HW1

1. An eraserless Turing machine (ETM) has two special tape characters □ and ■ (in addition to any other tape characters). The idea is that the ETM can only “add more ink” to each cell of its tape. So □ is the “blank” character, which an ETM can replace with any other character that it likes. But an ETM’s only choice when reading a non-blank character is to leave it unchanged or to replace it with ■.

Show that an arbitrary TM $M$ can be simulated by a ETM. Describe the behavior of the ETM in terms of its general tape-head movements & modifications, and be sure to argue why the ETM uses only a constant number of internal states. Consider giving short illustrative examples.

Note: This implies that the halting problem for ETMs is undecidable.

Solution

The idea is to simulate one step of $M$ by copying an entire configuration of $M$ to the fresh part of the tape, and updating it as it’s being copied. Copy the configuration one character at a time, and use the ability to write once (“cross things off”) to keep track of which characters have been copied.

When copying a character $c$, first check to see if the tape head is just to the left of $c$. If so, then the tape head might move onto character $c$ in the next configuration, and you’ll have to know this while copying $c$. When copying the character that contains the tape head, update that tape cell according to the transition rule of $M$. You can use finite state to “remember” which state $M$ was in.

2. One of the following is decidable, and the other is not. Give a high-level algorithm for the decidable language (e.g., simulate this TM on this input for this many steps, then do this...); justify why it is correct and why it always halts. For the undecidable language, show a reduction (in the correct direction) involving a problem known to be undecidable.

(a) \{ $M$ | $M$ halts on every input $w$ after at most $|w|$ steps\}
(b) \{ $\langle M, t \rangle$ | $M$ halts on every input $w$ after at most $t$ steps\}

Note: Rice’s theorem doesn’t apply to either of these. Can you see why?

Solution

Problem (a) was thrown out. Let $L$ be the language defined in part (b). It is decidable.

The algorithm is: for every input $w$ of length at most $t$, run $M$ for $t$ steps. If for any $w$, $M$ has not halted in $t$ steps, then reject; else accept.

The algorithm rejects only when it has concrete proof that $M$ exceeds $t$ steps on some input. So the algorithm’s “no” answer is always correct.

If the algorithm accepts, then $M$ halts in at most $t$ steps for all inputs of length at most $t$. But that means $M$ must halt on all inputs in at most $t$ steps, because $t$ steps are not enough time to “touch” more than $t$ input bits.

3. Let $\mathcal{P}$ be a property of pairs of languages. $\mathcal{P}$ is nontrivial if there exist languages $A_0, B_0, A_1, B_1$ such that $(A_0, B_0)$ do not have property $\mathcal{P}$ and $(A_1, B_1)$ have property $\mathcal{P}$. For example, the
following are all nontrivial properties of language pairs \((A, B)\):

\[
[A \subseteq B]; \quad [A = B]; \quad [A \cap B = \emptyset]; \quad [|A| < |B|]; \quad [A \text{ is finite and } B \text{ is infinite}]
\]

Show the following generalization of Rice’s theorem:

\[
\{\langle M, M' \rangle \mid (L(M), L(M')) \text{ have property } P\} \text{ is undecidable if } P \text{ is nontrivial.}
\]

\((M \text{ and } M' \text{ are Turing machines in the above expression.})\) This implies that it is undecidable to determine whether two TMs accept the same language, whether one accepts a subset of the other’s, whether they accept disjoint languages, etc.

**Solution**

Let \(P\) be a nontrivial property. By symmetry suppose \((\emptyset, \emptyset)\) do not have the property, but \((L(M'_Y), L(M'_Y))\) do have the property.

Let \(L_P\) denote the language above, and suppose for contradiction it is decidable. We show how to solve the acceptance problem.

On input \(\langle \tilde{M}, \tilde{x} \rangle\), we wish to determine whether \(\tilde{M}\) accepts \(\tilde{x}\).

Write down, but don’t execute, the code for the following algorithms, which have \(\tilde{M}\) and \(\tilde{x}\) and \(M_Y\) and \(M'_Y\) hard coded:

\(M\) : on input \(z\), simulate \(\tilde{M}\) on input \(\tilde{x}\). If \(\tilde{M}\) accepts, then run \(M_Y\) on \(z\).

