Practice Final Exam — Comments

Here we provide some solutions and discussion on selected problems on the practice final exam (including for all the problems for which students asked questions).

True, False, or Unknown

1. For each of the following, circle one of the choices to indicate whether the statement is known to be *True*, is known to be *False*, or *Unknown* if its validity depends on something that is either currently unknown or not specified in the question.

Then, write a short justification to support your answer. When your answer is *Unknown*, your answer should make it clear what unknown the validity of the statement depends on (for example, that it is equivalent to a statement whose truth is currently unknown to anyone).

(a) The function, $XOR : \{0,1\}^* \to \{0,1\}$, which outputs the logical exclusive or of all the input bits, is in the complexity class P.

Circle one:

True *False* *Unknown*

Justification ($\leq$ 5 words):

We saw a finite automaton that implements the infinite XOR function, and any finite automaton can be simulated in polynomial time. Simulating each step involves a lookup in a table which is constant time since the size of the table does not depend on the size of the input, only on the size of the transition function, and other constant time operations. The number of steps is linear in the size of the input, since each input symbol is processed once.

(b) The function, XOR (from the previous question), is in the complexity class NP.

Circle one:

True *False* *Unknown*

Justification (3 symbols):

$P \subseteq NP$.

(c) The function, XOR (from the previous question), is in the complexity class NP-Complete.

Circle one:

True *False* *Unknown*

Justification ($\leq$ 15 words):

The answer depends on whether or not P $=$ NP.

(d) If a function A in NP has an exponential lower bound (e.g., requires $\Omega(2^n)$ time to compute), then no function in NP-Complete can be computed in polynomial time.

Circle one:

True *False* *Unknown*

Justification ($\leq$ 3 sentences):

If we prove that some $A \in$ NP requires exponential time to compute, this means that P $\neq$ NP and that no function in NP-Complete can be computed in polynomial time. (This was discussed in more detail in class on Dec 1.)

(e) If a function $Q : \{0,1\}^* \rightarrow \{0,1\}$ is computable, the function $\overline{Q}$ is computable where $\forall x \in \{0,1\}^* : \overline{Q}(x) = \mathsf{NOT}(Q(x))$.

Circle one:

True *False* *Unknown*

Justification ($\leq$ 3 sentences):

Since Q is computable, there is some TM T_Q that computes Q. We can construct a machine T_{NQ} that computes $\overline{Q}$ by flipping the final and non-final states of T_Q.

Proving Uncomputability

2. In this question, your goal is to show that the function $CELL_{15}$ defined below is uncomputable.

> **Input:** A string w that describes a Turing Machine.
>
> **Output:** **1** if the machine described by w would write a 1 on the fifteenth cell on its tape when executed on a tape that is initially all blank. Otherwise, **0**.

That is, a machine which computes $CELL_{15}$ outputs **1** when the input describes a Turing Machine which, when run on a blank tape, at some points writes the symbol 1 to the tape cell at index 15 (counting from the start-of-tape symbol at index 0).

(a) Which strategy would show that $CELL_{15}$ is uncomputable? (Circle one, no explication needed.)

Use a machine that computes $CELL_{15}$ to compute $HALTS$.

Use a machine that computes $HALTS$ to compute $CELL_{15}$.

(b) Employ the strategy you chose in the previous question to show that $CELL_{15}$ is uncomputable.

Countable, Uncountable, Unknown

3. For each set described below, indicate whether its cardinality is *Countable*, *Uncountable*, or *Unknown* (not determined by the question if it is countable or uncountable). Circle one option and give a proof of your answer.

(a) The set of all grades that students will get on the final exam.

Countable *Uncountable* *Unknown*

Proof: It is finite, and all finite sets are countable.

(b) The set of NAND circuits that compute XOR.

Countable *Uncountable* *Unknown*

Proof: This is a subset of the set of all NAND circuits. We set of all NAND circuits is countable. We could prove this by showing a way to map all NAND circuits to a unique natural number.

(c) The set of of all uncomputable languages.

Countable *Uncountable* *Unknown*

Proof: (Hint) Think of the languages as subsets of the natural numbers, or as functions from $\{0,1\}^* \rightarrow \{0,1\}$.

Always, Sometimes, Never

4. For a function $f : \{0,1\}^{3102} \rightarrow \{0,1\}$ that can be implemented by a NAND circuit with s gates, which of the statements that follow would be *Always True*, *Possibly True* (meaning there are some functions f for which the statement is true and others for which is it false), or *Never True* (circle one option). Give a brief statement to justify your answer.

(a) f is computable

Always True *Possibly True* *Never True*

Justification ($\leq$ 5 words): (Hint) It is finite.

(b) $f \in \mathsf{NP}$

Always True *Possibly True* *Never True*

Justification ($\leq$ 5 words): (Hint) $\mathsf{P} \subseteq \mathsf{NP}$.

(c) f can be implemented using 3102 NAND gates.

Always True *Possibly True* *Never True*

Justification ($\leq$ 3 sentences): Depends on s (and sensible functions with 3102 input bits cannot be implemented with 3012 NAND gates since most input bits cannot affect the output).

Induction

5. Define the function $\texttt{ALT}_n : \{0,1\}^{2n} \to \{0,1\}$ such that for a string $w \in \{0,1\}^{2n}$ we say that $\texttt{ALT}_n(w) = 1$ provided $w \in (01)^*$. We could compute $\texttt{ALT}_1$ using the following straightline program:

```
def ALT1(x1,x2):
    diff = XOR(x1,x2)
    return AND(x2, diff)
```

We could then implement $\texttt{ALT}_n$ as follows:

```
def ALTn(x1,x2,...,x2n):
    diff = XOR(x1,x2)
    first = AND(x2,diff)
    rest = ALTn(x3,...,x2n)
    return AND(first, rest)
```

Suppose we have similarly implemented $\texttt{ALT}_{n-1}$, $\texttt{ALT}_{n-2}$, etc., (and all other dependent subroutines).

