CS Theory (Fall '25) Assigned: Nov 2, 2025

Homework 4 Solutions

Instructors: *William Pires and Toniann Pitassi* Due: Nov 13, 2025 at 11am

0 Exercises

Here are some **highly** recommended exercises. You should not turn in your solutions for these, and we will not grade them.

(a) Prove that for every infinite set S, the following are equivalent:

(i) There exists a function $g : \mathbb{N} \to S$ that is onto (i.e., g is surjective).

(ii) There exists a function $f : S \to \mathbb{N}$ that is one-to-one (e.g., f is injective).

Solution.

Proof. (i) $\implies$ (ii): Assume there exists a function $g : \mathbb{N} \to S$ that is onto. Define the function $f : S \to \mathbb{N}$ so that for $s \in S$,

$$f(s) = \min g^{-1}(s),$$

where g^{-1} is the inverse image. In other words, f maps $s \in S$ to the smallest natural number $n \in \mathbb{N}$ such that $g(n) = s$. The preimage on any $s \in S$ is nonempty because g is surjective, plus a nonempty set of natural numbers will always have a minimum by the well ordering principle. Therefore f is well-defined. Furthermore, observe that $g(f(s)) = s$ for all $s \in S$ by definition. Now to show that f is injective, consider any $s_1, s_2 \in S$ such that $f(s_1) = f(s_2)$. This implies that $g(f(s_1)) = g(f(s_2))$. But $g(f(s_1)) = s_1$ and $g(f(s_2)) = s_2$ as noted earlier, so $s_1 = s_2$.

(ii) $\implies$ (i): Assume there exists a function $f : S \to \mathbb{N}$ that is one-to-one. Since f is injective, its inverse image on any natural number n must have size at most 1. Otherwise, if $f^{-1}(n)$ contains distinct elements $s_1, s_2 \in S$, then $f(s_1) = f(s_2) = n$ but $s_1 \neq s_2$, a contradiction. In fact, the inverse image of any injective function must be empty or a single point set. Therefore we can define the function $g : \mathbb{N} \to S$ as follows: for each $n \in \mathbb{N}$

$$g(n) = \begin{cases} s & \text{if } f^{-1}(n) = \{s\} \\ s_0 & \text{if } f^{-1}(n) = \emptyset \end{cases}$$

where s_0 is some fixed element we choose from S. To show g is surjective, consider any $s \in S$. Since f maps $s \in S$ to a natural number, $f(s) = n$ for some $n \in \mathbb{N}$. Therefore $f^{-1}(n) = \{s\}$ and $g(n) = s$. □

(b) Are the following languages (i) decidable or (ii) undecidable. Prove your answer.

- $L = \{\langle M \rangle \mid M \text{ is a TM and for every even length } \ell, \text{ there is a string } w, |w| = \ell, \text{ s.t. } M \text{ halts on } w\}$

Solution.

L is undecidable.

Proof. We show a Turing reduction $A_{TM} \leq_t L$, in class we've shown that A_{TM} is undecidable so this proves L is not decidable. Let N be a decider for L. We can construct a decider for A_{TM} as follows:

Algorithm 1 A decider D for A_{TM}

Input: $\langle M, w \rangle$ where M is a TM and w is a string

1: (Note: Technically the input string is not always a valid encoding, but we can fix something not in L and return it as output for this case. *For the purpose of this course, you can always assume the input has the right encoding.*)
2: Consider the TM M' defined as follows:
"On input x,
1. If x has odd length, *accept* x.
2. If x has even length:
3. Run M on input w. ▷ If M loops on w, we get "stuck" here and M' loops on x
4. If M accepted w, accept x.
5. If M rejected w, loop forever. " ▷ This means M' doesn't halt on x.
3: Run N on $\langle M' \rangle$. ▷ N always halts because it *decides* L
4: If N accepted, accept $\langle M, w \rangle$.
5: If N rejected, reject $\langle M, w \rangle$.

[a] We now show D is a decider for A_{TM}. We first observe the following: For any string x with $|x|$ being even, we can see that:

- If M loops on w, then M' loops on w (on line 3).
- If M rejects on w, then M' loops on w (on line 5).
- If M accepts w, then M' halts and accepts x (on line 4).