\(M'\) : on input \(z\), simulate \(\tilde{M}\) on input \(\tilde{x}\). If \(\tilde{M}\) accepts, then run \(M'_Y\) on \(z\).

Now accept if and only if \(\langle M, M' \rangle \in L_P\).

Note that this algorithm always halts. And also note that the TM descriptions \(M, M'\) that are generated can only do two things: either \(L(M) = \emptyset\) or \(L(M) = L(M'_Y)\), depending only on whether \(\tilde{M}\) accepts \(\tilde{x}\). Similarly, either \(L(M') = \emptyset\) or \(L(M') = L(M'_Y)\).

Our algorithm correctly decides the accepting problem because:

\[
\text{this algorithm accepts } \langle \tilde{M}, \tilde{x} \rangle \iff \langle M, M' \rangle \in L_P \\
\iff (L(M), L(M')) \text{ has property } P \\
\iff L(M) = L(M_Y) \text{ and } L(M') = L(M'_Y) \\
\iff \tilde{M} \text{ accepts } \tilde{x}
\]

**HW2**

1. A decompression algorithm \(D\) has the property that \(K_D(x)\) is always a power of two. Can \(D\) be asymptotically optimal?

**Solution**

No, it can’t be optimal. In lecture & in the notes, it was shown (using a simple counting argument) that for every \(n\), there exists a string \(x\) of length \(n\) such that \(K_D(x) \geq n\).

Consider \(n\) of the form \(n = 1 + 2^k\). We have that for every \(k\), there exists a string \(x\) of length \(1 + 2^k\) such that \(K_D(x) > 2^k\). But since \(K_D(x)\) is always a power of two, we have \(K_D(x) \geq 2^{k+1} = 2|x| - 1\).

But if \(K_D(x)\) were asymptotically optimal, then \(K_D(x) \leq |x| + O(1)\). Putting together \(|x| + O(1) \geq 2|x| - 1\) gives a contradiction since this is true for an infinite number of \(x\)'s.
2. Let $D_1, D_2, \ldots, D_i, \ldots$ be an infinite sequence of programs. We say that the sequence is **enumerable** if there is a TM $D^*$ that on input $i$ outputs a description $\langle D_i \rangle$.

Let $D_1, D_2, \ldots$ be an enumerable sequence of decompression algorithms. Let $U$ be a universal decompression algorithm. Show that

$$K_U(x) \leq \min_i [K_{D_i}(x) + \log i] + O(1)$$

for all $x$ (the $O(1)$ does not depend on $x$).

**Solution**

Given $x$ and any $i$, let $y$ be the optimal compression of $x$ with respect to $D_i$ (so that $|y| = K_{D_i}(x)$ and $D_i(y) = x$).

Consider the program

“Run $D^*$ on input $i$ to obtain $\langle D_i \rangle$, then run $D_i$ on $y$”

where $D^*$, $i$, and $y$ are hard-coded. $D^*$ doesn’t depend on $x$, but $i$ and $y$ do. The length of this program is $|y| + \log i + O(1) = K_{D_i}(x) + \log i + O(1)$ and this is a program that outputs $x$. So $K_U(x) \leq K_{D_i}(x) + \log i + O(1)$. Since this holds for any $i$, we get the desired inequality from the problem statement.

3. Suppose a string $x$ has length $n$ and also contains a run of $\ell$ consecutive zeroes. Show that $K(x) \leq n - \ell + \log n + \log \ell + O(1)$.

Show that incompressible strings of length $n$ contain no runs of zeroes longer than $O(\log n)$.

**Solution**

Suppose $x$ has a run of $\ell$ zeroes. We can completely describe $x$ by the following algorithm: “Insert $\ell$ [hard-coded] zeroes at position $i$ [hard-coded] into the following [hard-coded] $n - \ell$ bit string.” The size of this algorithm is $\log \ell + \log n + (n - \ell) + O(1)$, as desired.

An incompressible string of length $n$ has $K(x) \geq n$. Substituting & simplifying, we get:

$$n \leq K(x) \leq n - \ell + \log n + \log \ell + O(1)$$

$$\implies \ell \leq \log n + \log \ell + O(1)$$

$$\implies \ell \leq 2 \log n + O(1) = O(\log n)$$

The last inequality follows from the fact that $\ell \leq n$, and so $\log n + \log \ell \leq 2 \log n$.

