Extra Practice Problems 3

Here's another (giant!) compilation of practice problems you can use to review just about everything from this quarter. Some of these problems are marked with a star and touch on topics we haven't covered yet as of the time this set of problems is released (verifiers, non-**RE** languages, **P** and **NP**), so don't panic if you haven't seen those concepts yet. We figured it was better to get this packet of problems out early so that you could review anything you were shaky on than to hold off until the very end of the quarter.

Problem One: Cartesian Products and Subsets

Prove or disprove: if A, B, C, and D are sets where $A \times B \subseteq C \times D$, then $A \subseteq C$ and $B \subseteq D$.

Problem Two: Repeated Squaring

In many applications in computer science, especially cryptography, it is important to compute exponents efficiently. For example, the RSA public-key encryption system, widely used in secure communication, relies on computing huge powers of large numbers. Fortunately, there is a fast algorithm called *repeated squaring* for computing x^y in the special case where y is a natural number.

The repeated squaring algorithm is based on the following function *RS*:

$$RS(x, y) = \begin{cases} 1 & \text{if } y = 0 \\ RS(x, y/2)^2 & \text{if } y \text{ is even and } y > 0 \\ x \cdot RS(x, (y-1)/2)^2 & \text{if } y \text{ is odd and } y > 0 \end{cases}$$

For example, we could compute 2^{10} using $RS(2, 10)$ as follows:

In order to compute $RS(2, 10)$, we need to compute $RS(2, 5)^2$.
In order to compute $RS(2, 5)$, we need to compute $2 \cdot RS(2, 2)^2$.
In order to compute $RS(2, 2)$, we need to compute $RS(2, 1)^2$.
In order to compute $RS(2, 1)$, we need to compute $2 \cdot RS(2, 0)^2$.
By definition, $RS(2, 0) = 1$
so $RS(2, 1) = 2 \cdot RS(2, 0)^2 = 2 \cdot 1^2 = 2$.
so $RS(2, 2) = RS(2, 1)^2 = 2^2 = 4$.
so $RS(2, 5) = 2 \cdot RS(2, 2)^2 = 2 \cdot 4^2 = 32$.
so $RS(2, 10) = RS(2, 5)^2 = 32^2 = 1024$.

The *RS* function is interesting because it can be computed much faster than simply multiplying x by itself y times. Since *RS* is defined recursively in terms of *RS* with the y term roughly cut in half, *RS* can be evaluated using approximately $\log_2 y$ multiplications. (You don't need to prove this).

Prove that for any $x \in \mathbb{R}$ and any $y \in \mathbb{N}$, that $RS(x, y) = x^y$. *(Hint: use complete induction on y.)*

Problem Three: Domatic Partitions

On Practice Second Midterm Exam 4, there's a question about dominating sets. (If you haven't yet done that problem, stop and do that problem first – it's a really good one!)

Let's introduce some new terminology. A ***domatic partition of G*** is a way of splitting the nodes in G into disjoint, nonempty sets V_1, V_2, …, V_n such that each set V_i is a dominating set. (Two sets S and T are disjoint if $S \cap T = \varnothing$.) The ***domatic number*** of G, denoted $d(G)$, is the maximum number of sets in any domatic partition of G.

i. Let G be an undirected graph and let δ be the minimum degree of any node in G. Prove that $d(G) \leq \delta + 1$.

An ***isolated node*** in a graph G is a node that is not adjacent to any other nodes in G.

ii. Let G be an undirected graph with no isolated nodes. Prove that $d(G) \geq 2$. *(Hint: Use a result from Practice Second Midterm Exam 4. We normally won't ask anything directly off of practice exams, but since you're doing extra practice problems, we figured it's not unreasonable here.)*

iii. Prove that the bounds you came up with in parts (i) and (ii) are "tight" in the sense that, in general, you cannot improve upon these upper bounds or lower bounds without more knowledge of the structure of the graph. Specifically, give a graph G where $d(G) = \delta + 1$ and give a graph G with no isolated nodes where $d(G) = 2$. Briefly justify your answers.

