1. We consider two different languages that in some sense represent strings representing correct additive equalities. You can assume that the input is on a one-way read-only tape. Show that one of these problems can be solved by a finite state machine. Show from first principles (so no using results like the Pumping Lemma) that one of these problems can not be solved by a finite state machine. To show impossibility, you should assume that there is a finite automata $M$ with $k$ states that accepts this language. Then consider arbitrarily large strings of some particular type, and argue $M$ has to mess up on some string of length greater than $k$.

(a) The language consists of strings over the symbols: 0, 1, + and =. The languages consists of strings of the form $x + y = z$, where natural numbers $x$, $y$ and $z$ encoded in binary in the standard way, and where $x$ plus $y$ is indeed equal to $z$. So for example the string $10100 + 111 = 11011$ of 11 symbols is in the language.

(b) The language is a string of the following 8 symbols (so each line is a different symbol)

<table>
<thead>
<tr>
<th>[000]</th>
<th>[001]</th>
<th>[010]</th>
<th>[011]</th>
<th>[100]</th>
<th>[101]</th>
<th>[110]</th>
<th>[111]</th>
</tr>
</thead>
</table>

where the first column plus the second column is equal to the third column. So to reuse our example from the previous subproblem, the 5 symbol input

$$[101][001][110][011][011]$$

is in the language because $10100 + 111 = 11011$.

So this shows that whether one whether a finite state machine can add, depends on the definition of notion of addition that you use.

Due Friday January 12 (You need not use LaTeX, your solutions may be handwritten.)

2. Problem 1.1 from the text. That is, construct a complete formal description of a Turing machine (using the formalization given in section 1.2) that adds two binary numbers (you need not do multiplication). Assume that input is of the form NUMBER1#NUMBER2, e.g. 1011#110. You should write your output on a second tape.

Due Friday January 12 (You need not use LaTeX, your solutions may be handwritten.)

3. Read the description of the Post Correspondence Problem

https://en.wikipedia.org/wiki/Post_correspondence_problem
including the sketch of the proof that there is no algorithm to solve this problem. The article ends with “There are a number of details to work out, such as dealing with boundaries between states, making sure that our initial tile goes first in the match, and so on, ..”

Explain how to deal with these two details, that is, how to deal with boundaries between states, and how to make sure that the initial tile goes first. List any other details that you think need to be dealt with, and explain how they can be dealt with. Due Wednesday January 17.

4. (a) (warm up) Show by reduction from the Halting Problem that there is no Turing machine that takes as input a Turing machine $M$, and determines whether the language $L(M)$ accepted by $M$ is the empty language.

(b) (warm up) Show by reduction from the Halting Problem that there is no Turing machine that takes as input a Turing machine $M$, and determines whether the language $L(M)$ accepted by $M$ is the language of every string over the input alphabet.

(c) (warm up) Show by reduction from the Halting Problem that there is no Turing machine that takes as input a Turing machine $M$, and determines whether the language $L(M)$ accepted by $M$ includes the string 11110.

(d) Let $P$ be some property of languages. Further assume there is a Turing machine $M_1$ that accepts a language $L_1$ that has property $P$, and a Turing machine $M_2$ that accepts a language $L_2$ that does not have has property $P$. Show by reduction from the Halting Problem that there is no Turing machine that takes as input a Turing machine $M$, and determines whether the language $L(M)$ accepted by $M$ satisfies property $P$.

(e) Explain why the first three subproblems are consequences of the fourth subproblem.

Due Friday January 19. Note that some students may find the fourth subproblem somewhat tricky. Do the other subproblems first, and then make a good faith effort at the fourth subproblem. At the very least, at least understand the statement of the fourth subproblem.

5. Show that the following two definitions of recursively enumerable are logically equivalent:

(a) A language $L$ is recursively enumerable iff there is a Turing machine $M$ such that if $x \in L$ then $M$ accepts $x$ and if $x \notin L$ then $M$ loops forever on $x$.

