- Problems marked with **P** are graded on progress, which means that they are graded subjectively on the perceived progress toward a solution, rather than **solely** on correctness.
- Problems marked with a **G** are group problems. A group of 1-3 students may work on these together and turn in one assignment for the entire group. Each member listed should have made real contributions. The group problems will be turned in separately on gradescope.

For bonus and most extra credit questions, we will not provide any insight during office hours or Piazza, and we do not guarantee anything about the difficulty of these questions. We strongly encourage you to typeset your solutions in LaTeX. If you collaborated with someone, you must state their name(s). You must write your own solution for all problems and may not look at any other student's write-up.

G 1. (8) Code writing code.

(a) Write a program in C++ named A.cpp which, when run, creates a C++ program named "print.cpp". print.cpp should, when complied and run, print whatever argument was passed to A.cpp on the command line (you can assume it's a single word with no spaces).

Solution:

A.cpp > main(int, char * [])

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main(int argc, char *argv[]){
    string input=argv[1];

    ofstream output("print.cpp");

    output<<"#include <iostream>\n";
    output<<"using namespace std;\n";

    output<<"int main(){\n";

    output<<"cout<<\""<<input<<"\";\n";

    output<<"}\n";


}
```

(b) When using Turing reductions to show a language is undecidable, we may write a program which constructs a program. Consider the Turing reduction of $L_{HALT} \leq_{\mathrm{T}} L_{\varepsilon-HALT}$ done in lecture 10. In that case H took as input $(\langle M \rangle, x)$ and our program M'

took as input w and ignored it, while hardcoding x into M'. Find analogies for each of x, M', and w in that proof to elements of the code you wrote for part a.

Solution: x is the command line argument to A.cpp which is the string input variable in our code. M' is print.cpp which is the function made by A.cpp using the ofstream output. w is whatever command line arguments are passed into print.cpp which is analogous because we have made print.cpp so that it ignores all command line arguments. A.cpp does not run print.cpp it only creates the cpp file for it to be run.

2. (20) Prove or disprove the following statements about **Turing reducibility**.

   (a) If $L \leq_{\mathrm{T}} L'$ for some language L', then L is decidable.

   Solution:

   (b) For any decidable language L and an arbitrary language L', $L \leq_{\mathrm{T}} L'$

   Solution:

   (c) For all $L, L \leq_{\mathrm{T}} \overline{L}$

   Solution:

   (d) Let $L_1 = \{\langle a, b\rangle : a, b \in \mathbb{Z}^+, a \mid b \text{ and } b \mid a\}$ and $L_2 = \{\langle G = (V, E), s, t\rangle : G \text{ is a simple graph, } s, t \in V, \text{ and there is no path between } s \text{ to } t \text{ in } G\}$. We have $L_1 \leq_{\mathrm{T}} L_2$.

   Solution:

   (e) Let $L_{\varepsilon\text{-}ACC} = \{\langle M\rangle : M \text{ is a TM that accepts } \varepsilon\}$. We have $L_{\varepsilon\text{-}ACC} \leq_{\mathrm{T}} L_{\mathrm{ACC}}$.

   Solution:

3. (24) Determine, with proof, which of the following languages are undecidable. That is, for an undecidable language, present a proof via Turing reduction of its undecidability, and for a decidable language, exhibit a decider for that language.

   (a) $L_1 = \{\langle M_1, M_2\rangle : M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) \cap L(M_2) \neq \emptyset\}$.

   Solution:

   (b) $L_2 = \{\langle M\rangle : M \text{ is a TM that loops on the input string "eecs376"}\}$.

   Solution:

(c) $L_3 = \{(\langle M_1, M_2 \rangle) : M_1 \text{ and } M_2 \text{ are TMs such that } |L(M_1)| + |L(M_2)| \text{ is odd}\}$.

Solution:

4. (16) Prove whether each of the following statements about **undecidable** languages are *necessarily true*, *sometimes true*, or *never true*.

   (a) For an undecidable language L, its complement ($\overline{L}$) is undecidable.

   Solution:

   (b) For two undecidable languages L_1 and L_2, their intersection ($L_1 \cap L_2$) is undecidable.

   Solution:

   (c) For two undecidable languages, L_1, L_2, their union, ($L_1 \cup L_2$) is undecidable.

   Solution:

   (d) For two undecidable languages, L_1, L_2, $(L_1 \cup (\overline{L_1} \cap \overline{L_2}) \cup L_2)$ is undecidable.

   Solution:

5. (12) Let A, B, and C be languages over Σ. Prove or disprove the following.

   (a) If $A \leq_T B$ and $B \leq_T C$, then $A \leq_T C$.

   Solution:

   (b) If $A \leq_T B$ and $A \leq_T C$, then $B \leq_T C$.

   Solution:

   (c) If $A \leq_T C$ and $B \leq_T C$, then $A \cap B$ is undecidable when C is undecidable.

   Solution:

G 6. (20) Two models M_1, M_2 of computation are said to be equivalent in power if and only if M_1 can simulate M_2 and M_2 can simulate M_1. See lecture 7 or discussion notes 4 for a (sketch of a) proof that a 1-tape Turing Machine is equivalent in power to a 2-tape Turing Machine. The course notes has a full, formal proof of the equivalence.

Define the 3-76 Turing Machine be a Turing Machine that behaves exactly like our model from lecture, *except* that it can only move its tape head to the left or right by increments of 3 or 76, rather than by increments of 1. Show that a 3-76 TM is equivalent in power to the 1-tape Turing Machine described in lecture.

Solution:
To Show that a 3-76 TM is equivalent in power to the 1-tape Turing Machine we'll show:
1. A 3-76 TM can compute as much as a 1 tape TM can
2. A 1 tape TM can compute as much as a 3-76 TM can

A 3-76 TM can compute as much as a 1 tape TM can:
To show that a 3-76 TM can reach every position on the tape that a 1 TM can reach we can show that a 3-76 machine can increment its tape head by 1 by incrementing by 76 and then decrementing by 3 25 times. Additionally, a 3-76 machine can decrement its tape head by 1 by incrementing by 76 twice and then decrementing by 3 51 times. This method of incrementing and decrementing would be the transition function for the 3-76 TM in order for it to perform at the same power as a 1 TM.
A 1 tape TM can compute as much as a 3-76 TM can:
A 1 tape TM can do exactly what a 3-76 TM can because you could implement loops in the 1 tape TM so that the tape is only incremented by quantities of 3 or 76 by making states that continue to increment the tape head until it is known they have incremented by either 3 or 76, We could do this by having the transition function of the 1-tape TM use a loop with a count variable for the number of times it has incremented or decremented.