COL352 HW-4

Sarthak Agrawal	Piyush Gupta	Navneel Singhal
2018CS10383	2018CS10365	2018CS10360

April 25, 2021

Read the instructions carefully.

How to describe Turing Machines: Describing a Turing Machine can often be tricky: it's hard to strike the right balance between rigor and understandability. Here are a few ways Turing Machines can be described (inspired from Section 3.3 of Sipser's book).

1. **Formal definition**: Define all the components $(Q, \Sigma, \Gamma, \delta, q_0, \square, q_{\text{acc}}, q_{\text{rej}})$ mathematically. Except for some simple Turing Machines, this gets extremely messy and hard to comprehend for a human.
2. **Implementation description**: Write a sequence of steps, possibly enclosed in loops, so that a Turing machine can do the task of each step in a single, mostly-forward or mostly-backward pass. A mostly-forward (resp. mostly backward) pass is a sequence of transitions where the head moves right (resp. left), except for a constant number of transitions (that is independent of the length of the tape's content). Feel free to "call" other Turing Machines and to "abort" a call (if it takes too long to complete). If using more than one tape (but a constant number of tapes) and nondeterminism helps simplify the description, do so and specify your choice upfront. While designing Turing machines with more than one tape, explain the purpose of each tape (eg. "We store the value of variable v on tape 2").
3. **High-level description**: This is just an algorithm – like the ones you wrote in COL106 or COL351. We rely on the Church-Turing thesis which states that any such algorithm can be turned into a Turing Machine.

Unless specifically asked to define some Turing Machine formally, you should prefer writing an implementation description or a high-level description. **For the purpose of this homework, stick to implementation descriptions.**

1. **[1 mark]** We proved in class that the language $E_{\text{TM}} = \{w \mid \mathcal{L}(M_w) = \emptyset\}$ is unrecognisable. (Here M_w denotes the Turing machine whose description is w.) Prove that $\overline{E_{\text{TM}}}$ is recognisable.

Solution. To prove that the language $\overline{E_{TM}}$ is *recognizable* we will show how to construct a non-deterministic turning machine that essentially does the following.

1. Non-deterministically choose a string from Σ^* (possibly empty).
2. Simulate the turning machine M_w on the chosen string, and return its output.

First, suppose that such a NTM exists, then we will show that this proves that $\overline{E_{TM}}$ is recognizable. To see this, consider any turning machine M_w. If this machine accepts some string $x \in \Sigma^*$, then the *run* of our NTM in which it chooses the string x will be an accepting run, therefore this NTM accepts such M_w. Now, suppose M_w does not accept any string. Then, since the NTM ultimately simulates M_w on a string, there can not exist any accepting run of of this NTM on M_w. Hence, we can conclude that existence of such a NTM would imply recognisability of $\overline{E_{TM}}$.

Now, we shall give an *implementation description* of a 2-Tape NTM that realizes the above given *vague description*. Unless specified otherwise, it is assumed that in a transition tape heads do not move (i.e. a *stay*).

1. Initially, the first tape would contain the input string w, and the second tape would be empty.
2. **Generate Phase:** Transition into a state q_{decide}. Whenever the turning machine is in the state q_{decide} it will non-deterministically select one of the following 2 *epsilon* transitions.
   (a) It can transition into state q_{write}. In this state the following happens.
      i. Non-deterministically select a *symbol* from Σ.
      ii. Write the symbol at the position of the current head of the second tape and move it to its right.
      iii. Transition back to the state q_{decide}.
   (b) It can transition into state q_{stop} (Generate phase ends).
3. **Simulate Phase:** This phase starts once the turning machine hits the q_{stop} state.
   (a) Copy the contents (string) of tape 2 and paste it after the input stored in tape 1 (separated by a special symbol) so that tape 1 now holds some encoding of form $(\langle M_w \rangle, x)$.
   (b) Call the universal turning machine (A_{TM}) with contents of tape 1 as its input and return its result (accept *iff* it accepts).