(b) A language $L$ is recursively enumerable iff there a Turing machine $M$, with a read/write tape that is initially empty and a write-only output tape, such that only elements of $L$ are written to the output tape, and every element of $L$ is eventually written to the output tape.

Due Monday January 22
6. Consider a proof rule such as:

**Proof Rule:** From the statement $\forall x P(x)$ one can deduce the countably infinite number of statements $P(0), P(1), P(2), P(3), \ldots$.

Somewhat more formally, you can assume that there is a Turing machine $T$ that takes as input a statement $S$, and will output a list all the statements one can conclude from $S$ using this proof rule. Here “list” means what it means within the context of the definition of recursively enumerable, $T$ will only output statements we can deduce from $S$ by the proof rule, and every statement that we can deduce from $S$ by the proof rule will eventually be listed. Prove that the following language is still recursively enumerable:

$$L = \{ S \mid \text{statement } S \text{ is provable from the finite set } A \text{ of axioms} \}$$

Hint: Now the nodes in the tree of all proofs can have infinitely many children. So breadth first search will no longer work. So you will have to find another way to search this tree so that every node is eventually reached.

Due Monday January 22

7. (a) Consider the following one tape Turing Machine $M$ that halts iff the number of 1’s on the input tape is odd.

- **tape alphabet** = \{0, 1, space\}
- **start state** = $q_0$
- **states** = $Q = \{ q_0, q_1, q_{halt} \}$
- **halt state** = $q_{halt}$. So the machine stops running if it ever reaches state $q_{halt}$.

**Transitions:**
- $(q_0, 0) \rightarrow (q_0, 0, right)$
- $(q_0, 1) \rightarrow (q_1, 1, right)$
- $(q_0, \text{space}) \rightarrow (q_0, \text{space}, \text{stay})$. So this is an infinite loop.
- $(q_1, 0) \rightarrow (q_1, 1, right)$
- $(q_1, 1) \rightarrow (q_0, 0, right)$
- $(q_1, \text{space}) \rightarrow (q_{halt}, \text{space}, \text{stay})$

Show one valid computation history $H$ for the input $I = 101$ for Turing machine $M$. Note that the computation history will have the form #$C_0#C_1#C_2# \ldots #C_k#$ where $k$ is the number of steps that $M$ runs on $I$ and $C_i$ is the configuration of $M$ on $I$ after $i$ steps. Note that there technically there is more than one valid computation history depending on how many spaces at the right of the tape are included in each configuration.

(b) Give the Godel sentence $S$ for the $M$ and $I$ in the previous subproblem. That is, $S$ should be a first order sentence in the language of number theory that will be true iff and only if $M$ halts on $I$.

Hint: To get you started, $S$ should ask whether there exists a number $H$, which one can interpret as a computation history of $M$ on $I$. Conditions then need
to be added to $S$ to check that $H$ is a valid computation history of $M$ on $I$ that shows that $M$ halts. You are strongly encouraged to use macros (see http://www.computerhope.com/jargon/m/macro.htm), which of course you have to define.

Some macros that might be useful are:

- $\text{PLACE}(j)$ represents an arithmetic expression that returns the digit in location $j$ in $H$.
- $\text{SAME}(i, j, k, l)$ represents a logical expression that will be true iff digits $i$ through $j$ are identical to digits $k$ through $l$.
- $\text{STATE}(i)$ represents a local expression that will be true iff only digit $i$ of $H$ represents a state in $Q$.
- $\text{TABLE}(i, j)$ represents a local expression that will be true iff digits $i$, $i+1$ and $i+2$ in $H$ represent a tape symbol, state in $Q$ and a tape symbol respectively, and digits $j$, $j+1$ and $j+2$ in $H$ evolve properly from digits $i$, $i+1$, and $i+2$ according to $M$. So for example if digits $i$, $i+1$ and $i+2$ were $0, q_0, 1$ respectively, then $\text{TABLE}(i, j)$ would only be true iff digits $j$, $j+1$, $j+2$ were $0, 1, q_1$.

You are welcome to use other macros.

