COMS 3261 Fall 2023 Review Handout: Turing Machines

Yihan Shen
Credit to Fall 2022 TA Andrew Jin and Anastasija Tortevska

1 Definition

A Turing Machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$:

1. Q is a finite set of states.
2. Σ is the (finite) input alphabet not containing the blank symbol.
3. Γ is the (finite) tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subset \Gamma$.
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function.
5. $q_0 \in Q$ is the start state.
6. $q_{\text{accept}} \in Q$ is the accept state.
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

For a Turing Machine M:

- M receives its input $w = w_1 w_2 ... w_n \in \Sigma^*$ on the left-most n squares of the tape, leaving the rest of the tape blank. The start configuration on input w is q_0, where the reading head is pointing to the first (leftmost) square of the tape, and the state is q_0. Then at each step, the transition δ is applied.
- First blank symbol marks the end of input (initially, afterwards M could write blank symbols anywhere).
- If M ever tries to move its head to the left of the leftmost square of the tape, it stays in place.
- It may sometimes be helpful to have an indicator symbol like #, to indicate the leftmost position of the tape, but this is not automatically there (so if this is not defined as a requirement for a valid input, the TM could be designed to start by inserting it).

- If at any point δ takes M to q_{accept} or q_{reject}, then M halts, and we say it accepted/rejected the string. Otherwise, if δ keeps being applied (algorithm keeps running) without ever accepting or rejecting, then we say this machine runs forever (or is in an infinite loop). A TM that always halts on every input is called a decider.
- On any input, there are three possible behaviors of a TM M: M will accept, reject, or run forever on this input. The language recognized by a Turing Machine M is the set of strings that M accepts.

2 Implementation-Level Example: Using Turing Machine to Compute Functions

Example 1

Let $\Sigma = \{\#, a, b, c\}$. We aim to provide an implementation level description of an input-output Turing Machine (TM) that computes the following function:

$$f(x) = \#x$$

Solution 1

Idea: Essentially, we need to insert a # key in the beginning and move every character in x one position to the right on the tape. Therefore, we can use the states of the Turing Machine to remember which symbol we need to add to move.

Implementation:

States:

- q_{start}: The start state.
- q_{a}: State for remembering to add symbol "a" next
- q_{b}: State for remembering to add symbol "b" next
- q_{c}: State for remembering to add symbol "b" next
- q_{halt}: Halting(accepting) state

Transition Function (δ):

1. From q_{start}, look at the first symbol in the input string, transition to the next state representing that symbol, write a # on the tape, and move right.
2. In q_{a}, remember the symbol that the head is currently pointing to, transition to that corresponding state, and write a symbol "a", and move the head to the right. Same thing for q_{b},q_{c}

3. In any state, if current symbol encountered is blank, it means that we've reached the end of the string, then write the symbol corresponding to current state, and finish by transitioning to the halting state.

Diagram

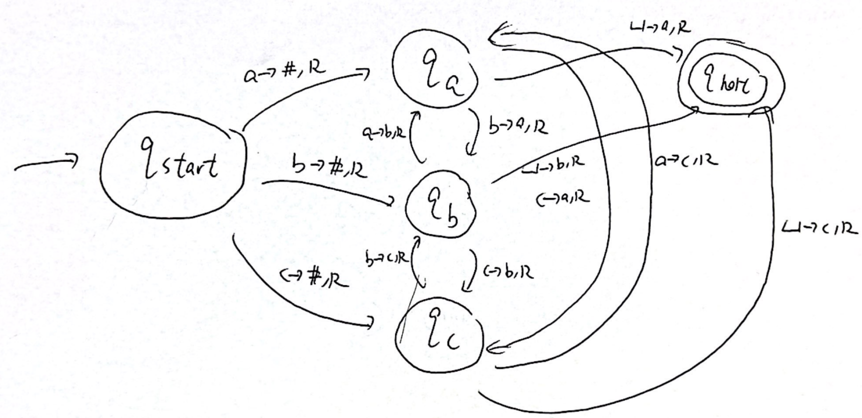

Diagram for Ex 1

Example 2

Let $\Sigma = \{\#, 0, 1\}$. We aim to provide an implementation level description of an input-output Turing Machine (TM) that computes the following function:

$$f(\#\langle a\rangle\#\langle b\rangle) = f(\#\langle a+b\rangle)$$

where ⟨x⟩ stands for the binary representation of the number x.

Solution 2

Idea We have two ways of achieving this: we can either use a(or b) as a counter and decrement it by 1 every time while increasing b(or a) by 1. Alternatively, we can also implement a full-adding Turing machine, which would perform addition as we normally would on paper. The description given here uses the first approach. Please observe that we still need to deal with the case of using extra tape cells.