To see how this NTM is equivalent to the originally envisioned machine note that in the generate phase the NTM can non-deterministically generate any string from Σ^* in tape 2. Then in simulate phase we copy it and append it to the end of tape 1 and use the universal turning machine to check if M_w accepts the generated string or not.

2. **[1 mark]** An *enumerator* is a 2-tape DTM connected to a printer. One of its tapes is called the "work tape" and the other is called the "print tape". The enumerator has a special state q_{print}. The computation of an enumerator proceeds just like a usual 2-tape DTM, except that it starts with both tapes being empty, and whenever the enumerator enters state q_{print}, the (non-blank) content of the print tape is sent to the printer. A language L is said to be *enumerable* if there exists an enumerator E such that

   - For every $x \in L$, E prints x at least once.
   - For every $x \notin L$, E never prints x.

   Prove that a language is enumerable if and only if it is recognisable. For the 'if' part, the following observation might be useful. An enumerator with k work tapes can be simulated by an enumerator with one work tape, just like a k tape Turing machine can be simulated by a single tape Turing machine. You should be able to prove this, although you don't have to give the proof in the homework. Proving one direction of the 'if and only if' is harder than the other. A correct proof of the harder direction gets the one mark.

Solution. We will prove from both sides.

Claim 0.1

Language L is *enumerable* $\implies$ L is *recognizable*

Proof. This is the easier direction. Suppose E is the enumerator of L. Then consider the 4-tape turning machine whose implementation description is as follows.

1. Initially, input will be one the first tape and the remaining 3 tapes will be empty.
2. Simulate 2-tape DTM enumerator E on the 2 of the remaining 3 tapes until it reaches the state q_{print}.
3. Once it reaches state q_{print}, copy the input from tape 1 and paste it into tape 4, then copy the contents of *print* tape and paste it to tape 4 (after the input).
4. Check if content of tape 4 is of form ww, if so, *accept* the string.
5. Else, clear tape 4 and continue the simulation of enumerator.

To prove that this TM recognized the language L, consider any string $w \in L$. By definition of enumerator, it will eventually *print* the string w, therefore when it prints the string w the TM would accept the input string w as content of tape 4 would be of form ww. Similarly, suppose the TM accepts some string w, then it means that the enumerator must have printed the string w. But, by definition of enumerator it only prints strings that are in L so this implies $w \in L$. □

Claim 0.2

Language L is *recognizable* $\implies$ L is *enumerable*

Proof. Suppose M is the Turing machine which recognizes L. Also, suppose f is some computable function which takes as input some natural number, and outputs a string in Σ^* such that all strings in $\Sigma*$ are enumerated by f. One example of such a computable function is

1. $f(1) = \varepsilon$
2. $f(n) =$ representation of $n-2$ in base $|\Sigma|$ by using symbols from Σ.

Then consider the enumerator with three work tapes and one print tape whose implementation description is as follows.

1. Initially all the 4 tapes will be empty.
2. **Preprocess Phase:** Write the natural number 1 into work tape 2.
3. **Generation Phase:** Repeat the steps below forever
   (a) Erase the contents of work tape 3 and write the natural number 1 into it.
   (b) Repeat the below steps until the number in work tape 3 is $\leq$ number in work tape 2.
      i. Erase contents of work tape 1, and copy contents of work tape 3 into work tape 1.
      ii. Use the computable function f to replace the number in work tape 1 with the corresponding string in $\Sigma*$

iii. Simulate the TM M on work tape 1 upto at most n number of transitions where n is the number written in work tape 2.

iv. if M reaches accept state, once again copy content of work tape 3 into print tape, use the function f to generate the corresponding string and goto the state q_{print} to print this string. Afterwards, increment the number in work tape 3 and goto next iteration.

v. If M rejects, or we run out of transitions, increment the number in tape 3 and goto next iteration.

(c) Increment the number in work tape 2.

