

Solutions for CS154 Final, Winter 2002-2003

Problem 1

(30 Points)

Grading: 3 points for Correct Answer, -1 point for Incorrect Answer and 0 points for no attempt.

Please circle your choice:

True / **False** There is no undecidable language L such that both L and \bar{L} are co-Turing-recognizable.

Both L and \bar{L} are co-Turing-recognizable $\Rightarrow \bar{L}$ and L are Turing-recognizable $\Rightarrow L$ is decidable.

True / **False** 3SAT is Turing-recognizable.

3SAT is actually decidable.

True / **False** If an NP-Hard problem can be solved in polynomial time, then $P = NP$.

A problem is NP-Hard if all problems in NP can be reduced to it. If we can solve this problem in polynomial time, then $P = NP$.

True / **False** If C is a CFL and $C \cap X$ is not a CFL, then it is possible that X is a CFL.

*Language $a^*b^nc^n$ is a CFL. Language $a^nb^nc^*$ is also a CFL. However, their intersection is $a^nb^nc^n$, which is not a CFL.*

True / **False** If R is regular and $R \cap X$ is regular, then it is possible that X is not a CFL.

R could be any finite set. X could be any non-CFL. Their intersection would be finite and hence, regular.

True / **False** There exists a non-regular language L such that L^* is regular.

Consider any non-regular language $L_1 = a^nb^n$. The language $L_2 = L_1 \cup \{a, b\}$ is also non-regular. However, L_2^ is regular.*

True / **False** The complement of a CFL is always Turing-recognizable

CFL's are decidable languages. Therefore, their complements are Turing-recognizable.

True / **False** If L is polynomial-time reducible to a finite language, then L is in P .

If L is polynomial-time reducible to a finite language, then L can be solved in polynomial-time. Clearly, $L \in P$.

True / **False** The number of co-Turing-recognizable languages is countably infinite.

A co-Turing-recognizable language is the complement of a Turing-recognizable language. There are countably infinite number of TM's \Rightarrow There are countably infinite number of Turing-recognizable languages \Rightarrow There are countably infinite number of co-Turing-recognizable languages.

True / **False** There is no undecidable language L such that neither L nor \bar{L} is Turing-recognizable.

Sipser provides examples of undecidable languages where neither L nor \bar{L} is Turing-recognizable.

Problem 2

(30 Points)

Grading: 2 Points for guessing the correct answer. **4 Points** for providing correct justification.

(a) We know that if language L is in P , then L^* is in P . Is the converse true as well?

Circle your Choice: **Yes** / **No**

Brief Justification: Let Σ denote the alphabet. Choose any undecidable language, say A_{TM} . Let $X = \Sigma \cup A_{TM}$. Then, X is still undecidable whereas $X^* = \Sigma^*$, which is clearly in P .

(b) A Non-deterministic Turing Machine T accepts a language that is infinite. T requires $\Omega(2^n)$ steps before it accepts a string of length n . If $P \neq NP$, is it possible that $L(T) \in P$?

Circle your Choice: **Yes** / **No**

Brief Justification: Although T takes exponential time to accept $L(T)$, it is quite possible that there is some other TM T' that accepts $L(T)$ in polynomial-time. Note that a language L belongs to P iff there *exists* a polynomial-time TM that accepts L .

(c) If R is regular, there exists an NFA with exactly three final states that accepts R .

Circle your Choice: Yes / No

Brief Justification: Let N denote an NFA that accepts R . Add three new states to N , draw ϵ -transitions from existing final states to these three new states, mark existing new states as non-final and mark the three newly added states as final. Clearly, the new machine is an NFA that accepts R .

(d) If A is an NP-hard problem, A is polynomial time reducible to B and B is in NP, then A is definitely NP-complete.

Circle your Choice: Yes / No

Brief Justification: A problem X is said to be NP-Complete if (a) all problems in NP are polynomial time reducible to X and (b) X itself is in NP. Since A is NP-Hard, condition (a) is satisfied. The polynomial time reducibility of A to an NP problem B is evidence that A is in NP. This shows that condition (b) is true. Therefore, A is NP-complete.

(e) If C is context free, and R is regular, then both $C - R$ and $R - C$ are context free languages.

Circle your Choice: Yes / No

Brief Justification: $C - R = C \cap \overline{R}$. The complement of R is regular. Intersection of CFL with a regular language is a CFL. Therefore, $C - R$ is a CFL. However, $R - C = R \cap \overline{C}$. If we let $R = \Sigma^*$, $R - C = \Sigma^* \cap \overline{C} = \overline{C}$. There are several CFL's whose complements are not CFLs. Therefore, $R - C$ is not always a CFL.