Problem Four: Diagonalization and Logic ★

Given the predicates

- *String*(w), which states that w is a string over alphabet Σ;
- *TM*(M), which states that M is a TM with input alphabet Σ; and
- *Accepts*(M, w), which states that M accepts w,

along with the function $\langle O \rangle$, which represents the encoding of some object O, write a statement in first-order logic that says "$L_D \notin$ **RE**." (We'll cover L_D on Wednesday, May 31. Looking forward: the language L_D is defined as $L_D = \{ \langle M \rangle \mid \langle M \rangle \notin \mathscr{L}(M) \}$)

Problem Five: Lifts and Hasse Diagrams

Let $A = \{1, 2, 3\}$. Draw the Hasse diagram of the lift of $<$ over A to $\wp(A)$, which is the relation $\tilde{<}$ defined as

$$X \tilde{<} Y \quad \text{if} \quad Y \neq \varnothing \text{ and for any } x \in X \text{ and } y \in Y, \text{ we have } x < y.$$

Problem Six: Addition, Formally

Let A, B, C, and D be sets where $|A| = |C|$, $|B| = |D|$, $A \cap B = \varnothing$, and $C \cap D = \varnothing$. Using the formal definition of equal cardinality, prove that $|A \cup B| = |C \cup D|$.

Fun fact: this is how we can define what it means to add two cardinalities. For example, $\aleph_0 + \aleph_0$ is formally defined as "the cardinality of $|A \cup B|$, where A and B are any two disjoint sets of cardinality $\aleph_0$."

Problem Seven: Partial Sums

Suppose that you have a set S of $n > 0$ natural numbers. Prove that there must be a nonempty subset of S where the sum of the numbers in that subset is a multiple of n. *(Hint: Number the elements of S as $x_1, x_2, \ldots, x_n$. Then, look at x_1, $x_1 + x_2$, $x_1 + x_2 + x_3$, etc.)*

Problem Eight: Fun with DFAs and NFAs

Here's some true-or-false questions to ponder:

i. True or false: If D is a DFA over alphabet Σ and D has no accepting states, then $\mathscr{L}(D) = \varnothing$.

ii. True or false: If D is a DFA over alphabet Σ and D has no rejecting states, then $\mathscr{L}(D) = \Sigma^*$.

iii. True or false: If N is an NFA over alphabet Σ and N has no accepting states, then $\mathscr{L}(N) = \varnothing$.

iv. True or false: If N is an NFA over alphabet Σ and N has no rejecting states, then $\mathscr{L}(N) = \Sigma^*$.

Let $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}\}$ and let L be the following language:

$$L = \{ w \in \Sigma^* \mid \text{every character from } \Sigma \text{ appears at least once in } w \}$$

Any DFA for L must have at least 32 states (you don't need to prove this.)

v. Prove that any DFA for $\overline{L}$ must have at least 32 states.

vi. Design a reasonably-sized NFA for $\overline{L}$. This shows that even if you can't find a small NFA for a language, you might be able to find a small NFA for its complement.

Problem Nine: Antitautonyms

Let $\Sigma = \{\texttt{a}, \texttt{b}\}$ and consider the language $L = \{ wx \mid w \in \Sigma^*, x \in \Sigma^*, |w| = |x|, \text{ and } w \neq x \}$. Prove that L is not a regular language.

Problem Ten: Closure Properties of CFGs

This question explores closure properties of CFLs.

i. Show that the context-free languages are closed under union, concatenation, and Kleene star.

ii. Although we didn't prove this, the context-free languages are not closed under complementation. In lecture, you saw a CFG for the language $\{ w \in \{\texttt{a}, \texttt{b}\}^* \mid w \text{ is a palindrome} \}$, and on Problem Set Seven you built a CFG for the complement of this language. Explain how this is possible even though the context-free languages aren't closed under complementation.

