Chapter 0: Prerequisites

It is expected that the student taking this Computability course is already well-versed in the basics of formal languages and automata theory. The exercises in this chapter should serve as a litmus test: if you cannot solve a significant portion of the regular (non-starred) exercises, you may be missing some of the prerequisite knowledge.

At the end of this chapter we include several more difficult exercises marked with one or two stars. These should be viewed as a challenge for extra credit.

Exercise 0.1
Do the sets $\mathbb{N}$ and $\mathbb{N} \times \mathbb{N}$ have the same cardinality? Why?

Exercise 0.2
What is the cardinality of the set $\{a, b\}^*$?
(This is the set of all finite strings over the alphabet $\{a, b\}$.)

Exercise 0.3
Suppose that χ and φ are two partial functions on $\mathbb{N}$. Additionally, we know that their composition $\chi \circ \varphi$ is total, i.e., defined on the entire set $\mathbb{N}$. Is χ necessarily total? Is φ necessarily total? Prove your answers.

(Let's use $f(x) = \perp$ to denote that f is not defined on x. For consistency, let's assume that for any f we have $f(\perp) = \perp$. Then we can formally define $f \circ g$ as follows: $\forall x : (f \circ g)(x) = f(g(x))$.)

Exercise 0.4
Show an algorithm that will convert any given nondeterministic pushdown automaton into a nondeterministic Turing machine (NTM) that recognizes the same language.

Exercise 0.5
Discuss whether and how the construction in the previous exercase changes if we change some occurrences of "nondeterministic" to "deterministic".

Exercise 0.6
The language recognized by a nondeterministic finite automaton (NFA) is defined as the language of all words for which an accepting computation *exists*.

What changes if we consider the set of words for which *all* computations that finish reading the input *must* accept?

Formally, for any NFA $A = (K, \Sigma, \delta, q_0, F)$ we can define the language

$$L'(A) = \{w \mid \text{if } (q_0, w) \vdash_A^* (q, \varepsilon) \text{ then } q \in F\}$$

Exercise 0.7
Consider the following problem: Given an NFA A, decide whether $L(A) = L(A)^R$. (I.e., whether the language remains the same if we reverse all of its words.) Is this problem decidable?

Exercise 0.8
Explain, on a high level, how a universal Turing machine operates.

Exercise 0.9
Consider a *nondeterministic* Turing machine with a single tape that is only infinite towards the right. There is a read-only endmarker to the left of the input word.

Instead of the classical movement (i.e., at most one cell in any direction) this Turing machine can move in only two ways: either it makes a step to the right, or it its head jumps to the endmarker at the beginning of the tape.

Give a high-level description of how to simulate a standard Turing machine on this modified Turing machine.

Exercise 0.10 $\star$
What would change in the previous exercise if the new "jumping" Turing machine were *deterministic*?

Exercise 0.11 $\star$
Let $\Sigma = \{0, 1\}$ and let $<$ be the length-lexicographic ordering of Σ^*. Prove or disprove: A language L is recursive if and only if there is a Turing machine that enumerates all words of L ordered by $<$.

(A formal definition of $<$: we have $u < v$ if $|u| < |v|$ and also if $|u| = |v|$ and there is a w such that u has a prefix $w0$ and v has a prefix $w1$.)

Exercise 0.12 $\star$
Consider the following problem: Given an NTM A and a word w, decide whether *all* computations of A on w terminate. Is this problem decidable, or at least partially decidable?

Exercise 0.13 $\star\star$
Prove or disprove: for any total order $\prec$ on Σ^*, a language L is recursive if and only if there is a Turing machine that enumerates all words of L ordered by $\prec$.

Chapter 1: Automaton-style models of computation

Exercise 1.1
Given is a deterministic Turing machine. Describe how to construct an equivalent Minsky register machine. Pay attention to the following details:

- The input for a Turing machine is a word, not a number. How would you provide this input to a register machine?
- How will you define acceptance of an input?
- There are three separate parts of a configuration of a Turing machine: the tape, the state, and the position of the head. How is each of them represented during the simulation?

Exercise 1.2
Consider the simplest deterministic Turing machine that accepts the language of words over $\{a, b\}$ in which the number of as is divisible by 3.