Now to prove that this enumerator is correct, we need to show that :

1. If $w \in L$ be some string, then E prints w at least once. Suppose $|w| = n$ and M accepts w in k transitions. Then, when content of work tape 2 will be the number $\max(n, k)$ and content of work tape 3 will be the number k, the enumerator will generate the string w into work tape 1. M will accept it without running out of transitions, so w will be copied to print tape and printed. Hence, the constructed enumerator prints all strings $w \in L$ at least once.
2. E never prints any string not in L. Note that E only prints a string whenever M accepts it. Therefore E, as constructed, can never print a string not in L.

□

3. **[1 mark]** A *two-stack Nondeterministic Pushdown Automaton* (2-NPDA) is like an NPDA, except that it has two stacks. In every transition a 2-NPDA reads 0 or 1 input symbols, pops 0 or 1 symbols from each stack, changes state, and pushes 0 or 1 symbol on each stack. (The transition relation Δ is a subset of $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon^2 \times Q \times \Gamma_\varepsilon^2$, where Q is the state space, Σ the input alphabet, and Γ the stack alphabet.) Prove that if a language is Turing-recognisable if and only if it is recognised by a 2-NPDA. Proving one direction of the 'if and only if' is harder than the other. A correct proof of the harder direction gets the one mark.

Solution. We shall break the proof into two parts.

Recognisable by 2-NPDA $\implies$ Turing-recognisable This is the easier part. In this case, we can directly simulate a 2-NPDA on a 3-tape NTM, as follows:

The state space consists of all states of the 2-NPDA along with one distinct new state each for each transition. Treat the first tape as read-only with the input on it, both of the remaining tapes as stacks, the head of the first tape as the location of the first character of the string, and the remaining two heads as pointers to the top of the stacks. Now note the following:

1. If we read an input character, then the first head moves to the right, else it stays there.
2. Consider what happens to one of the tapes on a transition. We have the following four types of transitions, which can be handled with the help of an intermediate state as follows. Note that a distinct intermediate state q_t is added for each transition $(q_1, a, (c_1, c_2), q_2, (d_1, d_2))$, and in each case we add two transitions (call them type 1 and type 2 respectively). A transition of the NTM is an element of $Q \times \Gamma_\epsilon^3 \times Q \times \Gamma_\epsilon^3 \times \{L, R, S\}$. Wlog assume that we talk about the second tape, the third tape can be handled similarly.
   (a) None of push and pop happen on this stack (which corresponds to a tape in the NTM): in this case, we add two transitions: one from the current state to the intermediate state without changing anything on the NTM, and one that carries out the read from the string (iff needed) as well as the transition to the state that the 2-NPDA transition intended to go to (and in both cases, stay at the same place on the relevant tape). So for a transition $(q_1, -, (\epsilon, -), q_2, (\epsilon, -))$, we add two transitions $(q_1, (-, c, -), q_t, (-, c, -), S)$ and $(q_t, (-, c, -), q_2, (-, c, -), S)$ to the Turing machine (the $-$ denotes don't-cares, i.e., they will be filled in the merge step towards the end, which is done in view of conciseness).
   (b) We push a character onto the stack but don't pop a character: in this case, we add two transitions: one that goes to the right and changes nothing and transitions to the intermediate state, and the other that reads from the string (iff needed) and stays, while changing the current tape content to the character intended to be pushed on the stack and transitioning to the intended state. So for a transition $(q_1, -, (\epsilon, -), q_2, (c, -))$, we add two transitions $(q_1, (-, r, -), q_t, (-, r, -), R)$ and $(q_t, (-, r, -), q_2, (-, c, -), S)$.
   (c) We pop a character from the stack but don't push a character: in this case, we add two transitions: one that goes to the left and changes nothing and transitions to the intermediate state, and the other that reads from the string (iff needed) and stays and transitions to the intended state. So for a transition $(q_1, -, (c, -), q_2, (\epsilon, -))$, we add two transitions $(q_1, (-, c, -), q_t, (-, c, -), L)$ and $(q_t, (-, r, -), q_2, (-, r, -), S)$.
   (d) We both pop and push a character from the stack: in this case, we add two transitions: one that stays and changes the current character and transitions to the intermediate state, and the other that reads from the string (iff needed) and stays and transitions to the intended state. So for a transition $(q_1, -, (c, -), q_2, (d, -))$, we add two transitions $(q_1, (-, c, -), q_t, (-, d, -), S)$ and $(q_t, (-, r, -), q_2, (-, r, -), S)$.