So, if M accepts w, M' halts on every string of even length. If M doesn't accept w, then M' loops on all strings of even length. So we have that $\langle M' \rangle \in L$ if and only if M accepts w.

(1) If $\langle M, w \rangle \in A_{TM}$, then M accepts w, so $\langle M' \rangle \in L$. So N accepts $\langle M' \rangle$ on line 3, and D accepts $\langle M, w \rangle$ on line 4.

(2) If $\langle M, w \rangle \notin A_{TM}$, then M doesn't accept w, so $\langle M' \rangle \notin L$. So N rejects $\langle M' \rangle$ on line 3, and D rejects $\langle M, w \rangle$ on line 5.

Therefore, D always halts and accepts $\langle M, w \rangle$ iff $\langle M, w \rangle \in A_{TM}$. So D decides A_{TM}, meaning $A_{TM} \leq_t L$. □

[a] Small note: to compute a description of M' from the description of M and w we simply needs to add extra states to M and to check the length of the input x. So step 1 always halts

- $L = \{\langle M \rangle \mid$ M is a TM and $L(M)$ is finite and has size divisible by 3 $\}$

Solution.

L is undecidable.

Proof. We show a Turing reduction $A_{TM} \leq_t L$, in class we've shown that A_{TM} is undecidable so this proves L is not decidable. Let N be a decider for L. We can construct a decider for A_{TM} as follows:

Algorithm 2 A decider D for A_{TM}

Input: $\langle M, w \rangle$ where M is a TM and w is a string

1: (Note: Technically the input string is not always a valid encoding, but we can fix something not in L and return it as output for this case.)
2: Build a TM M' as follows:
"On input x,
1. If x is not ϵ, 0, nor 1, *reject* x.
2. Else if x is ϵ or 0, *accept* x.
3. Else: ▷ this means $x = 1$
4. Run M on input w.
5. If M accepted w, accept x.
6. If M rejected w, reject x. "
3: Run N on $\langle M' \rangle$. ▷ N always halts because it *decides* L
4: If N accepted, accept $\langle M, w \rangle$.
5: If N rejected, reject $\langle M, w \rangle$.

We now show D is a decider for A_{TM}. We first observe the following:

- M' always accepts ϵ and 0 (on line 2).
- M' accepts 1 if and only if M accepts w (on line 5).
- M' rejects every other strings (on line 1).

So if M accepts w we have $L(M') = \{\epsilon, 0, 1\}$ and if M doesn't accept w we have $L(M') = \{\epsilon, 0\}$. By looking at the two cases for $|L(M')|$ we have $\langle M' \rangle \in L$ if and only M accepts w.

(1) If $\langle M, w \rangle \in A_{TM}$, then M accepts w, so $\langle M' \rangle \in L$. So N accepts $\langle M' \rangle$ on line 3, and D accepts $\langle M, w \rangle$ on line 4.

(2) If $\langle M, w \rangle \notin A_{TM}$, then M doesn't accept w, so $\langle M' \rangle \notin L$. So N rejects $\langle M' \rangle$ on line 3, and D rejects $\langle M, w \rangle$ on line 5.

Therefore, D always halts and accepts $\langle M, w \rangle$ iff $\langle M, w \rangle \in A_{TM}$. So D decides A_{TM}, meaning $A_{TM} \leq_t L$. □

(c) Prove that L_{pug} (as defined in the bonus problem) is undecidable.

Solution.

Proof. We show a Turing reduction $Halt_{TM} \leq_t L$, in class we've shown that $Halt_{TM}$ is undecidable so this proves L is not decidable. Let N be a decider for L_{pug}. Then we can construct a decider for $Halt_{TM}$ as follows:

Algorithm 3 A decider D for A_{TM}

Input: $\langle M, w \rangle$ where M is a TM and w is a string

1: (Note: assume that we first checked if the input string is a valid encoding, and rejected if it was not one)
2: Build a TM M' as follows:
"On input x,
1. If x is the string 000111000111 or 000000000000, *reject* x.
2. If x is the string 010101010101 or 111000111000, loop forever.
3. Else:
4. Run M on input w. ▷ if M doesn't halt on w, M' is "stuck" here and it loops on x
5. If M halted on input w, accept x.
3: Run N on $\langle M' \rangle$. ▷ N always halts because it *decides* L_{pug}
4: If N accepted, accept $\langle M, w \rangle$.
5: If N rejected, reject $\langle M, w \rangle$.

