CSE 105: Homework #4

Due on Feb 29, 2024 at 23:59pm

Professor Minnes

Ray Tsai

A16848188

Problem 1

Our first example of a more complicated Turing machine was of a Turing machine that recognized the language $\{w\#w \mid w \in \{0,1\}^*\}$, which we know is not context-free. The language

$$\{0^n 1^n 2^n \mid n \geq 0\}$$

is also not context-free.

(a) (*Graded for correctness*) Give an implementation-level description of a Turing machine that recognizes this language.

Proof. Consider a Turing machine T with $\Sigma = \{0,1,2\}$ and $\Gamma = \{0,1,2,\square\}$. Here is an implementation-level description of T:

$T =$ "on input string w:

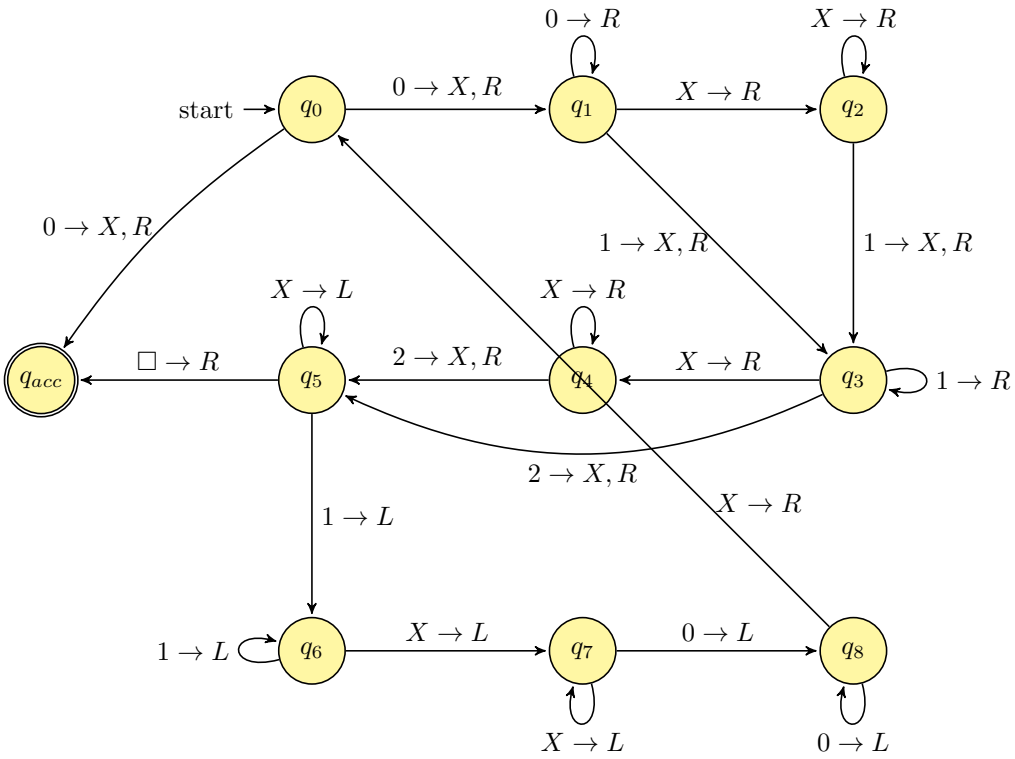
1. If first tape symbol was a blank, move right and accept. If first tape symbol was 0, print X and continue to the next stage. Otherwise, reject.
2. Scan right until reading a symbol that's not 0. If that symbol was X, continue scanning right until reading a tape symbol that is not X. Otherwise, proceed to the next stage.
3. Upon stopping, if the current tape symbol is not 1, reject. Otherwise, print X and scan right until scanning a symbol that is not 1. If that symbol was X, keep scanning right until scanning a symbol that is not X. Otherwise, proceed to the next stage.
4. After stopping, if current tape symbol is 2, print X and move right. Otherwise, reject.
5. If the current tape symbol is not a blank, proceed to the next step. Otherwise, scan all the way left to the next blank. If all symbols on the way were X's, move right and accept. Otherwise, reject.
6. Move all the way left until reading a blank. Then move right until scanning either $0, 1$, or 2, and go to stage 1."

□

Problem 1 continued on next page...

(b) (*Graded for completeness*) Draw a state diagram of the Turing machine you gave in part (a) and trace the computation of this Turing machine on the input 012. You may use all our usual conventions for state diagrams of Turing machines (we do not include the node for the reject state q_{rej} and any missing transitions in the state diagram have value $(q_{rej}, \square, R)$; $b \to R$ label means $b \to b, R$).