Problem Eleven: Designing Turing Machines

Design a TM over the alphabet $\Sigma = \{\texttt{a}, \texttt{b}\}$ whose language is $\{ w \in \Sigma^* \mid w$ does not contain `aa` or `bb` as substrings $\}$.

Problem Twelve: Approximating L_D ★

Prove that there is a language X where $X \subseteq L_D$, where X contains infinitely many strings, and where X is an **RE** language.

Problem Thirteen: Narcissistic Turing Machines ★

Let $L = \{ \langle M \rangle \mid M \text{ is a TM and } \mathscr{L}(M) = \{ \langle M \rangle \} \}$. In other words, L is the set of all TMs that accept themselves and only themselves. (We can think of them as narcissistic TMs.)

Prove that $L \notin \mathbf{R}$.

Problem Fourteen: Intersecting Sets

(Midterm Exam, Fall 2015)

We can use set-builder notation to describe a set by giving a rule that describes what elements are in the set. Specifically, if $P(x)$ is some predicate, then the set

$$\{ x \mid P(x) \}$$

is the set containing all objects x where $P(x)$ is true (and no elements besides these).

Let's suppose that we have a set S that is a set of sets (that is, every element of S is itself a set). Formally, this means we're talking about a set S where

$$\forall T.\ (T \in S \to \mathit{Set}(T)).$$

If S is a set of sets, then we can take the intersection of all of the sets contained in S. The resulting set, denoted $\cap S$, is called the ***intersection of S***. For example, if

$$S = \{ \{1, 2, 3, 4\}, \{2, 3, 4, 5\}, \{3, 4, 5, 6\} \},$$

then $\cap S = \{3, 4\}$.

Intuitively, an object x is an element of $\cap S$ if x belongs to every element of S. We can use this intuition to come up with a formal definition of $\cap S$, which is given below:

$$\cap S = \{ x \mid \forall T.\ (T \in S \to x \in T) \}$$

This is the standard definition of the intersection of $\cap S$ that's used throughout set theory. However, this definition of $\cap S$ has a pretty major edge case.

Prove that the set $\cap \varnothing$ does not exist. *(Hint: Think back to Problem Set Four.)*

Problem Fifteen: Powers, Multiples, and 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 Sixteen: Strongly Connected Graphs

A directed graph is called ***strongly connected*** if for any pair of nodes u and v in the graph, there's a path from u to v and from v to u. In a directed graph, the ***indegree*** of a node is the number of edges entering it, and its ***outdegree*** is the number of edges leaving it. Find a strongly-connected graph with 137 nodes where each node's ***indegree*** is equal to its ***outdegree***.

Problem Seventeen: Closure Properties and 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 Eighteen: Properties of 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 Nineteen: The Indistinguishability Relation

Let L be an arbitrary language over an alphabet Σ. We'll say that two strings $x, y \in \Sigma^*$ are ***indistinguishable*** relative to L, denoted $x \equiv_L y$, if the following is true:

$$\forall w \in \Sigma^*. (xw \in L \leftrightarrow yw \in L).$$

i. Prove that if L is any language over Σ, then $\equiv_L$ is an equivalence relation over Σ^*.

ii. Prove that if $x \equiv_L y$ and $x \in L$, then $y \in L$.

iii. Let $L = \{ w \in \{\mathtt{a}, \mathtt{b}\}^* \mid |w| \equiv_3 2 \}$. What are all the equivalence classes of $\equiv_L$?

Problem Twenty: Spin Me An Entree

Suppose that n people are seated at a round table at a restaurant. Each of the n people orders a different entrée for dinner. The waiter brings all of the entrées 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 Twenty One: Regular Languages and Parity

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

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

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 Twenty Two: Giant Balanced Strings

Let $\Sigma = \{$`a`, `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 Twenty Three: CFGs and Swedish Pop Music

Let $\Sigma = \{$`a`, `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 Twenty Four: Checking for Equality

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

Problem Twenty Five: Closure Properties of RE ★

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 Twenty Six: Output Restrictions ★

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 $L \notin$ **R**.

