Hajin Park A18596632
CSE 105 Winter 2025
Project Task 2
March 17, 2025

1 Formal Definitions

Definition of A_{TM}

The language A_{TM} is defined as:

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a Turing machine and } M \text{ accepts } w\}.$$

It is well-known that A_{TM} is undecidable.

Definition of EQ_{TM}

The language EQ_{TM} is defined as:

$$EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1, M_2 \text{ are Turing machines and } L(M_1) = L(M_2)\}.$$

Mapping Reduction from A_{TM} to EQ_{TM}

We define a mapping reduction f from A_{TM} to EQ_{TM} as follows:

- Given an instance $\langle M, w\rangle \in A_{TM}$, the function f outputs a pair $\langle M_1, M_2\rangle$ of Turing machines where:
  - M_2 is a fixed Turing machine that accepts every input, $L(M_2) = \Sigma^*$.
  - M_1 is constructed such that on input x:
    * If $x \neq$ ”0”, then M_1 immediately accepts.
    * If $x =$ ”0”, then M_1 simulates M on the input w.
- Therefore, if M accepts w, then M_1 accepts every string (i.e., $L(M_1) = \Sigma^*$); otherwise, M_1 rejects 0 while accepting all other inputs, implying $L(M_1) \neq \Sigma^*$. Since M_2 always accepts, we have:
$$\langle M, w\rangle \in A_{TM} \iff f(\langle M, w\rangle) = \langle M_1, M_2\rangle \in EQ_{TM}.$$

2 Implementation Overview

The Python implementation consists of the following components:

1. **Input Parsing:** The function `parse_instance` processes the input string to extract the Turing machine description and the input string w. The input is expected to be in the following format:

```
TM:
<description of Turing machine M>
Input:
<input string w>
```

2. **Construction of M_1:** The function `construct_M1` builds a Turing machine that accepts any input $x \neq$ "0" immediately and, for $x =$ "0", simulates the provided Turing machine M on w.
3. **Construction of M_2:** The function `construct_M2` returns a fixed Turing machine that unconditionally accepts every input.
4. **Reduction Function:** The function `reduce_ATM_to_EQTM` integrates the above components to generate the output in the format:

```
M1:
<encoding of M1>
---
M2:
<encoding of M2>
```

Code

```
"""
Project: CSE105W25 Task 2
Author: Hajin Park
Date: 3/16/2025
"""


def parse_instance(instance_str):
    """
    Parses the instance string to extract the Turing machine (TM) description
        and input string w.

    The expected input format is:
        TM:
        <description of Turing machine M>
        Input:
        <input string w>

    Parameters:
        instance_str (str): The complete input string.

    Returns:
        tuple: (M_description (str), w (str))
    """
    tm_lines = []
    input_lines = []
    reading_tm = False
    reading_input = False
```

```
    for line in instance_str.strip().splitlines():
        line = line.strip()
        if line.startswith("TM:"):
            reading_tm = True
            reading_input = False
            continue
        elif line.startswith("Input:"):
            reading_input = True
            reading_tm = False
            continue

        if reading_tm:
            tm_lines.append(line)
        elif reading_input:
            input_lines.append(line)

    M_description = "\n".join(tm_lines).strip()
    w = "\n".join(input_lines).strip()
    return M_description, w


def construct_M1(M_description, w):
    """
    Constructs the encoding for M1.

    M1 is defined to:
      - Accept any input x that is not "0" immediately.
      - For x equal to "0", simulate the provided Turing machine M on input w.

    Parameters:
        M_description (str): A high-level description of Turing machine M.
        w (str): The input string for simulation.

    Returns:
        str: The string encoding of M1.
    """
    M1 = (
        "TM:\n"
        "Description:␣This␣Turing␣machine␣M1␣works␣as␣follows:\n"
        "␣␣-␣If␣input␣x␣!=␣'0',␣accept␣immediately.\n"
        "␣␣-␣If␣input␣x␣==␣'0',␣simulate␣the␣following␣Turing␣machine␣M␣on␣
            input␣w.\n"
        "M_simulation:\n" + M_description + "\n"
        "Input␣for␣simulation:␣" + w + "\n"
    )
    return M1


def construct_M2():
    """
    Constructs the encoding for M2.

    M2 is defined as a Turing machine that unconditionally accepts any input.

    Returns:
        str: The string encoding of M2.
```

```
    """
    M2 = (
        "TM:\n"
        "Description: This Turing machine M2 accepts every input 
           unconditionally.\n"
        "States: A, B\n"
        "Alphabet: 0,1\n"
        "Tape_Alphabet: 0,1,_\n"
        "Start: A\n"
        "Accept: B\n"
        "Reject: B\n"
        "Transitions:\n"
        "A,0 -> B,0,R\n"
        "A,1 -> B,1,R\n"
        "A,_ -> B,_,R\n"
        "B,0 -> B,0,R\n"
        "B,1 -> B,1,R\n"
        "B,_ -> B,_,R\n"
    )
    return M2


def reduce_ATM_to_EQTM(instance_str):
    """
    Implements the mapping reduction from A_TM to EQ_TM.

    Parameters:
        instance_str (str): A string representing an instance of A_TM,
           formatted with a 'TM:' section
                            for the Turing machine description and an 'Input:'
                               section for the input w.

    Returns:
        str: A string representing the pair <M1, M2> in the format:
             M1:
             <encoding of M1>
             ---
             M2:
             <encoding of M2>
    """
    M_description, w = parse_instance(instance_str)
    M1_description = construct_M1(M_description, w)
    M2_description = construct_M2()
    output = "M1:\n" + M1_description + "\n---\nM2:\n" + M2_description
    return output
```