Due Wednesday January 24

8. Consider first order logical sentences of arithmetic where you are only allowed to use the arithmetic operators $=$ and $+$. Without loss of generality we may assume that the only logical operators are AND, represented by $\land$ and NOT, represented by $\neg$, and that all quantifiers appear first. So you might have a formula like:

$$\forall x \exists y \forall z \exists w \ (x + y = z) \land \neg(y + z = w + x)$$

Our goal here is to show that there is an algorithm to accept exactly the language of true formula. So the proof Godel’s Incompleteness Theorem doesn’t work if addition is the only allowed mathematical operation. We will use the following strategy.

Recall that we know how to build a finite state machine $L$ that accepts tuples $(x, y, z, w)$ properly encoded that satisfy $(x + y = z)$ and a finite state machine $M$ that accepts tuples $(x, y, z, w)$ properly encode that satisfy $(y + z = w + x)$.

(a) Explain how to construct a finite state machine $N$ that accepts tuples $(x, y, z, w)$ properly encoded that satisfy $\neg(y + z = w + x)$ from finite state machine $M$.

Hint: This is completely trivial. If you need more than a sentence to explain how to do this, you are not on the right track.

(b) Explain how to construct a finite state machine $P$ that accepts tuples $(x, y, z, w)$ properly encoded that satisfy $(x + y = z) \land \neg(y + z = w + x)$ from finite state machine $L$ and $N$.

Hint: The states in $P$ will be of the form $(l, n)$ where $l$ is a state in $L$ and $n$ is a state in $N$.  

4
(c) Now consider the quantifiers from the inside out. Explain how to construct a finite state machine \( Q \) that accepts tuples \((x, y, z)\) properly encoded with the property that \( \exists w \ (x + y = z) \land \neg (y + z = w + x) \) from finite state machine \( P \).

Hint: The states in \( Q \) will be subsets of states in \( P \).

(d) Explain how to construct a finite state machine \( R \) that accepts tuples \((x, y)\) properly encoded with the property that \( \forall z \exists w \ (x + y = z) \land \neg (y + z = w + x) \) from finite state machine \( Q \). Hint: The states in \( R \) will be subsets of states in \( Q \).

(e) Note that the same idea can be used to construct a finite state machine \( S \) that accepts strings \( x \) properly encoded with the property that \( \forall y \forall z \exists w \ (x + y = z) \land \neg (y + z = w + x) \) from finite state machine \( R \).

(f) Give an algorithm that, given as input a finite automata \( S \), can determine whether there is any string that \( S \) accepts.

(g) Give an algorithm that decides whether a given first order sentence in number theory, that only involves arithmetic operations + and = i and has the quantifiers appearing first, is true. More or less all the main ideas are contained in the previous subproblems. You more or less just need a couple of sentences to put everything together.

Due Friday January 26

9. What happens to the entropy/information of objects that fall into a black hole is an interesting subject. Watch and write a one paragraph summary of http://people.cs.pitt.edu/~kirk/cs1511/BlackHoleEntropy.mp4.

Due Monday January 29

10. Recall the Definition of entropy of a probability distribution \( X \):

\[
H(X) = \sum_x p(x) \log(1/p(x))
\]

Here we use the convention that capital letters are probability distributions and lower case letters are a possible outcome. So here \( x \) is some possible value of the random variable \( X \). So if the setting that \( X \) is a random coin flip, then \( x \) might either be Heads or Tails.