Problem Twenty Seven: Power Sets and Cartesian Products

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

Problem Twenty Eight: The Well-Ordering Principle

The ***well-ordering principle*** states that if $S \subseteq \mathbb{N}$ and $S \neq \varnothing$, 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

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

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 Twenty Nine: Tensor Products and Graph Coloring

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 Thirty: Restricting and Manipulating Logic

Consider the following formula in first-order logic:

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

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 Thirty One: Restrictions of Relations

Let R be a binary relation over a set A. For any set $B \subseteq A$, we can define the ***restriction of R to B***, denoted $R|_B$, to be a binary relation over the set B defined as follows:

$$x\ R|_B\ y \quad \text{if} \quad xRy.$$

In other words, the relation $R|_B$ behaves the same as R, but only on the elements of B.

i. Prove or disprove: if R is an equivalence relation over a set A and B is an arbitrary subset of A, then $R|_B$ is an equivalence relation over B.

ii. Prove or disprove: if R is a strict order over a set A and B is an arbitrary subset of A, then $R|_B$ is a strict order over B.

iii. Prove or disprove: there is a strict order R over a set A and a set $B \subseteq A$ such that $R|_B$ is an equivalence relation.

iv. Prove or disprove: there is an equivalence relation R over a set A and a set $B \subseteq A$ such that $R|_B$ is a strict order.

Problem Thirty Two: Functions and Relations, Together!

(Midterm Exam, Spring 2015)

In this question, let $A = \{1, 2, 3, 4, 5\}$. Let $f : A \to A$ be an arbitrary function from A to A that we know is ***not a surjection***. We can then define a new binary relation $\sim_f$ as follows: for any $a, b \in A$, we say $a \sim_f b$ if $f(a) = b$. Notice that this relation depends on the particular non-surjective function f that we pick; if we choose f differently, we'll get back different relations. This question explores what we can say with certainty about $\sim_f$ knowing only that its domain and codomain are A and that it is not a surjection.

Below are the six types of relations we explored over the course of this quarter. For each of the types, determine which of the following is true:

- The relation $\sim_f$ is ***always*** a relation of the given type, regardless of which non-surjective function $f : A \to A$ we pick.
- The relation $\sim_f$ is ***never*** a relation of the given type, regardless of which non-surjective function $f : A \to A$ we pick.
- The relation $\sim_f$ is ***sometimes, but not always*** a relation of the given type, depending on which particular non- surjective function $f : A \to A$ we pick.

Since these options are mutually exclusive, check only one box per row. *(Hint: Draw a lot of pictures.)*

$\sim_f$ is reflexive	☐ ***Always***	☐ ***Sometimes, but not always***	☐ ***Never***
$\sim_f$ is irreflexive	☐ ***Always***	☐ ***Sometimes, but not always***	☐ ***Never***
$\sim_f$ is symmetric	☐ ***Always***	☐ ***Sometimes, but not always***	☐ ***Never***
$\sim_f$ is asymmetric	☐ ***Always***	☐ ***Sometimes, but not always***	☐ ***Never***
$\sim_f$ is transitive	☐ ***Always***	☐ ***Sometimes, but not always***	☐ ***Never***
$\sim_f$ is an equivalence relation	☐ ***Always***	☐ ***Sometimes, but not always***	☐ ***Never***
$\sim_f$ is a strict order	☐ ***Always***	☐ ***Sometimes, but not always***	☐ ***Never***

Problem Thirty Three: Permutation Parity*

Let n be an odd natural number and consider the set $S = \{1, 2, 3, \ldots, n\}$. A ***permutation*** of S is a bijection $\sigma : S \to S$. 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.