3 Demonstration Tests

The function `reduce_ATM_to_EQTM` is demonstrated using two test instances:

Example 1 (Positive Instance)

- **Input:** A Turing machine M_{pos} that always accepts. The input string is arbitrary ("any").

- **Expected Outcome:** M_1 is constructed such that it accepts every input (since M_{pos} accepts), yielding $L(M_1) = \Sigma^*$. Because M_2 also accepts every input, we have $L(M_1) = L(M_2)$.

Example 2 (Negative Instance)

- **Input:** A Turing machine M_{neg} that always rejects. The input string is arbitrary ("any").
- **Expected Outcome:** M_1 is constructed so that it rejects the input 0 (because M_{neg} rejects) while accepting all other inputs. Thus, $L(M_1) \neq \Sigma^*$ and $L(M_1) \neq L(M_2)$.

Code

```
import pytest
from main import reduce_ATM_to_EQTM

# Positive instance: a Turing machine M_pos that always accepts.
positive_instance = """\
TM:
Description: This Turing machine M_pos always accepts.
States: q0,q_accept
Alphabet: 0,1
Tape_Alphabet: 0,1,_
Start: q0
Accept: q_accept
Reject: q_reject
Transitions:
q0,0 -> q_accept,0,R
q0,1 -> q_accept,1,R
q0,_ -> q_accept,_,R
q_accept,0 -> q_accept,0,R
q_accept,1 -> q_accept,1,R
q_accept,_ -> q_accept,_,R

Input:
any
"""

# Negative instance: a Turing machine M_neg that always rejects.
negative_instance = """\
TM:
Description: This Turing machine M_neg always rejects.
States: p0,p_reject
Alphabet: 0,1
Tape_Alphabet: 0,1,_
Start: p0
Accept: p_accept
Reject: p_reject
Transitions:
p0,0 -> p_reject,0,R
p0,1 -> p_reject,1,R
p0,_ -> p_reject,_,R

Input:
any
"""
```

```
def test_reduce_ATM_to_EQTM_positive():
    output = reduce_ATM_to_EQTM(positive_instance)
    assert "M1:" in output, "Output should include 'M1:' marker."
    assert "---" in output, "Output should include a separator '---'."
    assert "M2:" in output, "Output should include 'M2:' marker."
    assert (
        "always accepts" in output
    ), "Positive instance should mention 'always accepts'."


def test_reduce_ATM_to_EQTM_negative():
    output = reduce_ATM_to_EQTM(negative_instance)
    assert "M1:" in output, "Output should include 'M1:' marker."
    assert "---" in output, "Output should include a separator '---'."
    assert "M2:" in output, "Output should include 'M2:' marker."
    assert (
        "always rejects" in output
    ), "Negative instance should mention 'always rejects'."
```