Implementation

States:

- q_{start}: The start state.
- $q_{\text{a\_scan}}$: State for scanning right on string a to find the end of the first number.
- $q_{\text{a\_subtraction}}$: State for performing the subtraction on a.
- $q_{\text{a\_move right}}$: State for moving the head after the subtraction on a.
- $q_{\text{b\_scan}}$: State for scanning right on string b to find the end of the second number
- $q_{\text{b\_addition}}$: State for performing the addition on string b
- $q_{\text{b\_move left}}$: State for moving the head after the addition on b.
- q_{cleanup}: State for cleaning up the tape and preparing the output.
- q_{accept}: Accept state.

Transition Function (δ):

1. From q_{start}, expect the # symbol and transition to $q_{\text{a\_scan}}$, move to right position
2. In $q_{\text{a\_scan}}$, move right until you encounter a # symbol (representing the end of the first number). Transition to $q_{\text{a\_subtraction}}$ and move left.
3. In $q_{\text{a\_subtraction}}$, perform the subtraction by the following. If the next digit is 0, change it to 1 and continue moving left. If it's 1 change it to 0 and stop carrying. Transition to $q_{\text{a\_move right}}$ and move right
4. In $q_{\text{a\_move right}}$, move right until you see a # symbol, indicating that we've reached the end of a. Then, transition to $q_{\text{b\_scan}}$ and move right.
5. In $q_{\text{b\_scan}}$, similar to that of a, move right until you see a blank symbol, indicating that we've reached the end of b, at which point it moves left and transitions to $q_{\text{b\_addition}}$
6. The logic in $q_{\text{b\_addition}}$ is similar to that of a, except that it's reversed. It continues moving left and carrying over as long as it sees consecutive 1s. On the first 0, it changes 0 to 1 and transitions to $q_{\text{b\_move left}}$.
7. In $q_{\text{b\_move left}}$, it moves left with all symbols until seeing a #, which indicates that it has reached the end of number a. At that point, it repeats by transitioning to $q_{\text{a\_subtraction}}$ and moves left.

8. We need the Turing Machine to finish the computation when the first number has become 0. Therefore, when we are in state $q_{\text{a\_subtraction}}$, if we've reached the # symbol without seeing any 1s, we can transition to q_{cleanup}.
9. In q_{cleanup},we keep removing 1s to blanks until we reach #, at which point we erase it and move to q_{accept}

Diagram

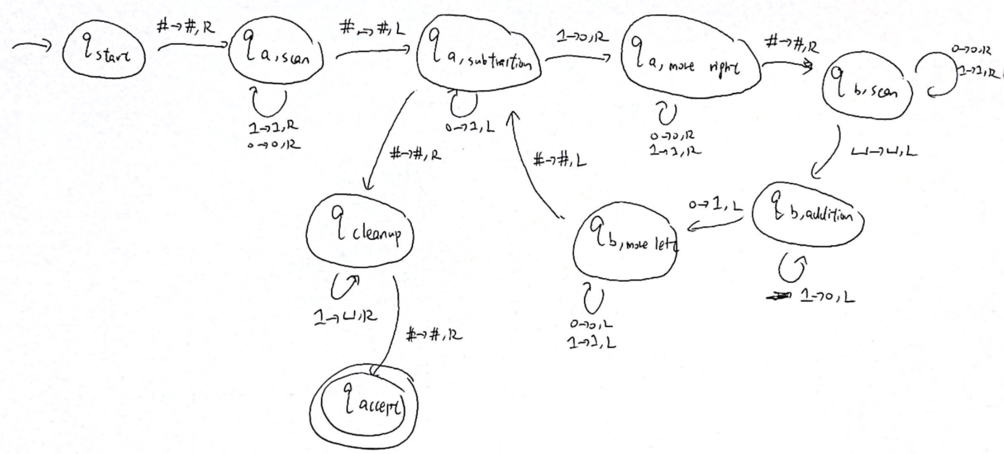

Diagram For Ex 2.

3 High-Level Example

First, recall from the definitions that:

A language is Turing-recognizable ⇔ there exists a TM that accepts strings in that language and doesn't accept strings that aren't in that language.

- $w \in L \Rightarrow M$ accepts w
- $w \notin L \Rightarrow M$ rejects or runs forever on w

A language is Turing-decidable ⇔ there exists a TM that accepts strings in that language and rejects strings that aren't in that language. A decider halts on every input.

- $w \in L \Rightarrow M$ accepts w

- $w \notin L \Rightarrow M$ rejects w

Remark. If M is a decider it will always halt, but if M is a recognizer it may not halt.