Problem 3

(20 Points)

Problem: Consider the language $A_{NFA} = \{\langle N, w \rangle \mid N \text{ is an NFA that accepts string } w\}$.

- a) Is A_{NFA} in NP? Prove your claim.
- b) Is A_{NFA} in P? Prove your claim.

Note: An NFA $N = (Q, \Sigma, \delta, s_0, F)$ is simply a graph with labels on vertices and edges. For an algorithm to qualify as being polynomial in the size of the input $\langle N, w \rangle$, the running time for the algorithm must be a polynomial in $|Q|$, $|\Sigma|$ and $|w|$, where $|w|$ is the length of string w .

Grading: 10 Points for showing that $A_{NFA} \in NP$. 10 Points for showing that $A_{NFA} \in P$.

If you show just $A_{NFA} \in P$ and say that $P \subseteq NP$, you get full 20 Points.

Solution: We will show that $A_{NFA} \in P$. This will prove $A_{NFA} \in NP$ also because $P \subseteq NP$.

```
BOOLEAN ATM(NFA N, STRING w)
{
    Let N = (Q, Σ, δ, s0, F)

    S ← {s0}
    FOR i = 1 TO |w| DO
        T ← φ
        FOREACH s ∈ S DO
            T ← T ∪ δ(s, w[i])
        S ← T
    IF (F ∩ S ≠ φ)
        RETURN TRUE
    ELSE
        RETURN FALSE
}
```

The outer loop iterates over w . In each iteration, set T is computed. This requires an iteration over $S \subseteq Q$. For each $s \in S$, we compute $T \cup \delta(s, w[i])$, whose time requirements are polynomial in $|T|$ and $|\delta|$. Since $T \subseteq Q$ and $|\delta| = O(\text{poly}(|Q|, |\Sigma|))$, the computation of $T \cup \delta(s, w[i])$ is polynomial in $|Q|$ and $|\Sigma|$. The actual time complexity depends on the machine model and the data structures we employ. The final step computes $F \cap S$, whose time requirements are polynomial in $|Q|$. Thus the overall algorithm is in P .

Common Pitfalls:

- The obvious algorithm for converting an NFA into a DFA and checking for membership is not polynomial-time in the size of the input. The reason is that there exist NFAs whose equivalent DFAs require an exponential number of states.
- The idea of *trying all paths* in the NFA to see if there is a path from the start state to some final state upon seeing string w is again exponential time.
- The correct approach is to begin with the start state, and hop from one set of states to another, as outlined above. A set is never more than $|Q|$ and the length of string is $|w|$, making this a polynomial-time algorithm. However, if you enumerate *all sets of states*, as happens when the equivalent DFA is constructed, the algorithm is exponential time.

Problem 4

(30 Points)

Problem: Consider the following two languages:

$$X = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are Turing Machines,} \\ \text{and } L(M_1) \text{ and } L(M_2) \text{ have at least 154 strings in common } \}.$$

$$Y = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are Turing Machines,} \\ L(M_1) \text{ and } L(M_2) \text{ have fewer than 154 strings in common } \}.$$

One of the above is Turing-recognizable and the other is not. Identify which one is which.

Prove your claims.

Grading: 15 Points for showing that X is Turing-recognizable. 15 Points for showing that X is undecidable.

Solution: There are several possible ways of showing that X is Turing-recognizable.

- Guess 154 (finite) strings. Simulate M_1 and M_2 on each string. If both machines accept all 154 strings, accept.
- The set $S = \{ \langle w, i \rangle \mid w \in \Sigma^*, i \in \mathcal{N} \}$, where \mathcal{N} denotes the set of natural numbers, is countably infinite. This means that there is a 1-1 mapping f between \mathcal{N} and S . A Turing Machine for X simply enumerates members of S ordered by the mapping f . For each member

$\langle w, i \rangle$, it first simulates M_1 on w for i steps and then M_2 on w for i steps. If both M_1 and M_2 accept within i steps, the TM adds string w to its collection of strings that it has verified as belonging to $L(M_1) \cap L(M_2)$. As soon as the size of this collection becomes 154, the TM halts in an accepting state.

- Since M_1 and M_2 are Turing-recognizable, they are *recursively enumerable*. Simply start listing the members of $L(M_1)$ and $L(M_2)$ (by interleaving the two enumerators), remembering strings in common. As soon as the size of this set becomes 154, accept.

