CMPU 240 · Spring 2026

Assignment 10

Submissions due: 5 May, 11:59 p.m.
Corrections due: 7 May, 11:59 p.m.

Exercise 1

In class, we saw that A_{TM} is undecidable – but it *is* Turing-recognizable! From a programming perspective, this means that it's possible to write a procedure `will_accept(program, w)` 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.

Use proof by contradiction to establish that the following program, `ow_my_brain`, must loop infinitely on all inputs.

It's fine for this to be a casual "proof sketch" as long as your thinking is clear.

```
def will_accept(program: str, w: str) -> bool:
   """Return True if and only if the given program accepts
   the input w."""
   ...

def ow_my_brain(input: str) -> bool:
    me = my_source()
    if will_accept(me, input):
        return False  # i.e., reject
    else:
        return True   # i.e., accept
```

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.[1] 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 $L(M) = \{\texttt{love}\}$. That is, M accepts your password, `love`, and it doesn't accept anything else.

[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 used passwords?

—`love`, `secret`, and, uh, `sex`, but not in that order necessarily, right?

Hackers, 1995

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 this language:

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

We'll prove the language $CHECKER$ is undecidable and, thus, there's no algorithm that can check whether a TM – or a program – is suitable as a password checker.

Suppose for the sake of contradiction that $CHECKER$ is decidable. This means that we could write a function `is_password_checker` with the following properties:

- If p is the source of a program that accepts just the string `love`, then calling `is_password_checker(p)` will return `True`.
- If p is not the source of a program that accepts just the string `love`, then calling `is_password_checker(p)` will return `False`.

We can prove that $CHECKER$ is undecidable by building a self-referential program that uses `is_password_checker` to obtain a contradiction. Here's a first try:

```python
def is_password_checker(p: str) -> bool:
    """Return True if p is a password checker.
    Otherwise, return False."""
    ...

def c(input: str) -> bool:
    checker = my_source()
    if is_password_checker(checker):
        # Reject the password my_input
        return False
    else:
        # Accept the password my_input
        return True
```

a. Suppose this program (`c`) 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.

Hint: What language does it recognize in this case?

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. Consider the following modified self-referential program:

```
def c(input: str) -> bool:
    checker = my_source()
    if is_password_checker(checker):
        return False
    else:
        if input == "love":
            return True
        else:
            return False
```

Briefly explain why this program *does* cause a contradiction in both cases.

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 following language:

$$ENTERS = \{\langle M, w, q\rangle \mid q \text{ is a state in Turing machine } M, \text{ and } M \text{ enters } q \text{ when run on } w\}$$

Show that *ENTERS* is undecidable by completing the proof reducing A_{TM} to it:

Take a look at the handout on reductions for examples of this kind of proof.

PROOF By contradiction. Assume that *ENTERS* is decidable. This means there is a TM D that decides it. We can use D to construct a TM A that decides A_{TM} ...

Hint: Keep this one simple! You don't need to construct an additional Turing machine to give to D; just use M. Since we have the encoding of M, we can tell what its accept state is.

...Because we know that A_{TM} is undecidable, this is a contradiction. The assumption must be false, and *ENTERS* is undecidable. ■

Exercise 4

Like you did in Exercise 3, prove that the following language is undecidable by reducing A_{TM} to it:

$$INFINITE = \{\langle M \rangle \mid L(M) \text{ is infinite}\}$$

Hint: This is very similar to the L_{VASSAR} proof in the reductions handout.

For that proof, we made a decider for A_{TM} by constructing a special TM, M_w, whose language is Σ^* when M accepts w and $\varnothing$ when M doesn't. Since Σ^* contains `Vassar` but $\varnothing$ doesn't, D's answer to "is `Vassar` $\in L(M_w)$?" tells us whether M accepts w.

For this exercise, the same M_w works! Σ^* is infinite and $\varnothing$ is finite, so D's answer to "is $L(M_w)$ infinite?" tells us whether M accepts w.