CMPU 240 · Spring 2026

Assignment 9

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

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]

[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.*

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_{TM} accepts $\langle M, w \rangle$.

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.

Exercise 2

Below are examples of Python functions that act as recognizers or deciders.[2] For each of the functions,

- What is the language the function recognizes? Use set-builder notation if appropriate.
- Is the function a decider or just a recognizer? Why?

For this exercise, let $\Sigma = \{\texttt{a}, \texttt{b}\}$, meaning that you only need to consider input strings made of `a`s and `b`s.

[2] *If you haven't used Python before, don't panic! You can basically treat them as simplified Java, but feel free to ask questions about any parts you don't understand.*

a. Answer the questions above about this function:

```
def diomira(w: str) -> bool:
    length = len(w)

    if length % 2 == 1:
        return False

    # Indices within strings range from 0 (the first
    # character) to length - 1 (the last character).
    for i in range(0, length / 2):
        if (w[i] != "a") or (w[length - 1 - i] != "b"):
            return False

    return True
```

`%` is the modulus operator, which gives the remainder after dividing the first number by the second.

b. Answer the questions above about this function:

```
def isidora(w: str) -> bool:
    strings = [""]

    while True:
        next_strings = []

        for x in strings:
            if w == x:
                return True

            next_strings.append(x + "a")
            next_strings.append(x + "b")

        # On the next iteration through the `while` loop,
        # use next_strings for the `for` loop.
        strings = next_strings
```

c. Answer the questions above about this function:

```
def dorothea(w: str) -> bool:
    x = 0
    y = 1
    while len(w) != x:
        z = x + y
        x = y
        y = z
    return True
```

Exercise 3

This exercise explores the following fundamental theorem about the relationship between the **R** and **RE** languages:

> If L is a language, then $L \in \mathbf{R}$ if and only if $L \in \mathbf{RE}$ and $\overline{L} \in \mathbf{RE}$.

This theorem has a beautiful intuition: It says that a language L is decidable ($L \in \mathbf{R}$) precisely if, for every string in the language, it's possible to *prove* it's in the language ($L \in \mathbf{RE}$) and, simultaneously, for every string *not* in the language, it's possible to *prove* it's not in the language ($\overline{L} \in \mathbf{RE}$). For this exercise, we'll look at one of the two directions of this theorem.

Let L be a language where $L \in \mathbf{RE}$ and $\overline{L} \in \mathbf{RE}$. This means that there's a recognizer for L and a recognizer for $\overline{L}$. In software, you could imagine that these correspond to functions `check_in_L` and `check_in_L_bar` with the following properties:

- For $w \in \Sigma^*$, $w \in L$ if and only if `check_in_L(w)` returns `True`.
- For $w \in \Sigma^*$, $w \in \overline{L}$ if and only if `check_in_L_bar(w)` returns `True`.

Now we can show that $L \in \mathbf{R}$ by writing pseudocode for a function `is_in_L` that takes as input a string w, then returns `True` if $w \in L$ and returns `False` if $w \notin L$:

```
def is_in_L(w: str) -> bool:
    if check_in_L(w):
        return True
    if check_in_L_bar(w):
        return False
```

This has the right idea, but it won't work! Explain what's wrong and correct the pseudocode. (It's just pseudocode, so you can make up appropriate notation as long as it's clear what you mean.)