Y is not Turing-recognizable. We prove by showing that Y is not decidable. Note that $Y = \overline{X \cup B}$, where B denotes wrongly-formatted strings (that are not proper encodings of pairs of TM's). We know that $X \cup B$ is Turing-recognizable (format checking is easy). The undecidability of Y will prove that Y is not Turing-recognizable.

Y is not decidable. There are several ways of showing this.

- Reduction from $A_{TM} = \{\langle M, w \rangle \mid M \text{ is a Turing Machine that accepts } w\}$. For convenience, we denote the TM for language A_{TM} by A_{TM} itself. Upon receiving $\langle M, w \rangle$, A_{TM} constructs two machines M_1 and M_2 as follows. The alphabet of M_1 and M_2 contains one extra symbol, say \pounds . Machine M_1 accepts all possible strings. Machine M_2 first checks whether the input consists solely of \pounds symbols and whether the length is at least 1 and at most 153. If the input string is one of these 153 strings, M_2 accepts. Otherwise, M_2 simulates M on w and accepts iff M accepts w . Thus, $\langle M_1, M_2 \rangle \in Y$ iff M does not accept w . If Y is decidable, then A_{TM} is decidable, a contradiction. Therefore, Y must be undecidable.
- Reduction from $A_{TM} = \{\langle M, w \rangle \mid M \text{ is a Turing Machine that accepts } w\}$. Upon receiving $\langle M, w \rangle$, A_{TM} constructs a new machine M' that first simulates M on w and accepts if M accepts w . Thus, M' accepts Σ^* if M accepts w , otherwise its language is empty. A_{TM} simply feeds $\langle M', M' \rangle$ to Y . If Y is decidable, then A_{TM} is decidable, a contradiction. Therefore, Y must be undecidable.

Common Pitfalls:

- (-5 Points) If you say that a TM for X simply starts enumerating strings in Σ^* lexicographically and for each string, checks whether both M_1 and M_2 accept, your solution is flawed. It might happen that M_1 and/or M_2 loop forever on some particular string. It is important to see how this is avoided. One way is to say that we maintain a counter c that counts from 1

to ∞ . For each c , we list all strings of length at most c and simulate M_1 and M_2 for *at most* c steps per string.

- If you don't show the reduction but present the big picture for part (ii) (by saying that you could reduce some undecidable language to Y (or X) to show that Y is undecidable and hence could conclude that Y is not Turing-recognizable), you get 5 points.

Problem 5

(10 Points)

Problem: Consider the following two languages:

$$\begin{aligned} \text{LARGE_FACTOR} &= \{ \langle n, t \rangle \mid n \text{ and } t \text{ are positive integers, } t < n, \\ &\quad \text{and } n \text{ has a factor } f \text{ satisfying } t \leq f < n \} \\ \text{PRIME} &= \{ n \mid \text{where } n \text{ is a positive integer that is also prime, } n \geq 2 \} \end{aligned}$$

Part (a) Show that LARGE_FACTOR is in NP.

Part (b) What exactly is the flaw in the following argument:

“A Turing Machine for PRIME receives an integer n as input. It rejects the input if $n < 2$. If $n = 2$, the input is accepted. Otherwise, the Turing Machine invokes LARGE_FACTOR($n, 2$) and accepts iff LARGE_FACTOR rejects. Thus PRIME is polynomial-time reducible to LARGE_FACTOR. Since LARGE_FACTOR is known to be in NP, we can conclude that PRIME is in NP.”

Grading: 2 Points for Part (a). 8 Points for Part (b).

Solution: *Part (a)* Just guess the factor f and verify that f divides n and $t \leq f < n$. The solution is actually provided in HW #6 solutions.

Part (b) Note the phrase “... *accepts iff* LARGE_FACTOR *rejects*”. The reduction is indeed polynomial time. However, we have reduced PRIME to the *complement* of LARGE_FACTOR. Since LARGE_FACTOR is in NP, we have shown that PRIME is in co-NP. It is actually an open research problem whether $\text{NP} = \text{co-NP}$.

Only a handful of students identified the flaw.

Problem 6

(20 Points)

Problem: Let $X = \{ \langle G_1, G_2 \rangle \mid G_1 \text{ and } G_2 \text{ are Context-Free Grammars, and } L(G_1) \text{ and } L(G_2) \text{ have at least 2 strings in common.} \}$

Reduce PCP (Post's Correspondence Problem) to language X to show that X is undecidable.