We now show D is a decider for $Halt_{TM}$. We first observe the following:

- M' always rejects 000111000111 and 00000000000 (on line 1).
- M' always loops on 10101010101 and 111000111000 (on line 2).
- If M halts on w, then M' accepts 11111111111 and 101010101010 (on line 5). If M doesn't halt on w then M' loops on 11111111111 and 101010101010 (on line 4).

So if M halts on w, then M' is pugnacious so $\langle M' \rangle \in L_{pug}$. And if M doesn't halt on w, then M' isn't pugnacious (since it doesn't accept 11111111111) so $\langle M' \rangle \notin L_{pug}$.

(1) If $\langle M, w \rangle \in Halt_{TM}$, then M halts on w, so $\langle M' \rangle \in L_{pug}$. So N accepts $\langle M' \rangle$ on line 3, and D accepts $\langle M, w \rangle$ on line 4.

(2) If $\langle M, w \rangle \notin Halt_{TM}$, then M doesn't halt on w, so $\langle M' \rangle \notin L$. So N rejects $\langle M' \rangle$ on line 3, and D rejects $\langle M, w \rangle$ on line 5.

Therefore, D always halts and accepts $\langle M, w \rangle$ iff $\langle M, w \rangle \in Halt_{TM}$. So D decides A_{TM}, meaning $Halt_{TM} \leq_t L$. □

1 Turing Machines (20 points)

Consider the following two Turing Machines M_1 and M_2. If a transition isn't drawn, it goes to the reject state. The accept state is q_{acc}. We use B to represent a blank ($\sqcup$ in the book).