4. Fix a universal machine $U$ and prove that the following language is undecidable:

$$L_{K\text{-compare}} = \{ \langle x, y \rangle \mid |x| = |y| \text{ and } K_U(x) \leq K_U(y) \}$$

**Hint:** compute an $n$-bit string of maximal Kolmogorov complexity.

**Solution**

Suppose for contradiction that $L_{K\text{-compare}}$ is decidable. Consider an algorithm $M$ that does the following. It has a value $n$ hard coded. It loops over every string of length $n$, and using the standard “high-water mark algorithm” it uses the $L_{K\text{-compare}}$ subroutine to find the $x \in \{0,1\}^n$ of highest Kolmogorov complexity, and finally outputs it.

The description of $M$ requires $\log n + O(1)$ bits (for-loop structure and subroutine are constant size, independent of $n$). So if $x$ is the output of $M$, then $K_U(x) \leq \log n + O(1)$.

But $M$ outputs a string $x \in \{0,1\}^n$ of maximal Kolmogorov complexity. By our standard counting argument, we know $K_U(x) \leq |x| = n$, a contradiction.
HW3

1. Let \( A \) and \( B \) be decision problems and define

\[
A \oplus B \overset{\text{def}}{=} \{ x \mid x \text{ is in exactly one of } A, B \}
\]

Show that \( \text{NP} \cap \text{coNP} \) is closed under the \( \oplus \) operation.

Solution

We have that \( A, B \in \text{NP} \cap \text{coNP} \). So \( A, \overline{A}, B, \overline{B} \in \text{NP} \) and \( A, B, \overline{A}, \overline{B} \in \text{coNP} \). Consider

\[
A \oplus B = (A \cap \overline{B}) \cup (B \cap \overline{A})
\]

Each of the terms on the right-hand side is a language in \( \text{NP} \), and \( \text{NP} \) is closed under intersection and union, so the result is in \( \text{NP} \). Similarly every term on the right-hand side is in \( \text{coNP} \) so the result is in \( \text{coNP} \). Hence we have \( A \oplus B \in \text{NP} \cap \text{coNP} \).

2. Show that the following problem is \( \text{NP} \)-complete:

\[
\{ \phi \mid \phi \text{ is a satisfiable boolean formula in which no variable appears more than } 5 \text{ times} \}
\]

Solution

Do a Karp reduction from SAT to this new problem. The reduction is as follows. On input \( \phi \), for every variable \( x \) that appears \( k > 5 \) times, introduce \( k \) new variables \( x_1, \ldots, x_k \) and replace the \( i \)th occurrence of \( x \) with \( x_i \) instead. Then for each \( i \), add a clause to the formula that encodes relation \( x_i \Leftrightarrow x_{i+1} \). You can do this via \( (x_i \lor \overline{x_{i+1}}) \land (\overline{x_i} \lor x_{i+1}) \). Each new \( x_i \) occurs at most 5 times: 2 times in each of the 2 “connecting” clauses that it appears in, plus the 1 time when it replaces an occurrence of \( x \). Then you can show that the modified formula is satisfiable if and only if the old one is satisfiable.

3. Show that the following problem is \( \text{NP} \)-complete: given a set of linear inequalities over a set of variables \( x_1, \ldots, x_n \), determine whether there is an assignment of integers to the \( x_i \)'s that satisfies all inequalities.

Example:

\[
\begin{align*}
x_1 + 3x_2 - x_3 &= 4 \\
2x_2 - x_4 &= x_3 \\
x_1 + x_2 + x_3 + x_4 + x_5 &= 0
\end{align*}
\]

is satisfied by \( x_1 = 10; x_2 = x_3 = x_4 = x_5 = 0 \) (among many others).

Solution

Do a Karp reduction from 3SAT to this new problem. Given a boolean formula \( \phi \), do the following. For each variable \( x \) in \( \phi \), add inequalities "\( x \geq 0 \)" and "\( x \leq 1 \)". This ensures that in any solution to the inequalities, we have \( x \in \{0, 1\} \), representing T/F in the natural way.

