Handout 8a: Turing Machines Review

Andrew Jin and Anastasija Tortevska

COMS 3261 Fall 2022

1 Turing Machines

1.1 High-level overview

Intuition: Recall that a PDA is essentially a DFA/NFA with a stack. Adding this stack increases your power. For example,

$$L = \{w \in \Sigma^* : w = 0^n 1^n, n \in \mathbb{N}\}$$

is context-free because you can use the stack to keep track of the number of 0's and 1's.

A Turing machine takes this idea further. You can think of a Turing machine as a DFA/NFA with a tape that contains your string. There are three main differences:

- You can now move along the tape in any direction, which allows you to read any character from the string, however many times you want (you're no longer limited to reading through the string one character at a time until you consume the string)
- You can write onto the tape, which allows you to record and use information that you otherwise wouldn't be able to.
- When using DFAs we generally have a fixed number of possible states and configurations that limits the possible actions we would like to do, on the other hand Turing Machines do not have the same limitation – allowing us to have access to as much space and memory needed (infinite tape) and various actions that can be performed by reading and writing from the tape.

For example,

$$L = \{w \in \Sigma^* : w = 0^n 1^n 2^n, n \in \mathbb{N}\}$$

is not context-free, but is Turing-recognizable because you can write onto the tape to keep track of the number of 0's, 1's, and 2's you see (implementation left as an exercise).

Remark. Note that when we will use high-level description, we have to make sure that each step we write is algorithmically implementable.

The Church-Turing thesis says that Turing Machines capture what computation, or an algorithm, is: any reasonable model of computation can be translated into an equivalent computation with a Turing machine. We will thus identify "algorithm" with a TM.

1.2 TM definition

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

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

For a Turing Machine M:

- M receives it's input $w = w_1w_2...w_n \in \Sigma^*$ 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 TM is the set of strings that are accepted by the TM.

1.3 Examples of Turing-Decidable Languages

- The set of all palindromes: $L = \{w \in \Sigma^* : w = w^R\}$.
  Idea: You can move along the tape to read from both ends. Compare the first and last letter while also marking them, then the second and second-to-last letter, and so on.

- The set of strings which are composed to two identical halves: $L = \{ww : w \in \Sigma^*\}$.
  Idea: You can first identify the middle point by moving back and forth from one end to another and marking a symbol on each end. Then, you can move along the tape to read both halves. Compare the first letter of each half, then compare the second letter of each half, and so on (marking where you are).

- The set of all integer square matrices with determinant zero: $L = \{A \in \mathbb{Z}^{n\times n} : \det(A) = 0\}\}$.
  Idea: By the Church-Turing thesis, a Turing machine can execute virtually any algorithm you want it to. Since computing a determinant is just an algorithm, you can use a Turing machine to check whether a matrix has determinant zero.

- The set of all (encodings of) DFAs that accept a particular string x: $A_{DFA} = \{\langle D, x\rangle : D \text{ is a DFA that accepts } x\}$.
  Idea: can check whether $\langle D, x\rangle$ should be accepted or not by running the DFA D with x as an input, and seeing if D accepts x or not (and let your Turing machine M accept $\langle D, x\rangle$ if D accepts x, and let M reject $\langle D, x\rangle$ if D rejects x).

Remark. In the final example, the input to the Turing machine isn't just the DFA D or the string x, but rather the combination of both of them. Since the Turing machine has access to both D and x as inputs, it can use that information to simulate running D on input x.

2 Recognizable and Decidable

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

$$w \in L \implies M \text{ accepts } w$$

$$w \notin L \implies M \text{ rejects or runs forever on } w$$

A language is **Turing-decidable** $\iff$ 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.

$$w \in L \implies M \text{ accepts } w$$

$$w \notin L \implies M \text{ rejects } w$$

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

How to prove that a TM recognizes or decides some language ?

- To prove that a TM M recognizes a language L you need to show that M accepts x if and only if $x \in L$.

  First way: directly show it in a single statement: $x \in L \iff \dots \iff M$ accepts x.