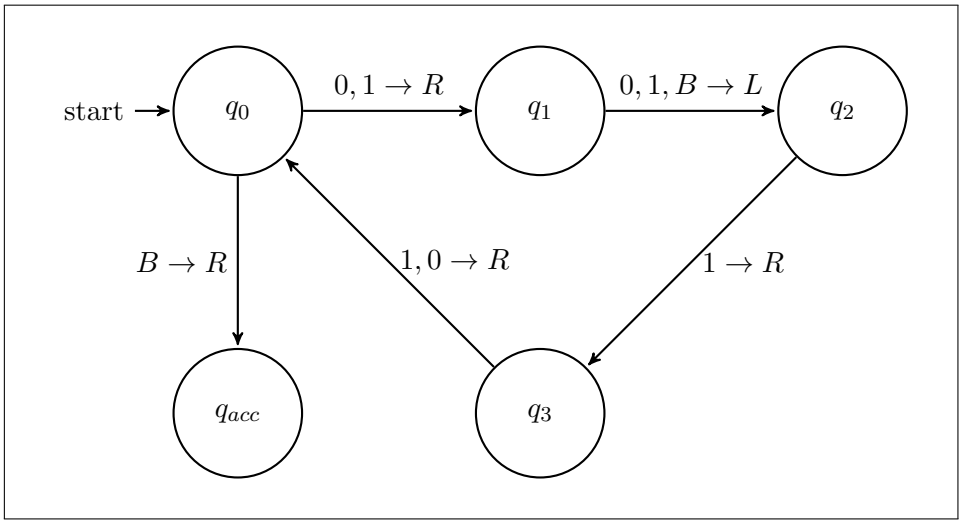

Figure 1: The Turing Machine M_1

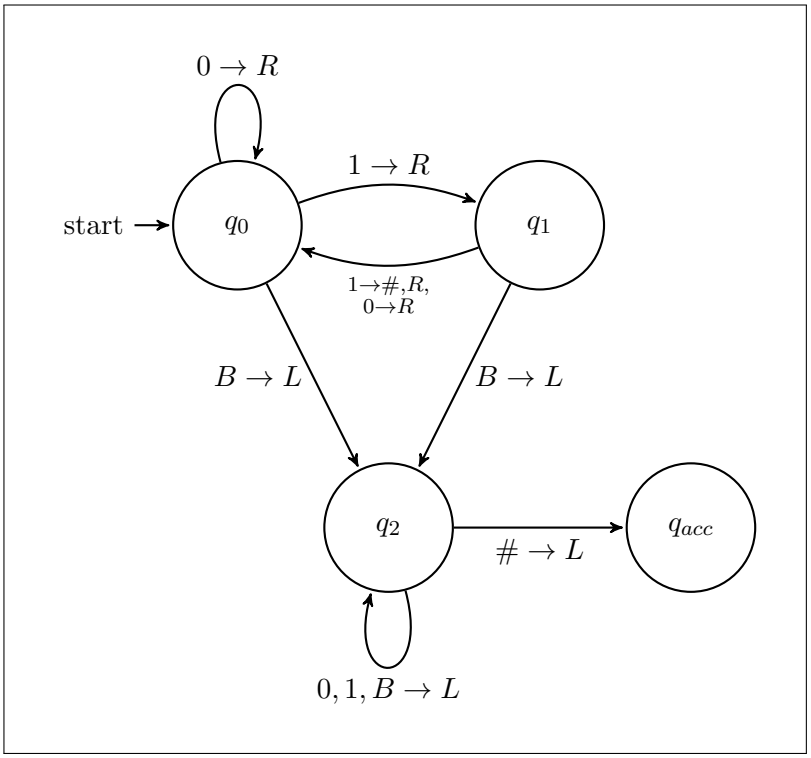

Figure 2: The Turing Machine M_2

Answer each of the following. No justification required.

- (2 points) True or False ? The Machine M_1 always halts.
- (6 points) What is $L(M_1)$, give your answer using set notation. [1]
- (2 points) True or False ? $L(M_1)$ is decidable.
- (2 points) True or False ? The Machine M_2 always halts.
- (6 points) What is $L(M_2)$, give your answer using set notation.
- (2 points) True or False ? $L(M_2)$ is decidable.

To answer these questions, it is worth trying to run the TM on a few short strings and try to see the pattern. [2]

Solution.

- True. Any time this TM moves left, it must have started in q_1. In order to return to q_1, it must move right at least twice (it may also just halt, in which case we're done). Since there are at least two rightward moves for every leftward move, the TM will be farther than any input on the tape after $2n$ steps. Therefore it must at some point read a blank symbol. If it is in q_0, it accepts. If it is in q_2 or q_3, it rejects. If it is in q_1, we break into cases. If the symbol to the left is 0 or B, the TM rejects. If the symbol to the left is 1, the TM moves right, once again reading the blank symbol while it is in q_3, and rejects.
- $L(M_1) = \{w \mid w \text{ has even length and all odd positions are } 1\}$.
- True. M_1 always halts, so M_1 is a decider for $L(M_1)$.
- False. Consider the string 0^n. The TM will read to the end of the string, then enter q_2, and then move left forever (eventually just staying at the first square).
- $L(M_2) = \{w \mid w \text{ contains two consecutive 1's}\}$.
- M_2 is not a decider for $L(M_2)$, but $L(M_2)$ is decidable. It is recognized by the regular expression $\Sigma^*11\Sigma^*$, so it is regular. All regular languages are decidable.

[1] For instance $L(M_1) = \{w \mid w \text{ starts with 11 and doesn't contain 00}\}$.

[2] You can use websites such as this one.

Instructions for problems 2 to 5

To prove a language L is recognizable (or decidable), you must give a **high-level** description of a TM that accepts L and proof it is correct. See the last page of the HW for what we expect when we mean **high-level** description and what we expect for the proof of correctness.

For each question, you can ignore inputs that are *bad encodings*. For instance in Problem 3, you can ignore inputs that are not of the form $\langle M \rangle$.

When proving something is not decidable, make sure you write a *proof* (See Chapter 5 in the book for examples). You are not allowed to use Rice's theorem in any of these problems. If you want to use the fact some language L is not decidable/recognizable, we must have seen this language in class[3]

2 Streaming "Algorithms" (10 points)

Before introducing Turing Machines, we introduced streaming algorithms (see the lecture notes for the definition). Is the following statement true ? Give a proof or disprove it.

If $L \subseteq \Sigma^*$ has a streaming algorithm[4], then L is decidable.

Note: There are many ways to prove (or disprove) the above statement. Make sure that you give a *proof*.

Solution.

The statement is false.