Proof. Here is the diagram of T:



	$q_0 \downarrow$			
$\square$	0	1	2	$\square$
		$q_1 \downarrow$		
$\square$	X	1	2	$\square$
			$q_3 \downarrow$	
$\square$	X	X	2	$\square$
				$q_5 \downarrow$
$\square$	X	X	X	$\square$
			$q_8 \downarrow$	
$\square$	X	X	X	$\square$
		$q_8 \downarrow$		
$\square$	X	X	X	$\square$
	$q_8 \downarrow$			
$\square$	X	X	X	$\square$
$q_8 \downarrow$				
$\square$	X	X	X	$\square$
	$q_{acc} \downarrow$			
$\square$	X	X	X	$\square$

$\square$

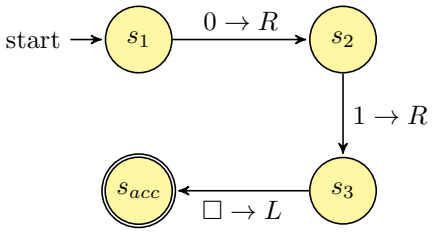
Problem 2

For this question, consider the alphabet $\Sigma = \{0, 1\}$.

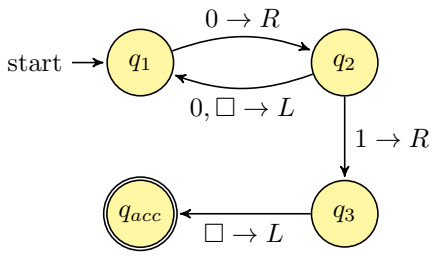
(a) (*Graded for correctness*) Give an example of a finite, nonempty language over Σ and two different Turing machines that recognize it: one that is a decider and one that is not. A complete solution will include a precise definition for your example language, along with **both** a state diagram and an implementation-level description of each Turing machines, along with a brief explanation of why each of them recognizes the language and why one is a decider and there other is not.

Proof. Consider the language $L = \{01\}$. Let $\Gamma = \{0, 1, \square\}$. Turing machines M_1, M_2 are defined as below:

State diagram for M_1



State diagram for M_2



M_1 = "On input string w:

1. If the first tape symbol was 0, move right. Otherwise, reject.
2. If the next symbol is 1, move right. Otherwise, reject.
3. If the next symbol is blank, move left and accept. Otherwise, reject."

M_2 = "On input string w:

1. If the first tape symbol was 0, move right. Otherwise, reject.
2. If the second tape symbol was 1, move right and accept. Otherwise, move left and go to stage 1.
3. If the next symbol is blank, move left and accept. Otherwise, reject."

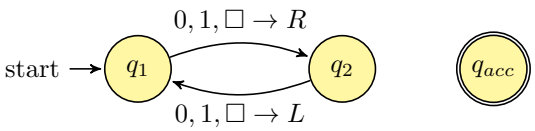
Since M_1 simply reads all the way right to check if the sequence is exactly 01, M_1 recognizes L and is a decider. For M_2, if the input string $w = 01$, it would get sent to the accept state like in M_1. However, if $w \neq 01$, it would either get rejected at q_1 or q_3, or get stuck in the loop of q_1 and q_2 forever. To see this, we may consider the string 0. 0 gets sent to q_2 sent back to q_1 as the second symbol is a blank. But then the machine head also goes back to the start of w after reading the blank. It follows that the process restarts and loops forever, and thus M_2 is not a decider. $\square$

Problem 2 continued on next page...

(b) (*Graded for correctness*) True or false: There is a Turing machine that is not a decider that recognizes the empty set. A complete solution will include a witness Turing machine (given by state diagram or implementation-level description or high-level description) and a justification for why it's not a decider and why it does not accept any strings, or a complete and correct justification for why there is no such Turing machine.

Proof. True. Consider T specified as below:

State diagram of T



Let $w \in \Sigma^*$. w gets sent to q_2 and sent back to q_1. But then the machine head also goes back to the starting point, and thus w ends up in a loop. Hence, T is not a decider but recognizes the empty set. $\square$

(c) (*Graded for correctness*) True or false: There is a Turing machine that is not a decider that recognizes the set of all string Σ^*. A complete solution will include a witness Turing machine (given by state diagram or implementation-level description or high-level description) and a justification for why it's not a decider and why it accept each string over $\{0, 1\}$, or a complete and correct justification for why there is no such Turing machine.