  Second way: break it into two. First prove $x \in L \implies M$ accepts x, and then prove $x \notin L \implies M$ either rejects x or runs forever on x.

- To prove that a TM M decides a language L, you need to show the same as above, and additionally that M halts on all input. There are a few equivalent ways to show this, for example:

  First way: Argue that M is a decider (halts on all inputs – every step is finite), and then prove $x \in L \iff M$ accepts x. Or equivalently:

  Second way: prove that $x \in L \implies M$ accepts x, and $x \notin L \implies M$ rejects x.

How to prove that a langauge is decidable or recognizable? The most straightforward way, is to construct a TM (make sure every step is indeed implementable!) and then prove that it recognizes or decides the language, as above. If you are constructing a recognizer, it's ok to have infinite loops, as long as the execution can run forever only when the input is not in the language. If you are constructing a decider then it must always halt, so if it has a loop that looks like it could run forever, you should make sure that on every input, the loop will eventually complete (within finitely many steps).

When the language consists of strings that are themselves encodings of some model of computation (eg DFA or TM), some common techniques for constructing a TM recognizing/deciding the language are the following: (a) have a loop that checks all strings, or all strings of a certain format (e.g, we used that to construct a recongizer for NE_{TM}) (b) inspect and/or manipulate the given input: its states, transitions, etc – check some properties of it (e.g, we used that to construct a decider for E_{DFA}) (c) use algorithms we have given in class in the past, and rely on things we already proved (e.g, we did that when constructing a decider for A_{NFA}). You will see more examples in this handout, and later in the class.

Another way to prove that a language L is decidable, is to separately prove that both L and $\overline{L}$ are recognizable (since we proved that a language is decidable if and only if both the language and its complement are recognizable).

2.1 Closure Properties

Theorem 1 (Closure Properties of Decideable 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 inpt 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 $x \iff M_1$ accepts x or M_2 accepts $x \iff x \in L_1 \cup L_2$ (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 inpt 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 spliting x into yz.

Complement was covered in class and intersection is left as 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 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 we attempt to repeat the proof above?

on inpt 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 inpt 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).

M on inpt x:

1. Write input x onto both tapes.
2. Repeat
    (a) Run M_1 for one step (unless it had already rejected). If M_1 reaches an accept state, accept.

   (b) Run M_2 for one step (unless it had already rejected). If M_2 reaches an accept state, accept.
   (c) If both M_1 and M_2 have already reached a reject state, reject.

We note that for high level description, step 2 is written as "In parallel, run M_1 on x and M_2 on x. If either of them accepts, accept. If both of them reject, reject".

Let's just consider some cases to see how this algorithm works.

1. Case 1: M_1 and M_2 both halt (either accept or reject). Then, M also halts.
2. Case 2: M_1 runs forever but M_2 accepts. Since we run both machines one step at a time, we will eventually reach the point where M_2 accepts (because this must occur in a finite number of steps). Therefore, M will accept, which is what we want.
3. Case 3: M_1 runs forever but M_2 rejects. Like with case 2, we will eventually reach the point where M_2 rejects. However, we will continue running M_1 infinitely inside the loop, so M will run forever (we only have M reject if both M_1 and M_2 reject, but this doesn't happen). This is fine though, because all we need is for M to not accept, and running forever is fine with a recognizer.
4. Case 4: M_1 and M_2 both run forever. Then, we will be stuck inside an infinite loop and M will run forever. This is fine because we just need M to not accept.

Hence, we can see that M will accept if and only if either M_1 or M_2 accepts, and M will either reject or run forever if both M_1 and M_2 either reject or run forever. Thus M recognizes the union of M_1 and M_2.

For intersection, concatenation: Left as exercise. Hint: For concatenation, you can use something similar to the proof we did for union. $\square$

2.2 More Examples of Turing-Decidable Languages

- $A_{DFA} = \{\langle D, w\rangle | D \text{ is a DFA and } D \text{ accepts } w\}$
- $A_{NFA} = \{\langle N, w\rangle | N \text{ is a NFA and } N \text{ accepts } w\}$
- $E_{DFA} = \{\langle D\rangle | D \text{ is a DFA and } L(D) = \phi\}$

All three languages have been shown in class.