Problem Thirty Four: Reversing Regular Languages

If w is a string, then w^R represents the reversal of that string. For example, the reversal of "table" is "elbat." If L is a language, then L^R is the language $\{ w^R \mid w \in L \}$ consisting of all the reversals of the strings in L.

It turns out that the regular languages are closed under reversal.

i. Give a construction that turns an NFA for a language L into an NFA for the language L^R. No proof is necessary.

ii. Give a construction that turns a regular expression for a language L into a regular expression for the language L^R. No proof is necessary.

Problem Thirty Five: Centrist Languages

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

Problem Thirty Six: Parenthesis Parity

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

Problem Thirty Seven: A Better Sorting Algorithm

In lecture, we designed a Turing machine that, given a string of 0s and 1s, puts them into ascending order and then halts. The sorting algorithm we used worked by finding a copy of the substring 10, reversing it, and repeating until no more copies of this substring exists. While this algorithm works, it's not very efficient.

Design a TM that sorts a string of 0s and 1s and does so more efficiently than the machine from class. By "more efficiently," we mean that the TM you design should, on average, take many fewer steps to complete than our TM.

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

Problem Thirty Eight: Computable Functions ★

A function $f : \Sigma^* \to \Sigma^*$ is called a ***computable function*** if it is possible to write a function `compute_f` that takes in a string w and outputs $f(w)$.

Given any computable function f and language L, let's 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 Thirty Nine: Complementary TMs ★

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

Problem Forty: Nonregular Languages via a Different Path

The Myhill-Nerode theorem is a powerful tool for proving that languages aren't regular, but it might not be the easiest way to prove that a given language isn't regular. This problem explores a different route you can take to prove that various languages aren't regular.

i. Prove that if L_1 is a language, L_2 is a regular language, and $L_1 \cap L_2$ is not regular, then L_1 is not regular.

ii. Using your result from part (i), but without using the Myhill-Nerode theorem, prove that the language $L = \{ w \in \{\texttt{a}, \texttt{b}\}^* \mid w \text{ has the same number of } \texttt{a}\text{'s as } \texttt{b}\text{'s} \}$ is not regular.

Problem Forty One: Just a Few More Grammars

Below is a list of alphabets and languages over those alphabets. Design a CFG for each language.

i. Let $\Sigma = \{\texttt{1}, \texttt{≥}\}$ and let $L = \{ \texttt{1}^m\texttt{≥}\texttt{1}^n \mid m, n \in \mathbb{N} \text{ and } m \geq n \}$. Write a CFG for L.

ii. On Problem Set 8, you explored the language $L_1 = \{ \texttt{1}^m\texttt{+}\texttt{1}^n\texttt{=}\texttt{1}^{m+n} \mid m, n \in \mathbb{N} \}$ over the alphabet {1, +, =} Consider the following generalization of this language, which we will call L_2, which consists of all strings describing unary encodings of two sums that equal one another. For example:

1 + 3 = 4 would be encoded as `1+111=1111`

4 = 1 + 3 would be encoded as `1111=1+111`

2 + 2 = 1 + 3 would be encoded as `11+11=1+111`

2+0+2+0=0+4+0 would be encoded as `11++11+=+1111+`

0=0 would be encoded as `=`

Notice that there can be any number of summands on each side of the =, but there should be exactly one = in the string; thus `1=1=1` $\notin L_2$. Write a CFG for L_2.

iii. Let $\Sigma = \{\texttt{(}, \texttt{)}, \texttt{[}, \texttt{]}\}$ and let $L = \{w \in \Sigma^* \mid w$ is a string of balanced parentheses and brackets$\}$. This means that all parentheses and brackets must match one another, and collectively they must obey the appropriate nesting rules. For example, `([])[]` $\in L$, but `([)]` $\notin L$. Write a CFG for L.