We now define the conditional entropy of probability distributions \( X \) and \( Y \) as:

\[
H(X \mid Y) = \sum_y p(y) \sum_x p(x \mid y) \log 1/p(x \mid y) = \sum_{x,y} p(x,y) \log 1/p(x \mid y)
\]

Here \( p(x \mid y) \) is the probability of event \( x \) given event \( y \) and \( p(x,y) \) the probability of event \( x \) and event \( y \) both happening. Further define the mutual information between \( X \) and \( Y \) as:

\[
I(X;Y) = H(X) - H(X \mid Y)
\]

Intuitively, the mutual information \( I(x;y) \) measures the average reduction in uncertainty (measured in bits) about \( x \) that results from learning the value of \( y \).
(a) Now assume $x$ is a bit that a sender wants to send to a receiver over a noisy channel, and let $y$ be the bit received by the receiver. Because the channel is noisy, $y$ may not equal $x$. Let $X$ be the probability distribution where $x$ is 0 with probability $1/3$ and $x$ is 1 with probability $2/3$. Assume the noisy channel has the following properties $P(y = 0 \mid x = 0) = .9$, $P(y = 1 \mid x = 0) = .1$, $P(y = 0 \mid x = 1) = .2$, and $P(y = 1 \mid x = 1) = .8$. Let $Y$ be the probability distribution for the received bit.

i. What is $H(X)$ for this example?

ii. What is the probability distribution $Y$? That is, what is the probability that $y=0$ and what is the probability that $y=1$?

iii. What is $H(Y)$ for this example?

iv. What is $H(X \mid Y)$ for this example?

v. What is $H(Y \mid X)$ for this example?

vi. What is $I(X; Y)$ for this example?

vii. What is $I(Y; X)$ for this example?

Hint: You should find that $I(X; Y) = I(Y; X)$.

viii. Restate in plain English what it means that $I(X; Y) = I(Y; X)$ in this setting.

(b) Now consider arbitrary probability distributions $X$ for $x$ and $Y$ for $y$.

i. Prove that $I(X; Y) \geq 0$.

ii. Prove $I(X; Y) = I(Y; X)$.

FYI, Shannon’s noisy channel coding theorem says: You can get about $\max_x I(X; Y)$ bits of information through to the receiver for each bit sent.

Due Monday January 29

11. Say that a string $x$ of $n$ bits is semi-incompressible if $K(x) \geq \sqrt{n}$. Here $K(x)$ is the Kolmogorov complexity of $x$. Show that the set of semi-incompressible strings is not computable.

Show that there are only finitely many incompressible strings that have the property that the number of bits that 0 in the string is equal to the number of bits that are 1 in the string. Recall that a string is incompressible if its Kolmogorov complexity is at least its length.

Hint: You can use without proof the fact that if you flip $n$ fair independent coins, the probability that the number of heads is equal to the number of tails is approximately $1/\sqrt{n}$.

• Show that the set of incompressible strings contains no infinite subset that is recursively enumerable.

• Show that the set of compressible strings is recursively enumerable. A string is compressible if it is not incompressible.

Due Wednesday January 31
12. (Extra credit for undergraduates; Required for graduate students) Show that for any $c > 0$, there exist strings $x$ and $y$ exist such that $K(xy) > K(x) + K(y) + c$. Here $K(x)$ is the Kolmogorov complexity of $x$.

Due Wednesday January 31

13. Prove that the following two definitions of $TIME(T(n))$ are equivalent in the sense that they contain exactly the same languages.

Definition 1: $TIME(T(n))$ is the set of all languages $L$ such that there exists a Turing machine $M$ such that (1) $M$ accepts $x$ iff $x \in L$ and (2) for all but finitely many $x$, $M$ on $x$ halts in $T(|x|)$ steps.

Definition 2: $TIME(T(n))$ is the set of all languages $L$ such that there exists a Turing machine $N$ and a number $b$ such that (1) $N$ accepts $x$ iff $x \in L$ and (2) for all $x$, $N$ on $x$ halts within $b \cdot T(|x|)$ steps.

Hint: You need to show that if you have a Turing machine $M$ that satisfies the first condition, then you construct from it a Turing machine $N$ that satisfies the second condition. This direction is pretty straight-forward. And you need to show that if you have a Turing machine $N$ that satisfies the second condition, then you construct from it a Turing machine $M$ that satisfies the first condition. This direction is a bit trickier, and you have to show how to speed up any Turing machine by a constant factor. The main insight one needs to accomplish this is to use a larger tape alphabet size.