Proof. A streaming algorithm can accept any language, including undecidable ones, simply by recording the entire input, and at the end accepting iff the memory content contains a string in the language (whether or not this is computable is a different question).

For example, we can construct the following streaming algorithm $(\Sigma, \Pi, I, \delta, \gamma)$ for A_{TM}: the input alphabet is the same as the alphabet for A_{TM}, and the memory alphabet is $\Pi = \Sigma$. At every point in time, the contents of memory will contain the prefix of the input string that has been read so far. Therefore the rules are:

- Initialization: The memory content is empty, $M = \epsilon$, because we have not seen any part of the input string yet.
- Update: If $M \in \Sigma^*$ is the current memory content, then after reading in the next symbol $\sigma \in \Sigma$, the updated memory content should be $\delta(M, \sigma) = M \circ \sigma$.
- Stopping: If $M \in A_{TM}$ then $\gamma(M) = 1$; otherwise $\gamma(M) = 0$.

Since the final memory content is exactly the input string, the streaming algorithm will accept exactly A_{TM}. We have shown that A_{TM} has a streaming algorithm, but it has also been shown in class to be undecidable. Therefore the statement is false. □

[3] for instance you're allowed to use A_{TM} or $HALT_{TM}$.
[4] I.e. there is a streaming algorithm $\mathcal{A}$ with $L(\mathcal{A}) = L$

3 Decidability/Recognizability of L_{pair} (24 points, 12 points each)

Let

$$L_{\text{pair}} := \{\langle M \rangle \mid \ M \text{ is a TM that accepts two consecutive binary numbers}\}.$$

For example, if a Turing machine M accepts both 100 and 101 then $\langle M \rangle \in L_{\text{pair}}$.

a) Prove that L_{pair} is recognizable.

b) Prove that L_{pair} is not decidable.

Solution.

a) We give a recognizer for L_{pair}:

Algorithm 4 Recognizer for L_{pair}

Input: $\langle M \rangle$ where M is a TM

1: (Note: assume that we first checked if the input string is a valid encoding, and rejected if it was not one).
2: Let C_0 be the starting configuration of M when run on input 0.
3: $\boldsymbol{C} \leftarrow (C_0)$
4: **for** $i = 0, 1, 2, 3, \ldots$ **do**
5: **for** $j \in |\boldsymbol{C}|$ **do**
6: Load configuration $\boldsymbol{C}_j$ and run it one more step.
7: Save the resulting configuration in $\boldsymbol{C}_j$.
8: **end for**
9: Let C_i be the starting configuration of M when run on input $\langle i \rangle$.
10: Append C_i to $\boldsymbol{C}$.
11: If $\boldsymbol{C}$ contains two consecutive configurations that are both in the accept state, *accept*.
12: **end for**

Intuitively, this algorithm runs the TM on all possible integers. There is a problem with this naive approach though, which is that M might run forever on some integers, but accept two consecutive integers at some point past that sticking point. You would never see them if you just ran on each integer in series. To get around this, we simulate all runs in parallel (this is a technique called dovetailing).

Now we prove that this is a recognizer for L_{pair}. Suppose the algorithm accepts. This means that there are two consecutive integers such that M accepts when executed on them, so $\langle M \rangle \in L_{pair}$.

In the other direction, suppose $\langle M \rangle \in L_{pair}$. Then, there exists k such that M accepts $\langle k \rangle$ and $\langle k+1 \rangle$. Suppose M runs for ℓ_1 steps on $\langle k \rangle$ and ℓ_2 steps on $\langle k+1 \rangle$. Set $\ell = \max\{\ell_1, \ell_2\}$. By the time the algorithm given above runs the outer for loop $k + \ell + 1$ times, it will have seen M run on $\langle k \rangle$ and $\langle k+1 \rangle$ for at least ℓ steps, so it will see them both accept, just like we want. So once i has reached $k + 1 + \ell$, the algorithm will accept on line 11 (if it hasn't already accepted for a smaller value of i). So if $\langle M \rangle \in L_{pair}$, the algorithm accepts. [a]

So the algorithm accepts iff $\langle M \rangle \in L_{pair}$.

b) Suppose we have a decider D for L_{pair}. We give a reduction $A_{TM} \leq_t L_{pair}$, since A_{TM} isn't decidable, this shows L_{pair} isn't decidable. Let N be a decider for L_{pair}, we build a decider D for A_{TM} as follows:

Algorithm 5 A decider D for A_{TM}

Input: $\langle M, w\rangle$ where M is a TM and w is a string

1: Build a TM M' as follows:
"On input x,
1. Run M on w.
2. If M accepted w, accept x.
3. If M rejected w, reject x. "
2: Run N on $\langle M'\rangle$. ▷ N always halts because it *decides* L_{pair}
3: If N accepted, accept $\langle M, w\rangle$.
4: If N rejected, reject $\langle M, w\rangle$.

