CS103 Handout 37
Fall 2015 November 23, 2015

Extra Practice Problems 8

Here's another batch of practice problems to work through. Please let us know if there are any topics you'd specifically like some more practice with. We'd be happy to provide extra practice problems on those topics!

Problem One: Set Theory

If $A \subseteq \mathbb{N}$ and $B \subseteq \mathbb{N}$, then the ***Minkowski sum*** of A and B, denoted $A + B$, is the set

$$A + B = \{ m + n \mid m \in A \text{ and } n \in B \}$$

This question explores properties of the Minkowski sum.

i. Prove or disprove: $|A + B| = |A| + |B|$ for all sets $A \subseteq \mathbb{N}$ and $B \subseteq \mathbb{N}$.
ii. What is $\mathbb{N} + \mathbb{N}$? Prove it.

Problem Two: Induction

Let $k \geq 1$ be any natural number. Prove, by induction, that $(k+1)^n - 1$ is a multiple of k for all $n \in \mathbb{N}$.

Problem Three: Graphs

Let G be a connected, undirected graph with $n \geq 2$ nodes.

i. Prove that $\alpha(G) \leq n - 1$. (Recall that $\alpha(G)$ is the independence number of G, the size of the largest independent set in G).
ii. Prove or disprove: for any $n \geq 2$, there is an undirected, connected graph G with n nodes where $\alpha(G) = n - 1$.

Problem Four: First-Order Logic

Given the predicates

- *TM*(M), which states that M is a TM;
- *String*(w), which states that w is a string; and
- *Accepts*(M, w), which states that M accepts w,

Write a statement in first-order logic that says "the **RE** languages are closed under union."

Problem Five: Functions

This question explores properties of special classes of functions.

i. Prove or disprove: if $f : \mathbb{R} \to \mathbb{R}$ is a bijection, then $f(r) \geq r$ for all $r \in \mathbb{R}$.

ii. Prove or disprove: if $f : \mathbb{N} \to \mathbb{N}$ is a bijection, then $f(n) = n$ for all $n \in \mathbb{N}$.

iii. Prove or disprove: if $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ are bijections, then the function $h : \mathbb{R} \to \mathbb{R}$ defined as $h(x) = f(x) + g(x)$ is also a bijection.

Problem Six: Binary Relations

If R is a binary relation over a set A, the complement relation R^c is the binary relation over A defined as follows:

$$a\overline{R}b \quad \text{if} \quad \neg aRb$$

In other words, $a\overline{R}b$ is true precisely if aRb is false.

i. Prove or disprove: if R is a binary relation over a nonempty set A, then ***at least*** one of R or $\overline{R}$ is a strict order.

ii. Prove or disprove: if R is a binary relation over a nonempty set A, then ***at most*** one of R or $\overline{R}$ is a strict order.

Problem Seven: The Pigeonhole Principle

Suppose that n people are seated at a round table at a restaurant. Each of the n people orders a different entree for dinner. The waiter brings all of the entrees out and places one dish in front of each person. Oddly enough, the waiter doesn't put anyone's dish in front of them.

Prove that there is some way to rotate the table so that at least two people have their entree in front of them.

Problem Eight: DFAs, NFAs, and Regular Expressions

Consider the following language over $\Sigma = \{ \texttt{0}, \texttt{E} \}$:

$$\begin{aligned} PARITY = \{ \ w \mid \ & w \text{ has even length and has the form } \texttt{E}^n \text{ or} \\ & w \text{ has odd length and has the form } \texttt{0}^n \ \} \end{aligned}$$

For example, `EE` $\in$ *PARITY,* `00000` $\in$ *PARITY*, `EEEE` $\in$ *PARITY*, and $\varepsilon \in$ *PARITY*, but `EEE` $\notin$ *PARITY*, `E0` $\notin$ *PARITY,* and `0000` $\notin$ *PARITY.*

i. Write a regular expression for *PARITY*.

ii. Design a DFA that accepts *PARITY*.

Problem Nine: Nonregular Languages

Let $\Sigma = \{\texttt{a}, \texttt{b}\}$ and let $L = \{ w \in \Sigma^* \mid w$ has the same number of `a`'s and `b`'s and $|w| \geq 10^{100} \}$.

i. Prove or disprove: L is not a regular language.

ii. Prove or disprove: there is at least one infinite subset of L that is regular.