Imagine that we do the following sequence of transformations: First, we convert the DTM into an equivalent pushdown automaton with two stacks. Next, we shall convert that into an equivalent automaton with three counters (the first stack encoded into a number, the second counter encoded into a number, and one counter for auxiliary storage during the computation). Finally, we convert that automaton into an equivalent one that only uses two counters.

Suppose that we run the last automaton on the input *aababab*. Estimate the largest number that will appear in one of its counters.

What does the answer imply about the number of steps the computation took?

Exercise 1.3
Give an alternate definition of Minsky register machines where the program is a flowchart.

Exercise 1.4
Show how to compute the function $f(n) = \lfloor \sqrt{n} \rfloor$ on a register machine.

Exercise 1.5
Show how to compute the function $g(n) = 2^{2^n}$ on a register machine.

Exercise 1.6
Show how to compute the function $h(n) = \lceil \log_2(n+1) \rceil$ on a register machine.

Exercise 1.7
Give a high-level description of a register machine that takes n as input and computes the value of the n-th prime.

Exercise 1.8
Show how to construct a bidirectional finite automaton with a single counter for the language of all palindromes over $\{a, b\}$.

Chapter 2: Grammar-style models of computation

Exercise 2.1
Consider the conversion from a Turing machine to a Markov algorithm. The lecture notes are vague when it comes to steps when the input symbol is a blank. Fill in the missing details.

(Hint: Use the fact that a rule is only applied if none of the preceding rules can be applied. You can use this to make sure that the head does not see any letter.)

Exercise 2.2
Construct a Markov algorithm with input alphabet $\{a, b\}$ that terminates iff the input is a palindrome.

Exercise 2.3
Give a general construction that will convert any Markov algorithm into an equivalent type-0 (i.e., unrestricted) grammar.

Exercise 2.4
Prove or disprove: for every recursively enumerable language there is an equivalent type-0 grammar with the extra property that each word generated by the grammar has exactly one valid derivation.

Exercise 2.5
Consider the following Post tag system: $(1, \{a, b, c, d, e, f, g\}, \{a \to \varepsilon, b \to feebac, c \to \varepsilon, d \to afgafgafgaf, e \to babadge, f \to caga, g \to \varepsilon\})$.

Find the language accepted by this tag system. (I.e., find the set of all starting words for which this tag system will terminate after finitely many steps of computation.)

Chapter 3: Exotic models of computation

Exercise 3.1
Let $WT(f)$ be the family of languages that can be recognized by a Wang tile set whose space complexity is $O(f)$. Find a language that belongs to $WT(\log_2 n) - WT(1)$.

Exercise 3.2
Prove or disprove: all context-free languages are in $WT(n)$.

Exercise 3.3
Prove that the language of all palindromes over $\{a, b\}$ is not in any $WT(f)$ where $f \in o(n)$.

Exercise 3.4
Construct a set of Wang tiles for the language $\{ww \mid w \in \{a, b\}^*\}$.

Exercise 3.5
One of the original goals of Wang's research was to find an algorithm that would decide whether a given set of tiles can be used to tile an infinite plane. In 1961 he managed to come up with such an algorithm but it was based on an unproved assertion that any such tiling must necessarily be periodic.

However, in 1965 Robert Berger found a finite set of tiles (consisting of exactly 20 426 tile types) that did tile the infinite plane but all valid tilings were aperiodic. Currently, the smallest known set of tiles with this property has only 13 tiles, and it is conjectured that one of its 12-tile subsets also works.

Try solving a slightly easier problem: Show a finite sequence of tiles that tiles the plane but any tiling that contains at least one occurrence a tile of the first type must be aperiodic.

Exercise 3.6 ⋆
Find a finite set of tiles that tiles the plane only aperiodically.

Exercise 3.7
Write a program using the `SUBLEQ` instruction that will multiply the values in `memory[1]` and `memory[2]`, and store the result into `memory[3]`. (You may assume that the first two numbers are positive and the third one is initially zero.)

Exercise 3.8
Consider the instruction `SUBEQ` that behaves like `SUBLEQ` but the conditional jump is only executed if the result was exactly zero.

Describe a program in your favorite programming language that can translate any program that uses `SUBLEQ` into an equivalent program that uses `SUBEQ`.