- $EQ_{DFA} = \{\langle D_1, D_2\rangle | D_1 \text{ and } D_2 \text{ are DFAs and } L(D_1) = L(D_2)\}$

We prove that EQ_{DFA} is TM-decidable.

Proof. Let R be a decider for E_{DFA} (which we have shown in class exists). Construct the following TM:

TM M = "On input $\langle D_1, D_2 \rangle$,

1. Check whether $\langle D_1, D_2 \rangle$ is a valid input, otherwise *reject*
2. Construct DFA $\langle D_3 \rangle$ such that $L(D_3) = (L(D_1) \setminus L(D_2)) \cup (L(D_2) \setminus L(D_1))$
3. Run R on $\langle D_3 \rangle$
4. If R accepts, *accept*; otherwise *reject*"

Note that We can implement step 2 in our algorithm since we've seen in class algorithms for computing DFAs for complement, intersection, and union of the languages of given DFAs, and these algorithms can be combined to get step 2 above.

We can see that M is a decider because R is a decider, and the other steps always halt.
$x \in EQ_{DFA} \iff x = \langle D_1, D_2 \rangle$ where D_1, D_2 are DFAs and $L(D_1) = L(D_2)$ $\iff (L(D_1) \setminus L(D_2)) \cup (L(D_2) \setminus L(D_1)) = \emptyset \iff \langle D_3 \rangle$ constructed in step 2 encodes a DFA D_3 that satisfies $L(D_3) = \emptyset \iff R(\langle D_3 \rangle)$ accepts $\iff M$ accepts.

$\square$

3 Variants

The following Variations of a Turing machine are equivalent in power to a standard Turing machine. Hence, if it's easier to use these definitions for a particular problem, it's implied that there exists an equivalent standard TM.

Note that this is true also for many other models, including for example your favorite programming language, which can be simulated by a TM. However, the variants below can be proven to be equivalent to a TM much more directly (we did not see full proofs in class for this equivalence since there was not enough time, but it would be doable – you can try to prove it yourself or check the textbook if you'd like to see a full proof of equivalence).

- Two-sided Tape Turing Machines (Tape is infinite in both directions)
- Multi-Tape Machines (For a FINITE fixed number k, the Turing machine has k tapes and k heads reading each tape).
- Left/Right/Stay Turing Machines (Turing machine, in addition to moving left and right, can choose to stay put in the same position).
- Non-Deterministic Turing Machine (multiple possible transitions and configurations yielded at configurations) - akin to the relation of NFAs to DFAs
- 2-Dimensional Turing Machine (Instead of a 1D-tape, the TM has a 2D-grid)

3.1 Enumerators

An **enumerator** is a Turing Machine with an attached printer that it can use as an output device to print strings. Every time the Turing machines wants to add a string to the list, it sends the string to the printer. Another term for "recognizable" is "recursively enumerable" - the set of languages recognized by enumerators (and another term for "decidable" is "recursive").

- An enumerator starts with a blank input tape
- If the enumerator does not halt it may print an infinite list of strings
- The language recognized by the enumerator is the collection of strings that it eventually prints out.
- Note: an enumerator may generate the strings of the language it recognizes in any order, possibly with repetitions.
- If L is recognizable and accepts string w, then it will eventually appear on the list printed by L's enumerator, E.

Theorem: A language is Turing-recognizable if and only if some enumerator enumerates it. (Sipser, theorem 3.21).

(Try to prove it yourself! you can also prove that L is Turing-decidable if and only if some enumerator enumerates it in length-increasing order)

4 Practice Problems

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

   (a) Provide the complete sequence of configurations of M when ran 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?

2. Let $\Sigma = \{\#, 0, 1\}$. Provide an *implementation level* description of a input-output TM that computes the function

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

   where $\langle x\rangle$ stands for the binary representation of the number x.

   (For example, if the TM starts with #100 on the tape it should halt with #10 on the tape; if it starts with #11, it should halt with #1010.)

   You may use a TM with more than one tape – in this case the output should be written on the first tape.

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

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 that L is decidable.