Problem Ten: Context-Free Grammars

Let $\Sigma = \{\texttt{a}, \texttt{b}\}$ and let $L = \{ w \in \Sigma^* \mid w$ is a palindrome and w contains `abba` as a substring $\}$. Write a context-free grammar for L.

Problem Eleven: Turing Machines

Let $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{=}\}$. Draw the state-transition diagram of a TM for the $\{ w\texttt{=}w \mid w \in \{\texttt{a}, \texttt{b}\}^* \}$.

Problem Twelve: R and RE Languages

Earlier in this packet of problems you translated the statement "the **RE** languages are closed under union" into first-order logic. It turns out that this statement is true, but a bit trickier to prove that you might expect.

If we take a language $L \in$ **RE**, we know that we can get a recognizer M for it. A recognizer for L, in software, would be a function

```
bool inL(string w)
```

that takes as input a string w. If $w \in L$, then `inL(w)` returns true. If $w \notin L$, then `inL(w)` *may* return false, or it may loop infinitely.

Let L_1 and L_2 be **RE** languages. Below is an ***incorrect*** construction that purportedly is a recognizer for $L_1 \cup L_2$:

```
bool inL1uL2(string w) {
     return inL1(w) || inL2(w);
}
```

Here, `inL1` and `inL2` are recognizers for L_1 and L_2, respectively.

i. Give concrete examples of languages L_1 and L_2 and implementations of methods `inL1` and `inL2` such that the above piece of code is not a recognizer for $L_1 \cup L_2$. Justify your answer.

To show that the **RE** languages are closed under union, it's easiest to think about combining together two verifiers for the input languages to produce a verifier for their union.

ii. Using the verifier definition of **RE**, prove that the **RE** languages are closed under union.

Problem Thirteen: Impossible Problems

Let $\Sigma = \{\texttt{a}, \texttt{b}\}$ and let $L = \{ \langle M \rangle \mid M \text{ is a TM and } \mathscr{L}(M) \subseteq \texttt{a*} \}$. Prove that $L \notin$ **RE**.

Problem Fourteen: P and NP

Recall from lecture that the language $3COLOR = \{ \langle G \rangle \mid G \text{ is a 3-colorable graph} \}$ is **NP**-complete. The language $2COLOR = \{ \langle G \rangle \mid G \text{ is a 2-colorable graph} \}$ is known to be in **P** (you don't need to prove this). Below is a purported proof that **P** = **NP**:

Theorem: **P** = **NP**.

Proof: As we will prove in the lemma below, we have $2COLOR \le_P 3COLOR$. Since *3COLOR* is **NP**-hard, this means *2COLOR* is **NP**-hard. Because $2COLOR \in$ **P** and **P** ⊆ **NP**, we know that $2COLOR \in$ **NP**. Thus $2COLOR \in$ **NP** and *2COLOR* is **NP**-hard, so **NP**-complete. Since *2COLOR* is **NP**-complete and $2COLOR \in$ **P**, we thus have that **P** = **NP**. ■

Lemma: $2COLOR \le_P 3COLOR$.

Proof: We'll give a polynomial-time mapping reduction from *2COLOR* to *3COLOR*, which proves that $2COLOR \le_P 3COLOR$.

Given a graph $G = (V, E)$, let $f(\langle G \rangle)$ be the graph G' defined in terms of G by adding a new node v to G and adding an edge from v to each other node in G. We state without proof that f can be computed in polynomial time. Therefore, we will prove that $\langle G \rangle \in 2COLOR$ iff $f(\langle G \rangle) = \langle G' \rangle \in 3COLOR$, from which we can conclude that f is a polynomial-time mapping reduction from *2COLOR* to *3COLOR*, so $2COLOR \le_P 3COLOR$.

First, we prove that if G is 2-colorable, then G' is 3-colorable. To see this, consider any 2-coloring of G. For each node in G' that also appears in G, color that node the same color as its corresponding node in G. Then, color v the unused third color. This gives a 3-coloring of G'.

Next, we prove that if G' is 3-colorable, then G is 2-colorable. To see this, consider any 3-coloring of G'. We claim that all of the nodes in G' that also belong to G are colored with only two colors. To see this, note that since all these nodes are connected to the new node v, they must all have a color distinct from v's color, and thus must be colored using only two colors. Therefore, if we color the nodes in G the same color as the corresponding nodes in G', we end with a 2-coloring of G. ■

The above proof, unfortunately, is incorrect. What is wrong with this proof? Be specific. (The function f described in the lemma can indeed be computed in polynomial time, so that isn't the error in the proof.)