Exercise 3.9 ⋆
Write a program using the `SUBLEQ` instruction that will test primality of `memory[1]`.

Chapter 4: Primitive recursion

Starting from this chapter, we will occasionally use "$\overline{x}$" as shorthand for "$x_1, \ldots, x_k$".

Exercise 4.1
Directly by using the definition prove that the following functions are primitive recursive:

- $p(x) = \begin{cases} 0 & \leftarrow x = 0 \\ x - 1 & \leftarrow \text{else} \end{cases}$
- $sub(x, y) = \begin{cases} x - y & \leftarrow x \geq y \\ 0 & \leftarrow \text{else} \end{cases}$
- $hop(x) = 47x + 74$
- $sgn(x) = \begin{cases} 0 & \leftarrow x = 0 \\ 1 & \leftarrow \text{else} \end{cases}$
- $\overline{sgn}(x) = \begin{cases} 1 & \leftarrow x = 0 \\ 0 & \leftarrow \text{else} \end{cases}$
- $diff(x, y) = |x - y|$.
- $max(x, y)$
- $median(x, y, z)$
- $equals(x, y) = \begin{cases} 1 & \leftarrow x = y \\ 0 & \leftarrow \text{else} \end{cases}$
- $fact(x) = x!$

Exercise 4.2
Prove that each polynomial with positive integer coefficients is primitive recursive.

Exercise 4.3
Prove or disprove: Let p be a polynomial with integer coefficients, some of them possibly negative. Then the function f_p defined as $\forall n \in \mathbb{N} : f_p(n) = \max(0, p(n))$ is primitive recursive.

Exercise 4.4
Prove the primitive recursiveness of the logical operation xor (exclusive or).

Formally: Suppose that p and q are primitive recursive predicates with the same arity. Define r as a predicate with the same arity that returns 1 precisely for those inputs for which exactly one of p and q returns 1. Prove that r must also be primitive recursive.

Exercise 4.5
Prove that for any primitive recursive function f there is a primitive recursive function g with the following property: $\lim_{n \to \infty} f(n)/g(n) = 0$.

(Informally, g grows faster than f.)

Exercise 4.6
Prove that for any primitive recursive function f the function g defined as $g(x) = \sum_{i<x} f(i)$ must be primitive recursive as well. (Function g is usually called the prefix sum function of f.)

Exercise 4.7 ⋆
Prove a more general version of the previous exercise: For any primitive recursive functions lo, hi, and f we can define a new function g as follows:

$$g(\overline{x}) = \sum_{lo(\overline{x}) \le i \le hi(\overline{x})} f(i, \overline{x})$$

Prove that g must be primitive recursive.

Exercise 4.8
Using anything from the previous exercises, prove that the following functions and predicates are all primitive recursive:

- $mod(x,y) = \begin{cases} x \bmod y & \leftarrow y > 0 \\ 0 & \leftarrow \text{else} \end{cases}$
- $divides(x,y) = \begin{cases} 1 & \leftarrow y > 0 \ \wedge \ y \text{ divides } x \\ 0 & \leftarrow \text{else} \end{cases}$
- $div(x,y) = \begin{cases} \lfloor x/y \rfloor & \leftarrow y > 0 \\ 0 & \leftarrow \text{else} \end{cases}$

Exercise 4.9 ⋆
Prove the theorem on *bounded minimization*: Let $f(y, \overline{x})$ and $g(\overline{x})$ be primitive recursive functions, and let

$$h(\overline{x}) = \begin{cases} \min\{i \mid i < g(\overline{x}) \ \wedge \ f(i, \overline{x}) > 0\} & \leftarrow \text{if such an } i \text{ exists} \\ g(\overline{x}) & \leftarrow \text{else} \end{cases}$$

Then h must also be primitive recursive.

In words: $h(\overline{x})$ computes the values $f(y, \overline{x})$ for $y = 0$, $y = 1$, ..., until it either finds an y that yields a positive return value, or reaches the predetermined upper bound $g(\overline{x})$.

Exercise 4.10
The theorem on *bounded minimization* is a useful tool to show primitive recursiveness of various functions. Use it on the functions listed below. Note that for these functions a direct proof would usually be more painful.