3.1 Examples of Turing Decideable Languages

- ADFA $= \{\langle D, w\rangle | D$ is a DFA and D accepts $w\}$
- ANFA $= \{\langle N, w\rangle | N$ is an NFA and N accepts $w\}$
- EDFA $= \{\langle D\rangle | D$ is a DFA and $L(D) = \emptyset\}$
- EQDFA $= \{\langle D1, D2\rangle | D1$ and $D2$ are DFAs and $L(D1) = L(D2)\}$

4 Closure Properties of Turing Recognizeable and Decidable Languages

Theorem 1 (Closure Properties of Decidable Languages). Decidable languages are closed under the following:

- union
- intersection
- concatenation
- complement
- Kleene star

Proof. For union: Suppose M_1 and M_2 are deciders. We will create M to decide their union as follows:

M on input x:

1. Run M_1 on x. If M_1 accepts, accept.
2. Run M_2 on x. If M_2 accepts, accept.
3. Reject.

Observe that M will always halt (either reach an accept state or reject state, not run forever). This is because step 1 will always halt (since M_1 is a decider) and step 2 will always halt (since M_2 is a decider). Hence, M is also a decider. Moreover, M accepts $x \Leftrightarrow M_1$ accepts x or M_2 accepts $x \Leftrightarrow x \in L_1 \cup L_2$ (since M_1 is a decider for L_1 and M_2 is a decider for L_2).

For concatenation: Suppose M_1 and M_2 are deciders. Let L_1 and L_2 be their respective languages. We will create M to decide their concatenation as follows:

M on input x:

1. For every way to split x into $x = yz$:
   (a) Run M_1 on input y.
   (b) Run M_2 on input z.
   (c) If both M_1 and M_2 accept, let M accept.
2. Reject.

(Proof that this is a decider for concatenation left as an exercise).

Note: a different way to prove closure under concatenation is to construct a non-deterministic TM, which starts by non-deterministically splitting x into y and z. Complement was covered in class and intersection is left as an exercise for the reader.

Theorem 2 (Closure Properties of Recognizable Languages). Recognizable languages are closed under the following:

- union
- intersection
- concatenation
- Kleene star

In particular, we note that recognizable languages are **NOT** closed under the following:

- complement (will be shown later in the class)

Proof. For union: Suppose M_1 and M_2 are TMs, but not necessarily deciders. What happens when we attempt to repeat the proof above?

M on input x:

1. Run M_1 on x.
2. Run M_2 on x.
3. If either M_1 or M_2 accepts, let M accept, else reject.

A problem occurs if M_1 runs forever on x. Then, even if M_2 accepts x (and hence, M should accept x), we never get to that point because we're running forever with M_1. One solution to this is using an NTM (non-deterministic Turing machine) N, which allows us to run both M_1 and M_2 simultaneously on two different branches.

N on input x:

1. Non-deterministically choose either M_1 or M_2.
2. Run the chosen TM M_i on x. If it accepts, accept.

Recall that with an NTM, an input x is in the language if any branch of the computation tree accepts (just like an NFA). Thus, the machine N defined above will accept if and only if either M_1 or M_2 accepts, which happens if and only if $x \in L_1$ or $x \in L_2$, which is if and only if $x \in L_1 \cup L_2$, as we wanted.

An alternative solution using a deterministic TM M can also be designed. We can't run M_1 first then M_2 (or vice versa) because there's a chance one machine runs forever on an input x, which prevents us from attempting to run the other machine. However, to circumvent this, we can run the two machines in parallel using a two-tape machine (simulate M_1 using one tape, and simulate M_2 using the other tape).

5 More Practice Problems

5.1 Problem 1

Consider the input-output Turing Machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{halt}})$ where $Q = \{q_0, q_1, q_{\text{halt}}\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \sqcup\}$, and δ is given by:

$$\begin{aligned}
\delta(q_0, 0) &= (q_0, 0, R), \\
\delta(q_0, 1) &= (q_0, 1, R), \\
\delta(q_0, \sqcup) &= (q_1, \sqcup, L), \\
\delta(q_1, 0) &= (q_{\text{halt}}, 1, R), \\
\delta(q_1, 1) &= (q_1, 0, L), \\
\delta(q_1, \sqcup) &= (q_{\text{halt}}, \sqcup, L).
\end{aligned}$$

(a) Provide the complete sequence of configurations of M when run on input 100. What is the output of M on this input?

(b) What is the output of M on 10011? On input 11?

(c) What function is computed by M?

5.2 Problem 2

Let $\Sigma = \{\#, a, b\}$. We aim to provide an implementation level description of an input-output Turing Machine (TM) that computes the following function:

$$f(\#\langle x\rangle) = \begin{cases} \#\langle \frac{x}{2} \rangle & \text{if } x \text{ is even,} \\ \#\langle 3x+1 \rangle & \text{otherwise.} \end{cases}$$

where ⟨x⟩ stands for the binary representation of the number x.

5.3 Problem 3

Show that ECFG $= \{\langle G\rangle | G \text{ is a CFG and } L(G) = \emptyset\}$ is decidable.

5.4 Problem 4

Let $L = \{\langle M, k\rangle | M$ is a TM, k is a positive integer, and there exists an input to M that makes M run for at least k steps$\}$. Prove that L is decidable.