Assignments due: 28 April, 1:30 p.m.
Corrections due: 30 April, 1:30 p.m.

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

Exercise 1

We’ve introduced several terms and definitions related to Turing machines, languages, and what it means to solve a problem. Some of these terms can only describe Turing machines, while others can only describe languages. A statement that uses terms incorrectly has no meaning; it cannot be evaluated as being true or false.

Consider, by analogy, the statement “the set \( \mathbb{N} \) is even”. This statement doesn’t type-check because “even” only applies to individual numbers, and \( \mathbb{N} \) isn’t a number. Similarly, the statement “\( 1 \in 5 \)” doesn’t type-check because 5 isn’t a set.

On the other hand, the statement \( \mathbb{Z} \subseteq \mathbb{N} \) type-checks, but it isn’t true – the integers aren’t a subset of the natural numbers.\(^1\)

For each of the following statements, decide whether

- the statement type-checks and is true,
- the statement type-checks and is false, or
- the statement does not type-check.

Briefly explain your answers.

a. If \( M \) is a Turing machine, \( w \) is a string, and \( M \) accepts \( w \), then \( A_{\text{TM}} \) accepts \( (M, w) \).

\(^1\) This is the mathematical equivalent of a grammatically correct sentence that’s false, like “Otters aren’t cute”. You understand what the sentence means, but it’s objectively not true. They’re adorable.
b. If $M$ is a Turing machine, $w$ is a string, and $M$ loops on $w$, then $\langle M, w \rangle \notin L(U)$.

c. $U$ is decidable.

d. $\langle U \rangle$ is decidable.

e. $\{\langle U \rangle\}$ is decidable.
For Exercises 2 and 3, follow the style of proof by construction seen in the textbook in sections 4.1 and 4.2. In particular, you may want to review the solutions to problems 4.10, 4.12, and 4.14 on pages 213–4.

Exercise 2

Prove that the collection of decidable languages (R) is closed under the intersection operation.
Exercise 3

Prove the following language is decidable:

\[ L = \{ \langle M, n \rangle \mid M \text{ is a Turing machine with } n \text{ states, and there exists a string } w \in \Sigma^* \text{ of length at most } n^2 \text{ such that } M \text{ accepts } w \text{ in at most } n^3 \text{ transitions} \} \]

The choices of \( n^2 \) as the length limit and \( n^3 \) as the number of transitions are arbitrary. Your proof should require almost no modification if I'd chosen \( n \) or \( 2n \) or \( 100^n \).
Exercise 4

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.\(^2\) 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 CHECKER:

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

We’ll prove the language 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 CHECKER is decidable. This means that we could write some 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:

File c.py:

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

def main(my_input):
    my_source = open("c.py").read()
    if is_password_checker(my_source):
        # Reject the password my_input
        return False
    else:
        # Accept the password my_input
        return True
```

\(^2\) 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! :-(/