Show that the number of NAND gates needed to represent a circuit for $\texttt{ALT}_n$ is no more than $10n$ gates (hint: XOR requires 4 NAND gates and AND requires 3 NAND gates).

Comments. We won't provide a written solution to this (no one asked about it), but you should all be able to construct a solid proof by induction. Four main things to remember:

1. First, state what you are proving clearly. (This is true for all proofs!)
2. Then, state the induction hypothesis, $P(n)$ where n is an input natural number, and your goal is to prove $\forall n \in \mathbb{N} : P(n)$ to prove the theorem from the first step. (For some problems you will have to think carefully if you need $\forall n \in \mathbb{N}$ or some subset of $\mathbb{N}$ or some other countable set.
3. Prove the base case: $P(0)$.
4. Prove the inductive case: $P(n) \implies P(n+1)$. You should think carefully about which values n this should apply for. Also, in some cases it may be easier to prove $P(n-1) \implies P(n)$.

Complexity Classes

6. For an arbitrary given function A, for each of the complexity classes below, describe a way to prove that A belongs to the given class.

(a) P

Hint: show an algorithm for a deterministic TM.

(b) NP

Hint: show an algorithm connected to definition of NP.

(c) NP-Hard

Hint: this is a *hardness* class, unlike the first two sub-questions which are about *easiness* classes. To prove a problem is hard, we usually use a reduction to show that if it could be solved we could use it to solve some other problem we already know is hard.

(d) NP-Complete

Hint: you should understand the definition of NP-Complete. and be able to solve this using parts (b) and (c).

(e) $O(n^2)$ (where n is the length of the input to A)

Hint: show an algorithm with particular properties.

(f) $\Theta(1)$

Hint: this is a *tight* bound, so need to show both "easiness" (there is an algorithm with some property) and "hardness" (there isn't a better one). For constant running time, though, showing "hardness" should be easy (but for most other running times, that is the hard, and often unknown, part).

(g) $\Omega(n)$ (where n is the length of the input to A)

See hint above — $\Omega(n)$ is a hardness property, meaning we there is no algorithm that solves the problem with asymptotic complexity that is less than linear in the input length.

7. Prove the following: If a function A in NP has an exponential lower bound (e.g. requires $\Omega(2^n)$ time to compute), then no language in NP-Complete can be computed in polynomial time.

Hints: review the definition of NP-Complete (and problem 1d).

Regular Expressions and Automata

8. For the following 4 sub-problems you will be asked to get both a regular expression and a finite state automaton for two different languages.

(a) Draw a finite state automaton (either an NFA or DFA) for the language:

$$\{x \in \{0,1\}^* \mid x \text{ as interpreted as a binary representation of a natural number is odd}\}$$

(note that the empty string is a binary representation of 0, which is even).

(b) Give a regular expression for the language:

$$\{x \in \{0,1\}^* \mid x \text{ as interpreted as a binary representation of a natural number is odd}\}$$

(note that the empty string is a binary representation of 0, which is even).

(c) Draw a finite state automaton (either an NFA or DFA) for the language: $\texttt{XOR} : \{0,1\}^* \to \{0,1\}$. In other words, the language $\{x \in \{0,1\}^* \mid \texttt{XOR}(x) = 1\}$.

(d) Give a regular expression for the language: $\texttt{XOR} : \{0,1\}^* \to \{0,1\}$. In other words, the language $\{x \in \{0,1\}^* \mid \texttt{XOR}(x) = 1\}$.

Comments. Not providing solutions for these, but you should be able to easily check if your answers are correct by simulating it on representative inputs.

Models

9. List the essential things that are required to define a model of computing.

Comments. There are two different ways you could answer this question. One is answering what a mathematical model (in general) must do: (1) provide a description of all the objects it covers, and (2) provide a way of mapping those objects to a meaning. For computing models, (1) means describing a set of machines or circuits or some other computing representation using a formal notation. (2) means showing how the computing model executes to produce (or not produce) an output for a given input.

A second, and more straightforward, interpretation of the question is about what components are needed for any sensible computing model:

1. A way to describe the **input**.
2. A way to interpret the **output**.
3. A way to do **processing**.
4. (Not completely essential, but hard to have interesting computing models without it:) **memory** (some way to keep track of things are processing is done).

For all of the computing models we have seen in class (as well as new ones that might be presented to you), you should be able to identify these four components.

10. Describe how to show that two models of computing are equivalent.

Comments. Should consider first what "equivalent" means. Usually, we mean equivalent in *power*, ignoring differences in cost, expressiveness, smell, color, taste, etc.

Two models are equivalent if the set of functions they can compute are equivalent. To complete the answer, explain how to show two sets are equivalent ($A \subseteq B$ and $B \subseteq A \implies \text{A} \equiv B$).

11. A Turing Machine's configuration contains all the information needed to describe the current status of its computation (i.e., if I paused my computation then wrote the configuration down, I could resume the computation using what I had written). List all the necessary components of a Turing Machine's configuration.

Comments. This was a cohort problem, so you should already have a good answer to it.

Asymptotics

12. Let $f(n) = 8n^{4.5}$ and $g(n) = 5n^5$, which of the following are true? Support your answer to each part with a convincing argument.

(a) $f \in O(g)$

True

(b) $f \in \Omega(g)$

False

(c) $f \in \Theta(g)$

False

