

Extra Practice Problems 9

Here's a final batch of practice problems. Solutions are available in the solutions drawer in the Gates building.

Problem One: Set Theory

Prove or disprove: there are sets A and B where $\wp(A \times B) = \wp(A) \times \wp(B)$.

Problem Two: Induction

The *well-ordering principle* states that if $S \subseteq \mathbb{N}$ and $S \neq \emptyset$, then S contains an element n_0 that is less than all other elements of S . There is a close connection between the well-ordering principle and the principle of mathematical induction.

Suppose that P is some property such that

- $P(0)$
- $\forall k \in \mathbb{N}. (P(k) \rightarrow P(k+1))$

Using the well-ordering principle, *but without using induction*, prove that $P(n)$ holds for all $n \in \mathbb{N}$. This shows that if you believe the well-ordering principle is true, then you must also believe the principle of mathematical induction.

Problem Three: Graphs

Let $G = (V_1, E_1)$ and $H = (V_2, E_2)$ be undirected graphs. The *tensor product* of G and H , denoted $G \times H$, is an undirected graph. $G \times H$ has as its set of nodes the set $V_1 \times V_2$. The edges of $G \times H$ are defined as follows: the edge $\{(u_1, v_1), (u_2, v_2)\}$ is in $G \times H$ if $\{u_1, u_2\} \in E_1$ and $\{v_1, v_2\} \in E_2$.

Prove that $\chi(G \times H) \leq \min\{\chi(G), \chi(H)\}$.

Interestingly, the following question is an open problem: are there any undirected graphs G and H for which $\chi(G \times H) \neq \min\{\chi(G), \chi(H)\}$? A conjecture called *Hedetniemi's conjecture* claims that the answer is no, but no one knows for sure!

Problem Four: First-Order Logic

Consider the following formula in first-order logic:

$$\forall x \in \mathbb{R}. \forall y \in \mathbb{R}. (x < y \rightarrow \exists p \in \mathbb{Z}. \exists q \in \mathbb{Z}. (q \neq 0 \wedge x < p/q \wedge p/q < y))$$

This question explores this formula.

- i. Translate this formula into plain English. As a hint, there's a very simple way of expressing the concept described above.
- ii. Rewrite this formula so that it doesn't use any universal quantifiers.
- iii. Rewrite this formula so that it doesn't use any existential quantifiers.
- iv. Rewrite this formula so that it doesn't use any implications.
- v. Negate this formula and push the negations as deep as possible.

Problem Five: Functions

A function $f : A \rightarrow A$ is called an *involution* if $f(f(x)) = x$ for all $x \in A$.

Prove that if f is an involution, then f is a bijection.

Problem Six: Binary Relations

Suppose that R and S are binary relations over some set A . The *composition of R and S* , denoted $S \circ R$, is the relation defined as follows:

$$a(S \circ R)b \text{ if there is some } c \text{ such that } aRc \text{ and } cSb.$$

One interesting special case to consider is the composition of a relation with itself. Given a binary relation R , the relation $R \circ R$ is defined as follows:

$$a(R \circ R)b \text{ if there is some } c \text{ such that } aRc \text{ and } cRb.$$

We use the notation R^2 to denote $R \circ R$.

- i. Prove or disprove: if R is an equivalence relation over a set A , then $R^2 = R$. (Two relations S and T are equal to one another if the following claim holds: $\forall a \in A. \forall b \in A. (aSb \leftrightarrow aTb)$)
- ii. Prove or disprove: if R is a partial order relation over a set A , then $R^2 = R$.

Problem Seven: The Pigeonhole Principle*

Let n be an odd natural number and consider the set $S = \{1, 2, 3, \dots, n\}$. A *permutation* of S is a bijection $\sigma : S \rightarrow S$. In other words, σ maps each element of S to some unique element of S and does so in a way such that no two elements of S map to the same element.

Let σ be an arbitrary permutation of S . Prove that there is some $r \in S$ such that $r - \sigma(r)$ is even.

* Adapted from http://www.cut-the-knot.org/do_you_know/pigeon.shtml.

Problem Eight: DFAs, NFAs, and Regular Expressions

This question explores transformations between different representations of regular languages and some nuances of how they work.

- i. Let $\Sigma = \{a, b\}$. Design a small NFA for the language $\{ w \in \Sigma^* \mid w \text{ ends in } aba \text{ or } bbba \}$. Make your NFA actually use nondeterminism as part of its operation; don't just design a DFA.
- ii. Using the subset construction, convert the NFA you designed in part (i) into a DFA.
- iii. Using the state elimination algorithm, convert the NFA you designed in part (i) into a regular expression.
- iv. Briefly explain why the DFA you came up with in part (ii) is not the smallest possible DFA for the language.
- v. Show that the regular expression you came up with in part (iii) is not the shortest possible regular expression for the language.

Problem Nine: Nonregular Languages

Prove that the language $\{ w \in \{a, b\}^* \mid |w| \equiv_3 0 \text{ and the middle third of the characters in } w \text{ contains at least one } a \}$ is not regular.

Problem Ten: Context-Free Grammars

Let $\Sigma = \{ (,) \}$ and let $L = \{ w \in \Sigma^* \mid w \text{ is a string of balanced parentheses and } w \text{ has an even number of open parentheses} \}$. Write a CFG for L .

Problem Eleven: Turing Machines

Let $L = \{ w^2w \mid w \in \{a, b\}^* \}$. Design a TM that's a decider for L at the level of individual states and transitions.

Problem Twelve: R and RE Languages

A *computable function* is a function $f : \Sigma^* \rightarrow \Sigma^*$ with the following property: there is a TM M that, when given w on its input tape, always halts with $f(w)$ on its tape.

Given any computable function f and language L , we define $f(L) = \{ w \in \Sigma^* \mid \exists x \in L. f(x) = w \}$. In other words, $f(L)$ is the set of strings formed by applying f to each string in L .

Prove that if $L \in \mathbf{RE}$ and f is a computable function, then $f(L) \in \mathbf{RE}$.

Problem Thirteen: Impossible Problems

Prove that $L = \{ \langle M, N \rangle \mid M \text{ is a TM, } N \text{ is a TM, and } \mathcal{L}(M) = \overline{\mathcal{L}(N)} \}$ is not in **RE**.

Problem Fourteen: P and NP

Suppose that V is a polynomial-time verifier for an **NP**-complete language L . That is,

$$w \in L \text{ iff } \exists c \in \Sigma^*. V \text{ accepts } \langle w, c \rangle$$

and

$$V \text{ runs in time polynomial in } |w|$$

Now, suppose that V is a “superverifier” with the property that for any string $w \in L$, V accepts $\langle w, c \rangle$ for almost all choices of c . Specifically, for any $w \in L$, there are at most five strings c for which V rejects $\langle w, c \rangle$.

Under these assumptions, prove that **P** = **NP**.