Grading: 20 Points for showing the reduction and proving it correct.

Solution: The input to PCP is a set of k pairs of strings $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)$, where each string belongs to Σ^* for some alphabet Σ . The input is accepted iff there exists a finite non-empty sequence $\langle i_1, i_2, \dots, i_n \rangle$ of indices such that the strings $x_{i_1}x_{i_2}x_{i_3} \dots x_{i_n}$ and $y_{i_1}y_{i_2}y_{i_3} \dots y_{i_n}$ are identical.

There are several ways of solving this:

- CFG G_1 has the productions $S \rightarrow \epsilon \mid x_1Sc_1 \mid x_2Sc_2 \mid x_3Sc_3 \mid \dots \mid x_kSc_k$, where $c_1, c_2, c_3, \dots, c_n$ are new characters that do not belong to Σ . CFG G_2 has productions $S \rightarrow \epsilon \mid y_1Sc_1 \mid y_2Sc_2 \mid y_3Sc_3 \mid \dots \mid y_kSc_k$, where $c_1, c_2, c_3, \dots, c_n$ are the same characters described above. Both G_1 and G_2 accept the empty string. They both accept a second string in common iff PCP has a solution. The new characters c_1, c_2, \dots, c_k make sure that both the number and ordering of (top and bottom parts of) dominoes is identical.
- CFG G_1 has productions $S \rightarrow \# \mid x_1Sy_1^R \mid x_2Sy_2^R \mid \dots \mid x_kSy_k^R$. CFG $G_2 = \{w\#w^R \mid w \in \Sigma^*\}$. Both G_1 and G_2 accept the string $\#$. They accept another string in common iff PCP has a solution.

Common Pitfalls:

- (-5 Points) It is crucial that the number and ordering of dominos be remembered. The following grammars are wrong: Let G_1 have the productions $S \rightarrow \epsilon \mid y_1S \mid y_2S \mid y_3S \mid \dots \mid y_kS$, Let G_2 have the productions $S \rightarrow \epsilon \mid x_1S \mid x_2S \mid x_3S \mid \dots \mid x_kS$. Now, although a string might be common, it could be the result of a different number or ordering of top and bottom parts of the dominoes.
- (-5 Points) The problem asks for a proof that PCP has a solution (i.e, has *at least one string* that can be made by juxtaposing dominoes) iff G_1 and G_2 have *at least two strings* in common. The connection between *one string* for PCP and *two strings* for CFG's must be brought out.

Problem 7

(20 Points)

Problem: Let

LARGE-CUT = $\{\langle G, k \rangle \mid G \text{ is a graph with a cut of width at least } k\}$

LARGE-BISECTION = $\{\langle G, k \rangle \mid G \text{ is a graph with a bisection of width at least } k\}$

SMALL-BISECTION = $\{\langle G, k \rangle \mid G \text{ is a graph with a bisection of width at most } k\}$

Definitions: Let $G = (V, E)$ denote an undirected graph.

A *cut* of G is a partition of V into two disjoint subsets i.e., $V = V_1 \cup V_2$ such that $V_1 \cap V_2 = \phi$.

A cut is said to be a *bisection* if the partitions are equi-sized, i.e., $|V_1| = |V_2| = |V|/2$.

The *width* of a cut is the number of edges in E that connect pairs of vertices in different partitions. Formally, the width is the size of the set $\{(v_1, v_2) \mid (v_1, v_2) \in E, v_1 \in V_1 \text{ and } v_2 \in V_2\}$.

LARGE-CUT is known to be an NP-Complete problem.

Part (a) Show that LARGE-BISECTION is NP-Complete.

Part (b) Show that SMALL-BISECTION is NP-Complete.

Grading: 2 Points for showing LARGE-BISECTION is in NP. 8 Points for showing a reduction from LARGE-CUT (or any other NP-Complete language) to LARGE-BISECTION.

2 Points for showing SMALL-BISECTION is in NP. 8 Points for showing a reduction from LARGE-BISECTION (or any other NP-Complete language) to SMALL-BISECTION.

Solution: Showing that LARGE-BISECTION \in NP is straightforward: Just guess the partitions and verify that the cut width is at least k . For SMALL-BISECTION, verify that the cut width is at most k .

