What problems are beyond our capacity to solve? In this last assignment, you’ll explore the limits of computation.

Exercise 1

In class, we saw that $A_{TM}$ is undecidable but Turing-recognizable. From a programming perspective, this means that it’s possible to write a procedure `will_accept(program, input)` that takes as its arguments string representations of a program and that program’s input and has the following return values:

- If the program would accept that input, `will_accept` must return true.
- If the program would not accept the input, `will_accept` may return false or it may go into an infinite loop.

Now consider the following program, `ow_my_brain.py`:

```python
def will_accept(program, input):
    # An implementation of a recognizer for ATM:
    # Does the program accept the input?
    ...

def main(my_input):
    # Read "ow_my_brain.py"
    my_code = my_source()

    if will_accept(my_code, my_input):
        return False  # i.e., reject
    else:
        return True  # i.e., accept
```

Use proof by contradiction to show that this program must loop infinitely on all inputs.

It's fine for this to be a casual “proof sketch” as long as your thinking is clear.
Exercise 2

When you log onto a website with a password, you hope that your password is the only possible input that will log into your account. If you have the source code for the password checking system, could you tell whether your password was the only one it would accept?

Let’s frame the question using Turing machines. If we wanted to build a TM password checker, “entering your password” would correspond to starting up the TM on some string, and “gaining access” would mean that the TM accepts your string. Suppose your password is the string love. A TM that would work as a valid password checker would be a TM $M$ where $L(M) = \{\text{love}\}$. That is, $M$ accepts your password, love, and it doesn’t accept anything else.

Given a TM, is there some way you could tell if the TM is a valid password checker? This question corresponds to deciding if the encoding $\langle M \rangle$ of a TM $M$ is in the language $\text{CHECKER}$:

$$\text{CHECKER} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \{\text{love}\} \}$$

We’ll prove the language $\text{CHECKER}$ is undecidable and, thus, there’s no algorithm that can check whether a TM is suitable as a password checker.

Suppose for the sake of contradiction that $\text{CHECKER}$ is decidable. This means that we could write some function $\text{is\_password\_checker}$ with the following properties:

- If $p$ is the source of a program that accepts just the string love, then calling $\text{is\_password\_checker}(p)$ will return True.
- If $p$ is not the source of a program that accepts just the string love, then calling $\text{is\_password\_checker}(p)$ will return False.

We can prove that $\text{CHECKER}$ is undecidable by building a self-referential program that uses $\text{is\_password\_checker}$ to obtain a contradiction. Here’s a first try:

File c.py:

```python
def is_password_checker(p):
    ...some implementation...

def main(my_input):
    # Read c.py
    checker = my_source()

    if is_password_checker(checker):
        # Reject the password my_input
        return False
    else:
        # Accept the password my_input
        return True
```

1 For instance, a system that allows a “master key” password for logging in to any account has a major security vulnerability!

“All right now, what are the three most common[ly] used passwords?”

“‘love’, ‘secret’, and, uh, ‘sex’, but not in that order necessarily, right?”

Hackers, 1995
a. Suppose this program (c.py) is a password checker. Briefly explain why running this program leads to a contradiction.

b. Suppose this program is not a password checker. Briefly explain why running this program does not lead to a contradiction.
c. Our goal in building a self-referential program is for the program to cause a contradiction regardless of whether it's a password checker. As you saw in (b), this program does not cause a contradiction if it is not a password checker. Modify the program so that it does. Briefly explain why your modified program is correct.

You've now (informally) proven that CHECKER is undecidable. Thus there's no way for a computer to decide if an arbitrary program is a password checker or if it contains a backdoor! :-(
Exercise 3

Consider the language

$$A_{\epsilon_{TM}} = \{\langle M \rangle \mid M \text{ is a TM that accepts } \epsilon\}.$$ 

Use proof by reduction to show that $A_{\epsilon_{TM}}$ is undecidable.