We now show D is a decider for A_{TM}. We first observe the following. On input x, M' does the following:

- If M rejects w, M' rejects x (on line 3).
- If M loops on w, M' loops x (on line 1).
- If M accepts w, M' accepts x (on line 2).

Hence, if M accepts w we have $L(M') = \Sigma^*$, so M' accepts the strings 1 and 10 which are the representations in binary of 1 and 2. If M doesn't accept w we have $L(M') = \emptyset$ so M' doesn't accept consecutive binary numbers. So we can see that $\langle M'\rangle \in L_{pair}$ iff M accepts w.

(1) If $\langle M, w\rangle \in A_{TM}$, then M accepts w, so $\langle M'\rangle \in L_{pair}$. So N accepts $\langle M'\rangle$ on line 3, and D accepts $\langle M, w\rangle$ on line 4.

(2) If $\langle M, w\rangle \notin A_{TM}$, then M doesn't accept w, so $\langle M'\rangle \notin L_{pair}$. So N rejects $\langle M'\rangle$ on line 3, and D rejects $\langle M, w\rangle$ on line 5.

Therefore, D always halts and accepts $\langle M, w\rangle$ iff $\langle M, w\rangle \in A_{TM}$. So D decides A_{TM}, meaning $A_{TM} \leq_t L$.

[a]Note that when proving your dovetailing is correct, you should be precise, and not just say "eventually i is large enough that it will accept".

4 Recognizable versus Decidable (20 points, 10 points each)

Classify each of the following languages as either: (i) decidable or (ii) undecidable. *Prove* your answers.

a) $L = \{\langle M, w\rangle \mid M$ is a TM, w a string and M moves its head left at least once on input $w\}$.

Solution.

L is decidable.

Proof. We can construct the following Turing Machine to decide L. There are two main strategies that you might take. The first is a little simpler and uses a pigeonhole principle idea. The second relies on looking at the transition function of the input TM in more detail. We'll show both solutions.

Solution 1: We claim the following TM decides the language:

Algorithm 6 Decider for L

Input: $\langle M, w\rangle$ where M is a TM (with states Q) and w is a string

1: (Note: assume that we first checked if the input string is a valid encoding, and rejected if it was not one)

2: For steps $i = 1, \ldots, |w| + |Q| + 1$:

   1. Simulate the ith step of running M on w by updating the current configuration t using the transition function of M.
   2. If M moved left, *accept*.
   3. If M transitioned to its accept or reject state, *reject*.

3: If we reach this step, *reject*.

This algorithm always terminates in $|w|+|Q|+1$ steps. Now we will show that this algorithm decides L.

First, we will show that all inputs that the algorithm accepts are in L. We do this by contrapositive If $\langle M, w\rangle \notin L$, M never moves left on w. Therefore, we will never hit the only part of the algorithm (line 2.2) that accepts. Since the algorithm always terminates, it must reject.

Now, we will show that all inputs in L are accepted by the algorithm. Again, we do this by contrapositive. Suppose that this algorithm rejects $\langle M, w\rangle$. This means either that M terminates without moving left within $|w| + |Q| + 1$ steps, or that it runs for more than $|w| + |Q| + 1$ steps without moving left. In the former case, $\langle M, w\rangle \notin L$. In the latter case, we claim that M never halts and never moves left on input w. After $|w|$ steps, since each step is a rightward move, M will be over a blank tape cell, and all cells to the right will be blank. Over the next $|Q| + 1$ steps, let the states be $q_1, \ldots, q_{|Q|+1}$. By the pigeonhole principle, there must be two indices $i < j \in \{1, \ldots, |Q| + 1\}$ such that $q_i = q_j$. This means that there is a loop: there are states $q_i, \ldots, q_j$ such that $q_i = q_j$ and $\delta(q_k, B) = (q_{k+1}, R, \sigma)$ for each $k \in \{i, \ldots, j - 1\}$. Since all squares to the right of the head are blank, M will continue executing this loop forever. Therefore, it will never move left, so $\langle M, w\rangle \in L$.

