SAT, then run the decider from part (a) on \( \phi \) and return its answer. Since the running time of the reduction is \( O(n^c) \), \( \phi \) has size at most \( O(n^c) \), so the running time of the decider for \( L \) is \( O(n^c) + 2^{O(n^c)} = 2^{O(n^c)} \).

(c) Suppose that the language of \( D \) can be decided in time \( t(n) = 2^{O(n^c)} \) and let \( M \) be a decider for the language of \( D \) with this running time. Consider what happens when \( D \) is given input \( \langle M, w \rangle \), where \( w \) is any sufficiently long string. Specifically, assume the length \( n \) of \( w \) satisfies the condition \( t(n) \leq 2^{2n} \). Since \( D \) simulates \( M \) on input \( \langle M, w \rangle \) for \( 2^{2n} \) steps, \( M \) will have halted on input \( w \) by the end of the simulation. So if \( M \) accepts \( \langle M, w \rangle \) then \( D \) will reject \( \langle M, w \rangle \) and vice-versa. So the language of \( D \) and the language of \( M \) must differ on input \( \langle M, w \rangle \), and therefore \( M \) cannot be a decider for the language of \( D \).

Since \( D \) runs in time at most \( O(2^{2n}) \) it is a decider, but the language it decides cannot be decided in time \( 2^{O(n^c)} \) for any constant \( c \). By part (b), all NP languages can be decided in time \( 2^{O(n^c)} \). Therefore the language of \( D \) cannot be in NP.

Problem 4

A heuristic is an algorithm that often works well in practice, but it may not always produce the correct answer. In this problem, we will consider a heuristic for 3SAT.

Let \( \phi \) be a CNF formula and \( x \) a literal in \( \phi \). Suppose we set \( x \) to TRUE. The reduced form of \( \phi \) is the formula obtained by discarding all clauses of \( \phi \) that contain \( x \) and removing \( \overline{x} \) from all the other clauses. For example, if \( \phi = (x_1 \vee x_2) \land (\overline{x}_1 \vee x_3) \land (x_2 \vee x_3) \), then setting \( x_1 = \text{TRUE} \) gives the reduced form \( x_3 \land (x_2 \vee x_3) \), while setting \( \overline{x}_1 = \text{TRUE} \) gives the reduced form \( x_2 \land (x_2 \vee x_3) \). Consider the following heuristic \( H \) for 3SAT:

On input \( \langle \phi \rangle \), where \( \phi \) is a 3CNF formula with \( n \) variables:

For \( i := 1 \) to \( n \), repeat the following:

If \( x_i \) appears in \( \phi \) more often than \( \overline{x}_i \), set \( x_i = \text{true} \).

Otherwise, set \( \overline{x}_i = \text{true} \).

Replace \( \phi \) with its reduced form. If \( \phi \) contains an empty clause, reject.

accept.

(a) Show that \( H \) runs in polynomial time.

(b) Show that if \( H \) accepts \( \phi \), then \( \phi \in 3SAT \).
(c) Show that it is possible that $H$ rejects $\phi$, even though $\phi \in 3SAT$.

**Solution**

(a) Let $m$ be the length of $\phi$. In each loop of the iteration, $H$ has to count the number of appearances of $x_i$ and $\overline{x}_i$ in $\phi$, which takes $O(m)$ steps. Then it has to substitute a value for $x_i$ and reduce the formula, which also takes $O(m)$ steps. (While reducing, it figures out if there is an empty clause or not.) Since there are $n$ iterations, the total number of steps is $O(nm)$, which is polynomial in the input size.

(b) If $H$ accepts $\phi$, then in the process of assigning $x_1, x_2, \ldots, x_n$, no empty clause is ever created. Since $x_i$ satisfies all the clauses discarded in the $i$th iteration, all clauses of $\phi$ are satisfied by this assignment.

(c) Let $\phi$ be the formula $(x_1 \lor x_1 \lor x_1) \land (\overline{x}_1 \lor \overline{x}_1 \lor x_2) \land (\overline{x}_1 \lor \overline{x}_1 \lor x_3)$. This formula is satisfiable (set all variables to true), but $H$ will choose to assign $x_1$ to false, create an empty clause after reducing, and reject.