- $ceil(x,y) = \begin{cases} \lceil x/y \rceil & \leftarrow y > 0 \\ 0 & \leftarrow \text{else} \end{cases}$

- $sqrt(x) = \lceil \sqrt{x} \rceil$
- ($\star$) $isprime(x) = \begin{cases} 1 & \leftarrow x \text{ is a prime number} \\ 0 & \leftarrow \text{else} \end{cases}$

Hint 1: The value $ceil(x, y)$ can be defined as the smallest z such that $yz \geq x$. What can we use as an upper bound for the theorem on bounded minimization?

Hint 2: The construction of a primality test predicate will probably be easier if you define some helper functions along the way.

Exercise 4.11
Prove that the n-th prime (numbering from 0) is less than or equal to 2^{2^n}.

Exercise 4.12
Using the *isprime* predicate and the result of the previous exercise, prove that the function that takes n as its input and returns the n-th prime (numbering from 0) as its output is primitive recursive.

Exercise 4.13
The "recipe" of a primitive recursive function f is a sequence of functions $f_1, f_2, \ldots, f_k \equiv f$ such that each f_i is either zero, successor, a projection, or it can be produced from lower-numbered elements of the sequence using composition or primitive recursion.

Prove or disprove: Let t be a unary constant function defined as $\forall x : t(x) = 1000$. Then every recipe for t has $k \geq 1000$ (i.e., 1000 or more functions in the list).

Exercise 4.14
Prove that $f(x) =$ (the digit of the order 10^{-x} in the decimal expansion of $\sqrt{2}$) is a primitive recursive function.

(Hint: One possible solution starts by proving that $f(x) = \lfloor x\sqrt{2} \rfloor$ is primitive recursive.)

Exercise 4.15
Prove that 2D primitive recursion preserves primitive recursiveness.

Formally, prove that for any primitive recursive function g_1, g_2 and h (with appropriate arities) the function f defined below is also primitive recursive.

$$\begin{aligned} f(\ \ 0,\ \ \ \ y,\ \overline{z}) &= g_1(y, \overline{z}) \\ f(\ x{+}1,\ \ \ 0,\ \overline{z}) &= g_2(x, \overline{z}) \\ f(\ x{+}1,\ y{+}1,\ \overline{z}) &= h(\ f(x, y+1, \overline{z}),\ f(x+1, y, \overline{z}),\ x,\ y,\ \overline{z}) \end{aligned}$$

Exercise 4.16
Using 2D primitive recursion, prove that the binomial coefficients are primitive recursive.

Exercise 4.17
Find a closed-form formula for the l function mentioned in lecture notes. (Here, l is the "left" inverse function to Cantor's pairing function c.)

Exercise 4.18
Consider the encoding of finite sequences into prime powers. A unary function f is called a *reversing function* if it has the following property: For any sequence $a_1, \ldots, a_n$, let $\vec{q}$ be the code of this sequence and let $\overleftarrow{q}$ be the code of the sequence $a_n, \ldots, a_1$. Then $f(\vec{q})$ must return $\overleftarrow{q}$.

Prove or disprove: there is a primitive recursive reversing function.

Exercise 4.19
There are plenty of other encodings of finite sequences into integers. We will now show a way that is inspired by functional programming languages. The trick is to represent a non-empty sequence as a pair (head,tail) where head is its first element and tail is a code of the remainder of the sequence.

Let s be the successor function and let c be Cantor's pairing function. By composing them we obtain the function sc. Note that this function is one-to-one and never returns a zero. We can now encode a sequence as follows: the code of an empty sequence is 0, and the code of the sequence $a_1, \ldots, a_n$ is $sc(a_1, q_2)$ where q_2 is the code of $a_2, \ldots, a_n$.

Compare this new encoding with the old one on the following sequence: the first 10 digits of π. Which one of them encodes this sequence into a smaller number?

Exercise 4.20
Modify our proof of the theorem on primitive recursive time complexity to use the head/tail encoding.