Solution 2: The following TM also decides the language, and has a different style. Basically, it starts by looking into the transition function of the input TM in some detail to understand how it works (the first solution doesn't do this at all).

Algorithm 7 Decider for L

Input: $\langle M, w \rangle$ where M is a TM and w is a string

1: (Note: assume that we first checked if the input string is a valid encoding, and rejected if it was not one)
2: Determine the states in M that result in rightwards loops consisting of transitions of the form $B \to \pi, R$, for some tape symbol $\pi \in \Gamma$. Do so by computing, for each state $q \in Q$, the set of states S_q that q can reach solely through $B \to \pi, R$ transitions. Then check if there exists a pair of states (q, s) that are reachable by each other, that is $s \in S_q$ and $q \in S_s$.
3: Let t be the variable containing the configuration of M, and initialize it to $q_0 w$ where q_0 is the start state of M.
4: For steps $i = 1, 2, 3, \ldots$:

1. Simulate the ith step of running M on w by updating the current configuration t using the transition function of M.
2. If M moved left, *accept*.
3. If M transitioned to its accept or reject state, *reject*.
4. If M moved to a part of the tape on the right of the rightmost symbol of t and is in one of the looping states determined earlier, *reject*.

First, we verify that the algorithm in step 2 is actually computable in finite steps (determining if there are rightwards moving loops from reading blanks). Note that because the set of states Q and the tape alphabet Γ are finite, the number of steps to determine the reachable states S_q for a starting state $q \in Q$ is finite (this can be done with BFS/DFS/etc). Furthermore, checking if each pair of states (q, s) is in a loop is done in finite steps because $S_q, S_s \subset Q$ and there are finitely many pairs of states.

Now we prove that our constructed TM is indeed a decider for L.

If $\langle M, w \rangle \in L$, then M is a TM that will move its head left at least once on input w. Therefore there exists a step i such that M moves left. It follows that it's impossible for M to halt before step i. It's also impossible for the head of M to be on blank symbol section of the tape while also in one of the looping states before step i; otherwise, M would forever read blank symbols and only move rightwards. Thus the decider will simulate M on w until step i, after which it will detect that M moved left and accept $\langle M, w \rangle$.

If $\langle M, w \rangle \notin L$, then M only moves its head rightwards for the entirety of its computation on w. If M halts, it will have never moved left and accepted, so the TM will reject $\langle M, w \rangle$. If M does not halt, then it will eventually move rightwards onto solely blank cells. Since M only moves rightwards, eventually it will always read in blank symbols and move rightwards. Because there are finitely many states, eventually M must transition to the same state as it was previously in, meaning M will reach one of the looping states identified earlier and $\langle M, w \rangle$ will be rejected by the decider. □

b) $L = \{\langle M, w, q \rangle \mid M$ enters state q on input w (So, M is a TM, q a state of M and w a string)$\}$.

Solution.

L is undecidable.

Proof. It would suffice to show the Turing reduction $A_{TM} \leq_t L$ because we have shown in class that A_{TM} is undecidable. Assume that there exists a decider N for L. We can construct a decider D for A_{TM} as follows

Algorithm 8 A decider D for A_{TM}

Input: $\langle M, w \rangle$ where M is a TM and w is a string

1: (Note: technically the input string is not always a valid encoding, but we can fix something not in L and return it as output for this case. For the purpose of this course, you can always assume the input has the right encoding.)
2: Let q_{acc} be the accept state of M.
3: Run N on $\langle M, w, q_{acc} \rangle$.
4: If N accepted, *accept* $\langle M, w \rangle$.
5: If N rejected, *reject* $\langle M, w \rangle$.

We now show D is a decider for A_{TM}.

(1) If $\langle M, w \rangle \in A_{TM}$, then M accepts w. Therefore M must enter its accept state q_{acc} on input w, so $\langle M, q_{acc}, w \rangle \in L$. N will accept $\langle M, q_{acc}, w \rangle$ so D will accept $\langle M, w \rangle$.