   Now note that all of this was for one tape (and its corresponding stack). We can combine this analysis with the analogous analysis for the other tape (by noting that we always move to the intermediate state, and read the string (iff needed) always in the second state, preserving the 'global' parts of the transition, as opposed to the tape-local changes), to construct a complete table of transitions. Note that having a separate intermediate state for each transition of the 2-NPDA ensures that we never have stray unintended transitions in the NTM, and since none of these states is an accepting state, we never accept when the run is stuck in the case of a non-deterministic run. We have omitted the complete analysis here since it has a total of $4 \times 4 = 16$ cases and it is very cumbersome to explicitly list it out.
3. We can add a simple transition from all the accepting states to a single new accepting state, and add a transition from all non-accepting states to a single rejecting state, and this will be triggered

when we reach a blank on tape 1.

This construction works because of the following inductive claim: at a given position in some particular run of the 3-tape NTM on the string, either we are in an intermediate stage, or the following happens: the next character of the string to be read is pointed to by the head of the first tape, and there exists a run of the 2-NPDA that has the contents of the first stack and the second stack identical to the contents of the first tape and the second tape before and including the position of the respective tapes (apart from the bottom symbol, of course).

Turing-recognisable $\implies$ Recognisable by 2-NPDA This is the harder part. We construct a 2-NPDA which does things in the following stages as preprocessing:

1. Go from the initial state to a preprocessing state, and at the end of this transition, both stacks have a bottom symbol on them. So, we have a transition $(q_{init}, \epsilon, (\epsilon, \epsilon), q_{preprocess,1}, (\perp, \perp))$.
2. In a single pass, read the whole string into the first tape, and add an epsilon transition to the second preprocessing state (and after the first preprocessing state is exited, we will never read the input again and work only with epsilon transitions) – this helps in ensuring that if we accept, we accept only on an input that is read completely. So we have transitions $(q_{preprocess,1}, c, (\epsilon, \epsilon), q_{preprocess,1}, (c, \epsilon))$ and $(q_{preprocess,1}, \epsilon, (\epsilon, \epsilon), q_{preprocess,2}, (\epsilon, \epsilon))$.
3. In the second preprocessing state, we transfer everything from the first stack to the second stack (it is easy to add such transitions), till we reach the bottom of the stack, where we transition to the initial state of the Turing machine from where we never reach either the preprocessing states or the initial state of the 2-NPDA ever again. So we have transitions $(q_{preprocess,2}, \epsilon, (c, \epsilon), q_{preprocess,2}, (\epsilon, c))$ and $(q_{preprocess,2}, \epsilon, (\perp, \epsilon), q_0, (\perp, \epsilon))$.

Now we shall aim to preserve the following invariant:

Let x_1 be the string that is formed by the contents of the first stack (top to bottom), and let x_2 be the string that is formed by the contents of the second stack (assuming that stack grows leftwards), apart from the bottom symbols. Then, $rev(x_1) \cdot x_2$ is a prefix of the corresponding Turing machine's tape (on that instant of the run) such that the remaining part consists of only blank symbols, and the corresponding location of the element at the top of the second stack in this string is the position of the head of the DTM at that instant in that run. We are in either an intermediate state or in the same state as the Turing machine would have been in at this instance of the run.

For a run of length 0, we are trivially done, since after the preprocessing states, we have x_2 is the input and x_1 is empty (note that when constructing these strings, we have to start from the second-last element of the stack in order to ignore the bottom symbol).

Note that the only transitions allowed in a state are going either to the left or to the right.