Problem Forty Two: Formalizing the Lava Diagram ★

In the Guide to the Lava Diagram, we explored these two languages:

$$L_1 = \{ \langle M \rangle \mid M \text{ is a TM and } |\mathscr{L}(M)| \geq 2 \}$$

$$L_2 = \{ \langle M \rangle \mid M \text{ is a TM and } |\mathscr{L}(M)| = 2 \}$$

The Guide makes many claims about these languages, but never actually proves them.

i. Prove that L_1 is undecidable.

ii. Prove that L_2 is undecidable.

iii. Show that L_1 is recognizable by designing a verifier for it. Your verifier should be represented in pseudocode via a method with a signature like this one:

```
bool imConvincedIsInL1(TM M, Arg1Type arg1, Arg2Type arg2, …, ArgnType argn)
```

where the arguments beyond the first represent the certificate and can be of any type you'd like.

Problem Forty Three: Self-Reference and RE ★

Since A_{TM} is an **RE** language, there's a *recognizer* for A_{TM}, which we can represent in software as a method willAccept. Consider the following program *P*:

```
int main() {
    string me = mySource();
    string input = getInput();

    if (willAccept(me, input)) {
        reject();
    } else {
        accept();
    }
}
```

Prove that this program loops on all inputs.

Problem Forty Four: Threat Detection ★

There have been a ton of news articles about computer systems being attacked by independent actors and nation-states. You might wonder why our computers are so vulnerable – couldn't someone just write a program that analyzes a program's source code and determine whether it has any security problems?

Let's consider a simplified scenario. Imagine that there's a special method

```
void sendSecretDataTo(String emailAddress)
```

that, if called, sends an email containing a bunch of secret information to the specified email address. For example, you might call `sendSecretDataTo("john.roberts@supremecourt.gov")` to send all the secret data to Chief Justice John Roberts, or call `sendSecretDataTo("bad.actor@hackers.com")` to send all the secret data to evil hackers.

You are interested in whether it's possible to write a method

```
bool canLeakDataTo(String program, String emailAddress)
```

that takes as input the source code of a program and an email address, then returns true if there is some execution of `program` that causes the secret data to the specified email address and returns false otherwise. This program would let you check whether a particular program might ever leak data to a specified email address, which would make it easier to check whether the program is secure.

Is it possible to implement this method? If so, write code for the method, then prove that your code works as intended. If not, prove that it's not possible to implement this method.

(As a note, if you try implementing this method, you should do so in a way that doesn't call `sendSecretDataTo`, and if you try proving this method can't be written, you can assume that no correct implementation will ever call `sendSecretDataTo`.)

Problem Forty Five: Translating Out Of Logic

For each first-order statement below, write a short English sentence that describes what that sentence says. While you technically *can* literally translate these statements back into English, you'll probably have better luck translating them if you try to think about what they really mean. Then, determine whether the statement is true or false based on what you know about sets and set theory.

- $\exists S.\ (Set(S) \wedge \forall x.\ x \notin S)$
- $\forall x.\ \exists S.\ (Set(S) \wedge x \notin S)$
- $\forall S.\ (Set(S) \rightarrow \exists x.\ x \notin S)$
- $\forall S.\ (Set(S) \wedge \exists x.\ x \notin S)$
- $\exists S.\ (Set(S) \wedge \exists x.\ x \notin S)$
- $\exists S.\ (Set(S) \rightarrow \forall x.\ x \in S)$
- $\exists S.\ (Set(S) \ \wedge \ \forall x.\ x \notin S \ \wedge \ \forall T.\ (Set(T) \wedge S \neq T \rightarrow \exists x.\ x \in T))$
- $\exists S.\ (Set(S) \ \wedge \ \forall x.\ x \notin S \ \wedge \ \exists T.\ (Set(T) \wedge \forall x.\ x \notin T \wedge S \neq T))$
- $\exists S.\ (Set(S) \ \wedge \ \forall x.\ x \notin S) \ \ \wedge \ \ \exists T.\ (Set(T) \wedge \forall x.\ x \notin T))$