(2) If $\langle M, w \rangle \notin A_{TM}$, then M rejects or loops on input w. It's impossible for M to enter its accept state q_{acc} on input w because otherwise M would accept w. Therefore N will reject $\langle M, q_{acc}, w \rangle$ and D will reject $\langle M, w \rangle$.

Thus D always halts, and it accepts exactly the strings $\langle M, w \rangle \in A_{TM}$. So D decides A_{TM}, meaning $A_{TM} \leq_t L$. We conclude that L is undecidable. □

5 Bonus Problem

We say that a Turing Machine M is *pugnacious* if it has all of the following properties:

- It accepts the string 111111111111 and the string 101010101010
- It rejects the string 000111000111 and the string 000000000000
- It loops (never accepts or rejects) on the string 010101010101 and the string 111000111000

Define the language L_{pug} by

$$L_{pug} := \{\langle M \rangle \mid \ M \text{ is a pugnacious Turing Machine}\}.$$

(a) (up to 20 additional points) Your goal in this problem is to give the code for a pugnacious Turing Machine using the following website: `https://turingmachinesimulator.com/`. Be sure to look at some examples and tutorials on that site to learn about the notation it uses.

If your Turing Machine is not Pugnacious, or if it doesn't compile on the website as a **1-tape** Turing Machine, then you will get 0 points. Otherwise, we call your submission "valid", and your score on this problem will be calculated as follows.

Let n be the number of students in the class who make a valid submission. Let k be the number of students in the class who make a valid submission that uses at least as many total lines of code as yours. Then your score will be

$$5 + \left\lceil \frac{15k}{n} \right\rceil .$$

In other words, you want to use as few lines of code as possible, and you will get more points if you use fewer lines of code than other students. Good luck!

(We will post instructions for submitting your code as a text file closer to the deadline. We will measure "total lines of code" as the number of lines in the file you submit.)

(b) (up to 3 points) In light of part a, it may be difficult for the course staff to grade part b, even if they used the website `https://turingmachinesimulator.com/` or an AI tool like ChatGPT. Explain what the apparent issue is (we are looking for a specific, technical answer here), and give a suggestion for how the course staff might be able to grade it anyway. Give 2 sentences max.

You are not allowed to use "IDK" on the bonus problems.

Solution.

(a) We will not give a solution for part (a), use your creativity! We will post the top student submission after the homework is due.

(b) L_{pug} is undecidable (this was an exercise!), so there is no way for us to check that your submissions are correct. There are a few ways that we could grade anyways:

- Any algorithm must fail on *some* input, but there might be algorithms that correctly decide whether *large sets* of TMs are in L_{pug} or not. We could cross our fingers and hope that everyone submits TMs that are in this set, and run that restricted algorithm.

  In particular, suppose we make the assumption at the outset that none of you will submit a TM represented in more that 1 000 characters. Then, we could make an algorithm that decides whether TMs with 1 000 characters or less are in L_{pug} (a truth table would do) and use that. This is extremely inefficient, of course.

  A more realistic idea is that we could assume that no one's TM runs for longer than five seconds before accepting or rejecting, if they halt. Then, we can just run your TMs on the test strings, and make sure they accept, reject, or hit the time limit as required.
- We could ask you to submit proofs that your TMs are pugnacious with your submissions. Then, we can check the correctness of your proofs easily.

High Level Description: You do not need to discuss the mechanics of an actual TM (tapes, states, the transition function). *Instead you need present your algorithm in pseudocode.* See chapter 5 of the book for examples. Make sure your algorithm is clear and well defined (e.g., what is the input to the program, what are the main subroutines/loops/steps; specify the data structure(s) being used (variables, arrays, etc). You can give a few sentences to describe the ideas behind your algorithm, **but we expect your algorithm to be pseudocode (do not give us a big paragraph of text of as your algorithm).**

You need to give 3 to 5 sentences proof of why your TM is correct.

- If you are showing that L is recognizable, you need to explain why your TM accepts every string in L. And why it doesn't accept (loop or rejects) any string not in L.
- If you are showing L is decidable, you need to explain why your TM accepts every string in L. And why it rejects any string not in L.