Now we aim to add the following transitions for the following cases (for any given transition of the Turing machine, we need to handle all of the following cases):

1. If x_1 is empty: this implies that we are at the first position in the Turing machine (from the invariant), and at this point, we are at the bottom of the first stack. A replace-and-move-left transition corresponds to just pushing and popping the bottom symbol on the first stack, and performing the replacement on the second stack (insertion if we are at the bottom of the second stack), so we need to add in a single relevant transition here that does this (along with changing the state of course). A replace-and-move-right transition corresponds to the following two transitions on the 2-NPDA: pop and push to replace (or just push if we are on the bottom of the second stack) on the second stack and move to an intermediate state on this transition, and then move from that state to the required state, and pop the element from the second stack and push onto the first stack. All of these can be handled by adding epsilon transitions (i.e., not reading anything from the input), and at most 2 of them.
2. If x_1 is not empty but x_2 is empty: this implies that we are at a blank in the Turing machine (from the invariant), but there is something to the left of the pointer on the Turing machine's tape. A replace-and-move-right transition corresponds to the following transition of the 2-NPDA: push the replaced character on the first stack and do the intended state transitions. A replace-and-move-left transition corresponds to the following transitions of the 2-NPDA: push the replaced character on the second stack and go to an intermediate state, and pop from the first stack and push the popped character onto the second stack (and go to the intended state).
3. If neither x_1 is empty, nor x_2 is empty: a replace-and-move-right transition corresponds to the following transition of the 2-NPDA: push the replaced character on the first stack and pop from

the second stack and do the intended state transitions. A replace-and-move-left transition corresponds to the following transitions of the 2-NPDA: replace on the second stack and move to an intermediate state, and then pop the character on the first stack and push the character on the second stack and go to the intended state.

Note that in order to carry out the above transitions, while handling the cases where the relevant stacks are empty, we would need to take another transition into a state (by popping and pushing the bottom symbol, and thus this would require at most one more transition than originally specified in the above description).
Using these transitions, we note that we can maintain the mentioned invariant, so we have simulated a Turing machine on a 2-NPDA (the final stopping conditions are fairly straightforward).

4. Consider the language $L = \{w \mid \text{the Turing machine } M_w \text{ is a decider}\}$. Using mapping reductions, prove the following statements.

   1. **[1 mark]** L is unrecognisable.
   2. **[1 mark]** $\overline{L}$ is unrecognisable.

   (Note that L is different from the language $\{w \mid \text{the Turing machine } M_w \text{ recognises a decidable language}\}$. Rice's Theorem applies to the latter, but not to the former. Moreover, the version of Rice's Theorem discussed in class lets you claim undecidability only.)

Solution. We solve part 1 first. Consider the language E_{TM} which consists of encodings of Turing machines which only accept the empty language. We shall show that E_{TM} is mapping-reducible to L, and since E_{TM} is unrecognizable, part 1 shall follow.

Consider the function f where $f(w)$ is the encoding of a Turing machine which does the following to an input x:

Run M_w on all strings of length $\leq |x|$ for at most $|x|$ transitions. If any string is accepted, then get into an infinite loop, else accept the input.

Now consider M_w for a given string w. If $w \in E_{TM}$, then M_w doesn't accept any string, and hence only runs for a finite amount of time. So, $M_{f(w)}$ always terminates, and hence $M_{f(w)}$ is a decider, so $f(w) \in L$.

If $w \notin E_{TM}$, then there exists at least one string t such that M_w accepts t. Let k be the length of the accepting run of M_w on t. Then, consider any string of length $1 + \max(|t|, k)$. Since the length of t is less than the length of the input, it must be one of the strings on which M_w is run on. Since the accepting run of M_w has length less than that of the input, t is accepted by M_w. Hence, $M_{f(w)}$ goes into an infinite loop, and thus is not a decider.

From the above, we have that $w \in E_{TM} \iff f(w) \in L$, so $E_{TM} \leq_m L$, which completes the proof.

Now we solve part 2. We shall exhibit a mapping reduction from $\overline{HALT}$ to $\overline{L}$, and since $\overline{HALT}$ is unrecognizable, part 2 shall follow.

Consider the function f where $f(w, x)$ is the encoding of a Turing machine which does the following to an input t:

Erase t from the tape, and copy x onto the tape, and simulate M_w on x. If M_w halts on x, accept, else it is stuck infinitely and doesn't halt.