For every clause \( (x_i \lor x_j \lor x_k) \) in \( \phi \), add an inequality "\( x_i + x_j + x_k \geq 1 \)". If a variable is negated within a clause, then replace that variable \( x \) with \( 1 - x \). For example, \( (x_i \lor \overline{x_j} \lor x_k) \) becomes "\( x_j + (1 - x_j) + x_k \geq 1 \)". It is then straightforward to check that the system of inequalities is satisfiable (over the integers) if and only if \( \phi \) is satisfiable.
HW4

1. (a) Prove that \( L \) is \( NP \)-complete if and only if \( L \) is \( coNP \)-complete (both notions of completeness under Karp reductions).

Solution

Suppose \( A \leq_p B \) via function \( f \). Then \( x \in A \iff f(x) \in B \). You can easily check for yourself that the same \( f \) also demonstrates \( A \leq_p B \) (just negate both sides of the if-and-only-if). So:

\[
\begin{align*}
L \text{ is } NP \text{-complete} & \iff \forall A \in NP : A \leq_p L \\
& \iff \forall A \in NP : A \leq_p \overline{L} \\
& \iff \forall A \in coNP : A \leq_p \overline{L} \\
& \iff \overline{L} \text{ is } coNP \text{-complete}
\end{align*}
\]

(b) Show that the following problem is \( coNP \)-complete:

\[ EQV = \{\langle \phi_1, \phi_2 \rangle \mid \phi_1 \text{ and } \phi_2 \text{ are logically equivalent boolean formulas} \} \]

For the purposes of this problem, two boolean formulas are logically equivalent if they have the same set of variables and they evaluate to the same result on all truth assignments.

Solution

From part (a) it suffices to show that \( \overline{EQV} \) is \( NP \)-complete. We will show \( SAT \leq_p \overline{EQV} \).

The Karp reduction is:

\[ f : \text{On input } \phi, \text{ output } \langle \phi, \text{FALSE} \rangle, \text{ where FALSE is some formula that is always } \overline{\text{false}} \text{ (like } x \land \overline{x}) \]

Then \( f \) is a valid Karp reduction from \( SAT \) to \( \overline{EQV} \) because:

\[
\begin{align*}
\phi \in SAT & \iff \text{there is a truth assignment } x \text{ for which } \phi(x) = T \\
& \iff \text{there is a truth assignment } x \text{ for which } \phi(x) \neq \text{FALSE} \\
& \iff \phi \text{ and FALSE are not equivalent} \\
& \iff \langle \phi, \text{FALSE} \rangle \in \overline{EQV} \\
& \iff f(\phi) \in \overline{EQV}
\end{align*}
\]

2. Define the complexity class \( DNP \) \( \text{def} \) \( \{L \mid \exists L_1 \in NP, L_2 \in coNP : L = L_1 \cap L_2 \} \). It might be a good idea to first really understand the difference between \( DNP \) and \( NP \cap coNP \).

(a) Show that \( NP \neq coNP \) if and only if \( DNP \neq NP \cap coNP \).

Solution

Suppose \( NP = coNP \). By definition, every language in \( DNP \) is of the form \( L_1 \cap L_2 \) where \( L_1 \in NP \) and \( L_2 \in coNP \). But now both \( L_1, L_2 \in NP = coNP \). Since \( NP \) and \( coNP \) are closed under intersection, \( L \in NP = coNP = NP \cap coNP \). This shows that \( DNP \subseteq NP \cap coNP \).

For the other direction, we first observe that \( NP \cup coNP \subseteq DNP \) (unconditionally). Indeed, take \( L \in NP \). Then \( L = L \cap \{0,1\}^* \). But this expression is the intersection of an
NP problem \( (L) \) and a coNP problem \( (\{0,1\}^*) \), so it is in DNP by definition. The same thing holds if \( L \in \text{coNP} \).

So now suppose DNP = NP \( \cap \) coNP. Then we have NP \( \cup \) coNP \( \subseteq \) DNP = NP \( \cap \) coNP. But NP \( \cup \) coNP \( \subseteq \) NP \( \cap \) coNP implies NP = coNP.