Chapter 5: General recursion

Exercise 5.1
Let A be the binary Ackermann function. Suppose we define functions B and C as follows:

$$B(x,y,z) = \begin{cases} 1 & \leftarrow A(x,y) < z \\ 0 & \leftarrow A(x,y) \geq z \end{cases}$$

$$C(x,y,z) = \begin{cases} 1 & \leftarrow A(x,y) > z \\ 0 & \leftarrow A(x,y) \leq z \end{cases}$$

Is B primitive recursive? Is C primitive recursive?

Exercise 5.2
We have seen that the evaluation of the Ackermann function eventually terminates because each function call either decreases the first argument, or it preserves the first argument and decreases the second one.

Find a binary recursively-defined function B that will not have this property. (I.e., for any input B must terminate, but it must sometimes be the case that the recursive call either increases the first argument, or preserves the first and increases the second one.)

Then, find a binary recursively-defined function C with the following stronger property: for any $p, q \in \mathbb{N}$ there are $x, y \in \mathbb{N}$ such that the computation of $C(x,y)$ needs to evaluate some $C(x',y')$ with $x' > x+p$ and $y' > y+q$.

Exercise 5.3
Let f be a total unary function. Let $V(f) = \{f(x) \mid x \in \mathbb{N}\}$ be the set of all values returned by the function f.

Prove or disprove: For any primitive recursive function f there is a primitive recursive function g such that $V(f) = V(g)$ and for each $y \in V(g)$ there are infinitely many x such that $g(x) = y$.

Exercise 5.4
Let $V(f)$ be the set of all values of f, as defined above.

Prove or disprove: If f is a recursive function such that $V(f)$ is infinite, there is a recursive function g such that $V(f) = V(g)$ and g is one-to-one.

Exercise 5.5
Let $V(f)$ be the set of all values of f, as defined above.

Prove or disprove: If f is a primitive recursive function such that $V(f)$ is infinite, there is a primitive recursive function g such that $V(f) = V(g)$ and g is one-to-one.

Exercise 5.6 $\star$
Prove or disprove: Let f be a unary primitive recursive function that is one-to-one. Then there is a *primitive recursive* function g such that $\forall n : g(f(n)) = n$.

Exercise 5.7
Let $\varphi_0, \varphi_1, \ldots$ be any numbering of all unary primitive recursive function, and let U be the corresponding universal function. (I.e., we have $\forall n, x : U(n, x) = \varphi_n(x)$.)

Two binary total functions f and g are called similar if $\forall x, y : |f(x, y) - g(x, y)| \leq 47$.

Prove or disprove: There is no primitive recursive function that is similar to U.

Exercise 5.8
Define a universal function for all partial recursive functions.

Is the function you just defined a partial recursive function? If no, can we use the same proof as for the universal primitive recursive function? If yes, where and why does the same proof technique fail in this case?

Exercise 5.9
Is it possible to define an efficient numbering (and hence a recursive universal function) for all unary recursive functions?

Exercise 5.10
Consider the following definition: *Faux minimization* takes a function $f(y, \overline{x})$ and produces a new function $MIN[f] \equiv g$ as defined below.

$$g(\overline{x}) = \begin{cases} \min\{i \mid f(i, \overline{x}) > 0\} & \leftarrow \text{if there is such an } i \\ \bot & \leftarrow \text{otherwise} \end{cases}$$

(The difference between minimization and faux minimization is that here we omitted the constraint $\forall j < i : f(j, \overline{x}) = 0$. Hence, it may now be the case that some of those $f(j, \overline{x})$ are undefined.)

Prove or disprove: the class of partial recursive functions is closed under faux minimization.

Chapter 6: Computable reals

Exercise 6.1
Give a formal proof that Turing's original definition defines the same set of computable reals as the modern definition.

Exercise 6.2
A nonnegative real x is called $\mathbb{Q}$-computable if there are recursive function $a, b : \mathbb{N} \to \mathbb{N}$ such that:
- $\forall n : b(n) > 0$
- $\forall n : \left| x - \frac{a(n)}{b(n)} \right| < \frac{1}{10^n}$

Compare the sets of computable and $\mathbb{Q}$-computable reals.

Exercise 6.3
A nonnegative real x is called convergently computable if there are recursive function $a, b : \mathbb{N} \to \mathbb{N}$ such that:
- $\forall n : b(n) > 0$
- $\lim_{n \to \infty} \frac{a(n)}{b(n)} = x$.