Now if $(w, x) \in \overline{HALT}$, then M_w doesn't halt on x, so for any input t to $M_{f(w,x)}$, this machine never halts on t, and hence can't be a decider. So $f(w, x)$ is not a decider, and thus $f(w, x) \in \overline{L}$.

Now if $(w, x) \notin \overline{HALT}$, then M_w halts on x, so $M_{f(w,x)}$ always halts and accepts on any input t, so it is a decider, from where we get that $f(w, x) \notin \overline{L}$.

From here, we get to know that $(w, x) \in \overline{HALT} \iff f(w, x) \in \overline{L}$, so $\overline{HALT} \leq_m \overline{L}$, whence the proof is complete.

5. **[1 mark]** After an exhausting day of finishing and submitting homeworks (including COL352, of course), Alice falls into deep sleep, and in her dream, finds herself in Wonderland. She finds that Turing Machines in Wonderland have 2 tapes – one which has the input initially, and the other one which is called the "query tape". Each Wonderland Turing Machine (WTM) also has 3 more special states – q_{query}, q_{yes}, and q_{no} (in addition to the usual $q_0, q_{\text{accept}}, q_{\text{reject}}$). The computation of a WTM proceeds just like a usual 2-tape DTM, except that as soon as a WTM enters state q_{query}, its state changes instantaneously to q_{yes} or q_{no} depending on the content w of the query tape: if $w \in$ DIAG, the machine enters q_{yes}, and if $w \notin$ DIAG, the machine enters q_{no}. The definition of the language DIAG in Wonderland is same as that in real-life: it is the set of all strings x such that the **real-life** Turing machine whose description is x doesn't accept x. (Note that the domain of the transition function of a WTM is $(Q \setminus \{q_{\text{accept}}, q_{\text{reject}}, q_{\text{query}}\}) \times \Gamma$, where Q is its state space and Γ is its tape alphabet.)

   Obviously, every language that is recognisable in real-life is Wonderland-recognisable too (don't have any transition into q_{query}). Give an example of a language which is unrecognisable in real-life but, in fact, decidable in Wonderland (0 marks for this). More importantly, Alice is wondering whether there exists a Wonderland-unrecognisable language too. Help her by giving an explicit example of such a language, and prove that your example language is indeed Wonderland-unrecognisable.

   Solution. $DIAG$ is an example of a language which is unrecognizable in a real-life TM but is *decidable* by a WTM. As we may simply copy the input of the WTM into its query tape and immediately goto the state q_{query}. If we transition into state q_{yes} then we *accept* the string, else we reject it. This, WTM decides the language $DIAG$.
   Interestingly, the language $DIAG^{WTM}$ defined as

   $$DIAG^{WTM} = \{w \mid w \notin \mathcal{L}(W_w)\}$$

   is unrecognizable. Where W_w denotes the wonderland turning machine whose encoding is w (in case w is not a valid encoding, W_w is a WTM which rejects all strings).

   Proof. To prove this, we will use the proof very similar to the proof of unrecognizability of $DIAG$ in real-life. Suppose, $DIAG^{WTM}$ is recognizable in wonderland. Then let W_w be the WTM that recognizes $DIAG^{WTM}$. Then consider the string w. Two cases arise :

   1. W_w *accepts* w. Since $\mathcal{L}(W_w) = DIAG^{WTM}$, this implies that $w \in DIAG^{WTM}$. But $w \in DIAG^{WTM}$ implies that $w \notin \mathcal{L}(W_w)$ by definition of $DIAG^{WTM}$ which is a contradiction.
   2. W_w does not *accept* w. Since $w \notin \mathcal{L}(W_w)$ this implies that $w \in DIAG^{WTM}$, and since W_w recognizes $DIAG^{WTM}$ it must also accept w which is again a contradiction.

   Therefore it can not be the case that such a WTM W_w exists which recognizes $DIAG^{WTM}$. Hence it is proved that $DIAG^{WTM}$ is unrecognizable even in the wonderland. □