(b) Show that the following problem is DNP-complete under Karp reductions:

\[
\text{SATUNSAT} = \{ \langle \phi_1, \phi_2 \rangle \mid \phi_1 \text{ is satisfiable and } \phi_2 \text{ is not satisfiable} \}
\]

Solution The problem is in DNP since it can be written as:

\[
\{ \langle \phi_1 \rangle \mid \phi_1 \text{ is satisfiable} \} \cap \{ \langle \phi_2 \rangle \mid \phi_2 \text{ is not satisfiable} \}
\]

The first term is in NP and second term in coNP.

Now we must show that every \( L \in \text{DNP} \) reduces to SATUNSAT. Take an arbitrary \( L \in \text{DNP} \); it can be written as \( L = L_1 \cap L_2 \), where \( L_1 \in \text{NP} \) and \( L_2 \in \text{coNP} \). There is a Karp reduction \( f \) from \( L_1 \) to SAT, and a Karp reduction \( g \) from \( L_2 \) to \( \overline{\text{SAT}} \). From these we can construct:

\[
h(x) = \langle f(x), g(x) \rangle.
\]

I claim that \( h \) is a Karp reduction from \( L \) to SATUNSAT:

\[
x \in L \iff x \in L_1 \land x \in L_2
\]

\[
\iff f(x) \in \text{SAT} \land g(x) \in \overline{\text{SAT}}
\]

\[
\iff h(x) = \langle f(x), g(x) \rangle \in \text{SATUNSAT}
\]

3. The goal of this problem is to demonstrate that, if \( P = NP \), then not only can we solve NP decision problems in polynomial time, but we can also find witnesses in polynomial time.

(a) Suppose you have a subroutine that tells you whether a given boolean formula is satisfiable (i.e., it responds yes or no). Show how to use the subroutine to compute a satisfying assignment of any satisfiable boolean formula.

Solution Given a boolean formula \( \phi \), let \( x_1 \) denote its first variable, and let \( \phi|_{x_1=F} \) denote the formula you get by substituting variable \( x_1 = F \) in the formula and simplifying.

Query the subroutine on \( \phi|_{x_1=T} \) and \( \phi|_{x_1=F} \). If \( \phi \) is satisfiable, then at least one of these subformulas is satisfiable. If the first is satisfiable, then you know there is a satisfying assignment of \( \phi \) in which \( x_1 = T \). Recurse on \( \phi' = \phi|_{x_1=T} \) to obtain a satisfying assignment on \( \phi' \); add \( x_1 = T \) to that assignment and you've got a satisfying assignment to \( \phi \). Otherwise you recurse on \( \phi' = \phi|_{x_1=F} \) and augment its satisfying assignment with \( x_1 = F \).

(b) A 3-coloring of a graph is a way to color each vertex red, green, or blue so that no edge in the graph has endpoints with the same color. Determining whether a graph is 3-colorable is NP-complete.

Suppose you have a subroutine that tells you whether a given graph has a 3-coloring (i.e., it responds yes or no). Show how to use the subroutine to compute a 3-coloring of any 3-colorable graph (i.e., output a legal assignment of colors to the vertices).
Solution

Given \( G \), add three new vertices \( R, G, B \) to the graph and connect them with a triangle. Call the new graph \( G' \). There are no edges from the new vertices to those of \( G \), so \( G' \) is 3-colorable iff \( G \) is.

Now for a vertex \( v \) of \( G \), let \( G'|_{v=R} \) denote \( G' \) with edges \((v, G), (v, B)\) added. If \( G'|_{v=R} \) is 3-colorable, then there is a coloring of \( G' \) in which \( v \) has the same color as special vertex \( R \).

For each vertex \( v \) in \( G \), call the subroutine on \( G'|_{v=R}, G'|_{v=B}, G'|_{v=G} \). If \( G \) is 3-colorable the one of these subgraphs must be 3-colorable. Keep track of which color “works” for \( v \), then continue, setting \( G' = G'|_{v=\star} \). In this way, eventually all vertices of \( G \) will be adjacent to two among \( \{ R, G, B \} \). Then it is clear that we can color \( G \) using the only available color among \( \{ R, G, B \} \).