Compare the sets of computable and convergently computable reals.

Exercise 6.4
Prove that all nonnegative algebraic numbers are computable.

Exercise 6.5
Prove or disprove: is the set of all computable reals a field?

(The field operations are the standard addition and multiplication. A negative real is computable iff its absolute value is.)

Chapter 7: Busy Beaver and other nonrecursive functions

Exercise 7.1
Prove or disprove: the Busy Beaver function grows faster than any recursive function.

Exercise 7.2
Let B_M be the the Busy Beaver function for Minsky register machines: $B_M(n)$ is the largest number produced in a register by a program with n instructions that starts with all registers set to zero and eventually terminates.

Prove that B_M is not recursive.

Exercise 7.3
Consider the function B_M as defined above. Find as many of its exact values as you can.

Exercise 7.4
Consider the Turing machines as defined for the Busy Beaver function – deterministic, with a single tape infinite in both directions, all cells are initially set to 0, and cells may only contain a 0 or a 1.

Let $R(A) = 0$ if A doesn't terminate on the empty tape. Otherwise, let $R(A)$ be the distance between the starting cell and the cell where the head was located at the end of computation.

Let $h(n)$ be the maximum of $R(A)$ over all A with exactly n states. Is h recursive?

Chapter 8: Sets and reductions

In multiple exercises:
- $D(f)$ denotes the domain of f, i.e., the set of all inputs for which f is defined.
- $V(f)$ denotes the values of f, i.e., the set $\{y \mid \exists x : f(x) = y\}$.
- $G(f)$ denotes the graph of f, i.e., the set $\{\ (x, y) \mid x \in \mathbb{N}\ \wedge\ y = f(x)\}$.

Exercise 8.1
Prove or disprove: If f is a partial recursive function, its domain $D(f)$ is a recursively enumerable set.

Exercise 8.2
Prove or disprove: If f is a partial recursive function, its set of values $V(f)$ is a recursively enumerable set.

Exercise 8.3
Prove or disprove: If f is a primitive recursive function, its graph $G(f)$ is a primitive recursive set.

Exercise 8.4
Prove or disprove: If f is a total function and its graph $G(f)$ is a primitive recursive set, then f must be primitive recursive.

Exercise 8.5
Prove or disprove: If f is a recursive function, its graph $G(f)$ is a recursive set.

Exercise 8.6
Prove or disprove: If f is a total function and its graph $G(f)$ is a recursive set, then f must be recursive.

Exercise 8.7
Prove or disprove: If f is a total function and its graph $G(f)$ is a recursively enumerable set, then f must be recursive.

Exercise 8.8
Let A be a subset of $\mathbb{N}$ such that $A \neq \emptyset$ and $A \neq \mathbb{N}$. Prove or disprove: A is recursive iff $A \leq_m \{47\}$.

Exercise 8.9
Prove that if A is a recursively enumerable set such that $A \leq_m \overline{A}$ then A is recursive.

Exercise 8.10
Find a non-recursive set B such that $B \leq_m \overline{B}$.

Exercise 8.11
Let $\varphi_0, \varphi_1, \ldots$ be a computable numbering of all unary partial recursive functions, and let $c_k : \mathbb{N}^k \to \mathbb{N}$ be a computable k-ary pairing function (i.e., c_k is recursive and one-to-one).

We may now define a new sequence of k-ary functions as follows: let ψ_i be the function defined as:

$$\forall x_1, \ldots, x_k : \psi_i(x_1, \ldots, x_k) = \varphi_i(c_k(x_1, \ldots, x_k))$$

Prove or disprove: Each k-ary partial recursive function appears somewhere in this new sequence.

Exercise 8.12
Prove that each partial recursive function with a recursive domain has a recursive completion.

Exercise 8.13
In the proof that the universal function U has no recursive completion, we assume that it does and we use this recursive completion U' to define a function f and later to derive a contradiction.

Why won't we get the same contradiction if we define f in terms of the original U instead of U'?

Exercise 8.14
Prove that the universal set $UNIV = \{(n, x) \mid \varphi_n(x) > 0\}$ is m-complete.

Exercise 8.15
Let $W_0, W_1, \ldots$ be any recursive numbering of all recursively enumerable sets.