Part (a): There are several ways of proving LARGE-BISECTION NP-Complete:

- Reduction from LARGE-CUT to LARGE-BISECTION: Let $\langle G = (V, E), k \rangle$ be the input to LARGE-CUT. Add $|V|$ isolated vertices to G to create a new graph G' . Then it is easy to prove that LARGE-CUT(G, k) is true iff LARGE-BISECTION(G', k) is true.
- Reduction from LARGE-CUT to LARGE-BISECTION: Simply make two copies of G . The new graph (with twice as many nodes and edges) has a bisection of width at least $2k$ iff the original graph has a cut of width at least k .
- Reduction from \neq -SAT to LARGE-BISECTION: If you read HW # 6 solutions carefully, you would notice that the graph created in the reduction from \neq -SAT to MAX-CUT actually achieves its maximum cut size only when it is a bisection.

Part (b): There are several ways of proving SMALL-BISECTION NP-Complete:

- Reduction from LARGE-BISECTION to SMALL-BISECTION: Let $G'_{\text{complement}}$ denote the complement of G' . Then LARGE-BISECTION(G', k) is true iff SMALL-BISECTION($G'_{\text{complement}}, |V|^2/4 - k$) is true.
- Reduction from LARGE-CUT to SMALL-BISECTION: This is simply the two reductions described earlier, packaged into one. Just add $|V|$ isolated vertices (or duplicate the graph), and then take the complement of this graph. Then LARGE-CUT(G', k) is true iff SMALL-BISECTION($G'_{\text{complement}}, |V|^2 - k$) is true.
- Reduction from MAX-CUT to SMALL-BISECTION: The problem MAX-CUT was described in one of the HW Problems (in # HW 6). Just add $|V|$ isolated vertices to the graph. Then, MAX-CUT(G, k) is true iff SMALL-BISECTION($G, |V|^2 - k$) is true.

Common Pitfalls:

- The following idea is flawed: *If* SMALL-BISECTION($G, k - 1$) *rejects*, LARGE-BISECTION(G, k) *accepts*. The flaw is the same as in Problem 5 of this final. Please study that solution to understand what's wrong with this idea.
- Nobody caught the mistake in the definition of SMALL-BISECTION. Actually, as worded above, it allows for a partition in which one of the sets is empty which makes SMALL-BISECTION(G, k) true for all graphs with $k \geq 0$. Neither V_1 nor V_2 should be empty. Everybody got the idea, though.

Problem 8

(20 Points)

Problem: Let $3\text{-COLOR} = \{ \langle G \rangle \mid G \text{ is a 3-colorable graph} \}$

A graph is said to be k -colorable if its vertices can be assigned one of k distinct colors such that no edge connects vertices with same color.

Describe a polynomial-time reduction from 3-COLOR to 3-SAT. Prove that your reduction is indeed polynomial time.

Note: In class, we saw a reduction from any NP problem to 3SAT. Dont write that proof. For this problem, imagine that the year is 1969 when Cook-Levin Theorem (and the construction therein) was not known.

Grading: 20 Points for showing the reduction and proving that it works.

Solution: For each node v in G , there are three variables, namely v_R, v_G, v_B , denoting three possible colors R, G and B . Exactly one of v_R, v_G and v_B must be true. This is equivalent to satisfying the formula $(v_R \vee v_G \vee v_B) \wedge (v_R \Rightarrow \overline{v_G} \wedge \overline{v_B}) \wedge (v_G \Rightarrow \overline{v_B} \wedge \overline{v_R}) \wedge (v_B \Rightarrow \overline{v_R} \wedge \overline{v_G})$. This expression can be converted into 3-SAT format by observing that $(g \Rightarrow hk) \equiv (\overline{g} \vee h) \wedge (\overline{g} \vee k)$.

For each edge, we need to enforce that its two end-points do not have the same color. For example, for an edge (u, v) , this is equivalent to satisfying $(\overline{u_R} \vee \overline{v_R}) \wedge (\overline{u_G} \vee \overline{v_G}) \wedge (\overline{u_B} \vee \overline{v_B})$

Taking the AND of all formulas we just described, we obtain a reduction to 3-SAT.

Common Pitfalls:

- Assigning *one boolean variable* to each node and trying to come up with some formula with these variables. It is not clear what the connection between the truth values of this variable and 3-colors is. To represent k colors, we need $\approx \log k$ bits per node.
- There are many ways to assert that (a) every node must have a unique color, and (b) nodes connected by an edge must have different colors. It's okay to derive a formula that is not in 3SAT format because your formula will have $O(1)$ size and can easily be massaged into product-of-sums.