Proof. False. Note that Turing machines halt on acception. Since every string over Σ gets accepted, the machine would halt on any input. $\square$

Problem 3

Suppose M is a Turing machine over the alphabet $\{0,1\}$. Let $s_1, s_2, \ldots$ be a list of all strings in $\{0,1\}^*$ in string (shortlex) order. We define a new Turing machine by giving its high-level description as follows:

M_{new} = "On input w :

1.For $n = 1, 2, \ldots$

2. For $j = 1, 2, \ldots n$

3. For $k = 1, 2, \ldots, n$

4. Run the computation of M on $s_j w s_k$

5. If it accepts, accept.

6. If it rejects, go to the next iteration of the loop"

Recall the definitions we have: For languages L_1, L_2 over the alphabet $\Sigma = \{0,1\}$, we have the associated sets of strings

$$SUBSTRING(L_1) = \{w \in \Sigma^* \mid \text{there exist } a, b \in \Sigma^* \text{ such that } awb \in L_1\}$$

and

$$L_1 \circ L_2 = \{w \in \Sigma^* \mid w = uv \text{ for some strings } u \in L_1 \text{ and } v \in L_2\}$$

We say that self-set-wise concatenation of the set L_1 is $L_1 \circ L_1$.

(a) (*Graded for completeness*) Prove that this Turing machine construction **cannot** be used to prove that the class of decidable languages over $\{0,1\}$ is closed under **either** of the above operations ($SUBSTRING$ or self-set-wise concatenation). A complete answer will give a counterexample or general description why the construction doesn't work for both operations.

Proof. Consider a Turing machine M_1 that recognizes the language $L_1 = \{00\}$. and let M_1' be the resulting construction. Note that L_1 is finite, so it is Turing decidable. Let $w = 1$. Since for all $s_j, s_k \in \{0,1\}^*$, $s_j w s_k$ is never recognized by M, the loop M_1' never ends. It follows that M_1' does neither decides $SUBSTRING(L_1)$ nor $L_1 \circ L_1$. □

(b) (*Graded for correctness*) Prove that this Turing machine construction cannot be used to prove that the class of recognizable languages over $\{0,1\}$ is closed under the $SUBSTRING$ set operation. In particular, give a counterexample of a specific language L_1 and Turing machine M_1 recognizing it where M_{new} does not recognize $SUBSTRING(L_1)$.

Proof. Consider M_2 defined in problem 2 part (a). Let $w = 0$. We know $L(M_2) = \{01\}$, so w is in $SUBSTRING(L(M_2))$. However, since M_{new} enumerates the strings in shortlex order, 0 would get inputted to M before 01 does. Thus, 01 would not get read in finite time, as M loops on 0. It follows that w would not get accepted by M_{new}, and thus M_{new} does not recognize $SUBSTRING(L(M_2))$. □

Problem 3 continued on next page...

(c) (*Graded for completeness*) Define a new construction by slightly modifiying this one that can be used to prove that the class of recognizable languages over $\{0,1\}$ is closed under $SUBSTRING$. Justify that your construction works. The proof of correctness for the closure claim can be structured like: "Let L_1 be a recognizable language over $\{0,1\}$ and assume we are given a Turing machine M_1 so that $L(M_1) = L_1$. Consider the new Turing machine M_{new} defined above. We will show that $L(M_{new}) = SUBSTRING(L_1)$... *complete the proof by proving subset inclusion in two directions, by tracing the relevant Turing machine computations*"

Proof. Consider the construction

M_{new} = "On input w :

1.For $n = 1, 2, \ldots$

2. For $j = 1, 2, \ldots n$

3. For $k = 1, 2, \ldots, n$

4. Run the computation of M on $s_j w s_k$ for at most n steps

5. If it accepts, accept.

6. If it rejects, go to the next iteration of the loop"

Let L_1 be a recognizable language over $\{0,1\}$ and assume we are given a Turing machine M_1 so that $L(M_1) = L_1$. Consider the new Turing machine M_{new} defined above. We will show that $L(M_{new}) = SUBSTRING(L_1)$. Let $w \in L(M_{new})$. Since w is accepted by M_{new}, $s_j w s_k$ is accepted by M, for some s_j, s_k. Hence, $w \in SUBSTRING(L_1)$.

we now suppose $w \in SUBSTRING(L_1)$. Then, there exsits s_j, s_k such that $s_j w s_k \in L_1$. Since M_{new} only run each subloops in finite time, the subloops in M_{new} would eventually reach i, k, and thus it would eventually run $s_j w s_k$ on M, which would then get accepted. Hence, $w \in L(M_{new})$. □