Prove that the set $\{n \mid 47 < |W_n|\}$ is m-complete.

Exercise 8.16
Let $MONOTONE = \{n \mid \varphi_n$ is total and $\forall x : \varphi_n(x) \leq \varphi_n(x+1)\}$.

Prove that $\overline{HALT} \leq_m MONOTONE$.

Exercise 8.17
Let A be any simple set and c any computable pairing function. Consider the set $B = \{c(a, n) \mid a \in A \wedge n \in \mathbb{N}\}$. For each of the following statements, decide whether it is always, sometimes, or never true.
- B is recursively enumerable.
- B is recursive.
- B is creative. (In other words, B is m-complete.)
- B is simple.

Exercise 8.18
The first quadrant is divided into a grid of unit square cells. Each cell is either white or black. For each row and each column we may now define the set of black cells and the set of white cells in this row. E.g., white cells in row 47 are $WR_{47} = \{y \mid (47, y)$ is white$\}$.

Is there a coloring such that none of these sets is recursive?

Is there a coloring such that none of these sets contains an infinite recursive subset?

Is there a coloring such that all of these sets are recursively enumerable and none of these sets contains an infinite recursive subset?

Exercise 8.19
Prove or disprove: a set $A \subseteq \mathbb{N}$ is recursive iff there is a recursive function f such that:
- $\forall a \in A : \ \exists n \in \mathbb{N} : \ f(n) = a$
- $\forall n \in \mathbb{N} : \ f(n) \in A$
- $\forall n \in \mathbb{N} : \ f(n) < f(n+1)$

Exercise 8.20
Prove or disprove: There is a set $A \subseteq \mathbb{N}$ such that neither A nor $\mathbb{N} - A$ contains an infinite increasing arithmetic progression.

Exercise 8.21
Prove or disprove: There is an infinite recursively enumerable set that has no infinite primitive recursive subset.

Exercise 8.22
A set is called *immune* if it is infinite and doesn't contain an infinite recursively enumerable subset. Clearly, the complement of a simple set must be immune.

Find an immune set such that its complement is *not* simple.

Exercise 8.23
Is there an immune set such that its complement is also immune?

Chapter 9: Gödel's Theorem

Exercise 9.1
In our first-order predicate logic, construct a string that will correspond to the predicate "x is a composite number".

Exercise 9.2
In our first-order predicate logic, construct a string that will correspond to the statement "there are infinitely many primes".

Exercise 9.3
Find some formal system in which the set of provable statements is recursive.

Exercise 9.4
A set of axioms is called *minimal* if neither of them can be proved from the others.

Consider a decision problem where the instance is a pair (A, D) such that A is a finite set of axioms and $D(x, y, z)$ is a ternary recursive predicate "the string y is a proof of the theorem x that only uses the axioms in the set z". In the decision problem, the goal is to decide whether the set A is minimal w.r.t. the predicate D.

Show that this problem is not decidable. Also show that its complement is partially decidable – i.e., that we are able to enumerate all instances (A, D) where A is not minimal.

Chapter 10: Recursion Theorems

Exercise 10.1
Choose any modern programming language and write a program that will output its own source code.

Exercise 10.2
Choose any modern programming language and write a program that will output its own source code, but replacing all vowels with 'a's.

Exercise 10.3
Choose any modern programming language and write two programs X and Y such that X outputs the source code of Y and vice versa.

(Is there a connection to the Recursion theorems? Would it be possible to show that this task can be solved in all decent programming languages?)

Exercise 10.4
Use the Rogers version of the Second recursion theorem to show that there is a program that prints its own source code.

Exercise 10.5
In the Second recursion theorem, can there sometimes be more than one fixpoint?

E.g., is it possible that in the Rogers version for some specific transformation f there are two programs n_1 and n_2 that compute different partial recursive functions, but n_1 computes the same function as $f(n_1)$ and n_2 computes the same function as $f(n_2)$?

Exercise 10.6
Read about http://www.ioccc.org/2012/endoh2/endoh2.c (details: http://www.ioccc.org/2012/endoh2/hint.html). Then, think about how to implement a similar program (not necessarily with the formatting, just a similar behavior is enough).