Due Friday February 2

14. Consider a programming language mini-Java that only has one type of loop, and the number of iterations of the loop must be determined when the loop is first encountered. So a loop statement might look like "Repeat $x$ times", and the variable $x$ is evaluated when the statement is first reached. You can assume that the program has a variable $n$ that is instantiated to the input size when the program starts running (otherwise, you couldn’t even read the input). Note that all mini-Java programs must halt on all inputs. Show by diagonalization that there is a language accepted by a Java program that is not accepted by any mini-Java program.

You can think of any string as a mini-Java program. If the string is not a syntactically correct mini-Java program, then think of it as a mini-Java program that rejects all inputs.

Due Monday February 5

15. (a) Let $A$ be the language of properly nested parentheses. So for example, $()$ and $((()))()$ are in $A$ but $)$ is not in $A$. Show that $A$ can be accepted by a log-space Turing machine.

Hint: This is very easy. It is sufficient to store one number between 0 and $n$ (the number of parentheses in the input) on the work tape while making one pass over the read-only input tape.
(b) Let $B$ be the language of properly nested parentheses and brackets. So for example, $(((\text{ })))(\text{ })$ is in $B$, but $([\text{ }])$ is not in $B$. Show that $B$ can be accepted by a log-space Turing machine.

Hint: I don’t see how to do this by making only one pass over the input tape. Start by making sure that $[\text{ }]$ and $\text{ }]$ don’t appear in the input. Then rule out substrings like $[()()]$ and $(())()$. Then continue in this manner.

Due Monday February 5

16. Assume a log-space reduction from a language $A$ to a language $B$.

So more precisely, there is a Turing machine $T$ with three tapes, a read-only input tape, a read/write work tape, and a write-only output tape. $T$ only uses log of the input size many cells on the read/write work tape. Further $T$ never backs up the tape head on the write-only output tape, so the tape head on the write-only tape either stays in position or moves to the right. The machine $T$ has the property that a string $x$ is in $A$ iff the contents of the write tape, when $T$ ends computation on input $x$, is in $B$.

Now show that if there is a log space Turing machine $S$ that accepts $B$, then there is a log space Turing Machine $U$ that accepts $A$.

Hint: I think its harder to understand the question than it is to actually solve the problem once you really understand the question. The machine $U$ has to simulate the computation of both machines $S$ and $T$ while only using log space. Note that this is not trivial because $U$ does not have enough work tape to write down the output of $T$. This is a good problem to show how to make use of the fact that space is reusable.

Due Wednesday February 7

17. (a) Define EXPSPACE to be the set of languages $L$ where there exists a Turing machine $M$, and integer $k$ such that $M$ accepts exactly the language $L$ and using space at most $2^n^k$ on all inputs of size $n$. Define a language $C$, and show that $C$ is complete for EXPSPACE under polynomial time reductions.

Hint: This is should be more of less line by line the same logic as showing that PSPACE has a complete language, which we did in class.

(b) Define EXPSPACE to be the set of languages $L$ where there exists a Turing machine $M$, and integer $k$, and an integer $c$ such that $M$ accepts exactly the language $L$ and using space at most $c^n^k$ on all inputs of size $n$. Define a language $C$, and show that $C$ is complete for EXPSPACE under polynomial time reductions.

Hint: First ask yourself what issue arises here that didn’t arise in the previous subproblem. Then ask yourself how to address this issue. There are very easy ways to address this issue.

Due Wednesday February 7

18. Read the description of the generalized geography and the proof that it complete for PSPACE under polynomial time reductions at:

(a) Show the Generalized Geography game instance that would result from applying the reduction to the quantified Boolean formula (you can draw this by hand, you don’t need to use LaTeX):

$$\exists w \forall x \exists y \forall z (x \lor \neg y \lor z) \land (w \lor \neg y \lor \neg z) \land (\neg w \lor y \lor z) \land (y \lor z \lor \neg x)$$

(b) Who has a winning strategy for this instance of the Generalized Geography game, the first player or the second player? Give a brief justification for your claim.

Due Friday February 9