Problem 4

Recall the definitions of some example computational problems from class

Acceptance problem		
... for DFA	A_{DFA}	$\{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$
... for NFA	A_{NFA}	$\{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$
... for regular expressions	A_{REX}	$\{\langle R, w\rangle \mid R \text{ is a regular expression that generates input string } w\}$
... for CFG	A_{CFG}	$\{\langle G, w\rangle \mid G \text{ is a context-free grammar that generates input string } w\}$
... for PDA	A_{PDA}	$\{\langle B, w\rangle \mid B \text{ is a PDA that accepts input string } w\}$
Language emptiness testing		
... for DFA	E_{DFA}	$\{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$
... for NFA	E_{NFA}	$\{\langle A\rangle \mid A \text{ is a NFA and } L(A) = \emptyset\}$
... for regular expressions	E_{REX}	$\{\langle R\rangle \mid R \text{ is a regular expression and } L(R) = \emptyset\}$
... for CFG	E_{CFG}	$\{\langle G\rangle \mid G \text{ is a context-free grammar and } L(G) = \emptyset\}$
... for PDA	E_{PDA}	$\{\langle A\rangle \mid A \text{ is a PDA and } L(A) = \emptyset\}$
Language equality testing		
... for DFA	EQ_{DFA}	$\{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$
... for NFA	EQ_{NFA}	$\{\langle A, B\rangle \mid A \text{ and } B \text{ are NFAs and } L(A) = L(B)\}$
... for regular expressions	EQ_{REX}	$\{\langle R, R'\rangle \mid R \text{ and } R' \text{ are regular expressions and } L(R) = L(R')\}$
... for CFG	EQ_{CFG}	$\{\langle G, G'\rangle \mid G \text{ and } G' \text{ are CFGs and } L(G) = L(G')\}$
... for PDA	EQ_{PDA}	$\{\langle A, B\rangle \mid A \text{ and } B \text{ are PDAs and } L(A) = L(B)\}$

(a) (*Graded for completeness*) Pick five of the computational problems above and give examples (preferably different from the ones we talked about in class) of strings that are in each of the corresponding languages. Remember to use the notation $\langle \cdots \rangle$ to denote the string encoding of relevant objects. *Extension, not for credit:* Explain why it's hard to write a specific string of 0s and 1s and make a claim about membership in one of these sets.

Proof. Let M_1 a DFA, M_2 be an NFA, R be a regular expression, G be a CFG, P be a PDA. Then, $\langle M_1, M_1\rangle \in EQ_{DFA}$, $\langle M_2, M_2\rangle \in EQ_{NFA}$, $\langle R, R\rangle \in EQ_{REX}$, $\langle G, G\rangle \in EQ_{CFG}$, and $\langle P, P\rangle \in EQ_{PDA}$. □

(b) (*Graded for completeness*) Computational problems can also be defined about Turing machines. Consider the two high-level descriptions of Turing machines below. Reverse-engineer them to define the computational problem that is being recognized, where $L(M_{DFA})$ is the language corresponding to this computational problem about DFA and $L(M_{TM})$ is the language corresponding to this computational problem about Turing machines. *Hint*: the computational problem is not acceptance, language emptiness, or language equality (but is related to one of them).

Let $s_1, s_2, \ldots$ be a list of all strings in $\{0,1\}^*$ in string (shortlex) order. Consider the following Turing

Problem 4 continued on next page...

machines

M_{DFA} = "On input $\langle D \rangle$ where D is a DFA :

1. for $i = 1, 2, 3, \ldots$
2. Run D on s_i
3. If it accepts, accept.
4. If it rejects, go to the next iteration of the loop"

and

M_{TM} = "On input $\langle T \rangle$ where T is a Turing machine :

1. for $i = 1, 2, 3, \ldots$
2. Run T for i steps on each input $s_1, s_2, \ldots, s_i$ in turn
3. If T has accepted any of these, accept.
4. Otherwise, go to the next iteration of the loop"

Proof. M_{DFA} recognizes any D such that $L(D)$ is nonempty, and thus the computational problem can be defined as $\overline{E_{DFA}}$, the complement of E_{DFA}.

On the other hand, M_{TM} recognizes Turing machine T such that $L(T) \cap S \neq \emptyset$, for a fixed language $S = \{s_i \mid i \in \mathbb{N}\}$. Hence, we may define the computation problem as $\{\langle T \rangle \mid T \text{ is a Turing machine and } L(T) \cap \{s_i \mid i \in \mathbb{N}\} \neq \emptyset\}$ □