HW4CSE105W24: Homework assignment 4 Solution

CSE105W24

March 5, 2024

1. **Classifying languages** (10 points): 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.

Solution:

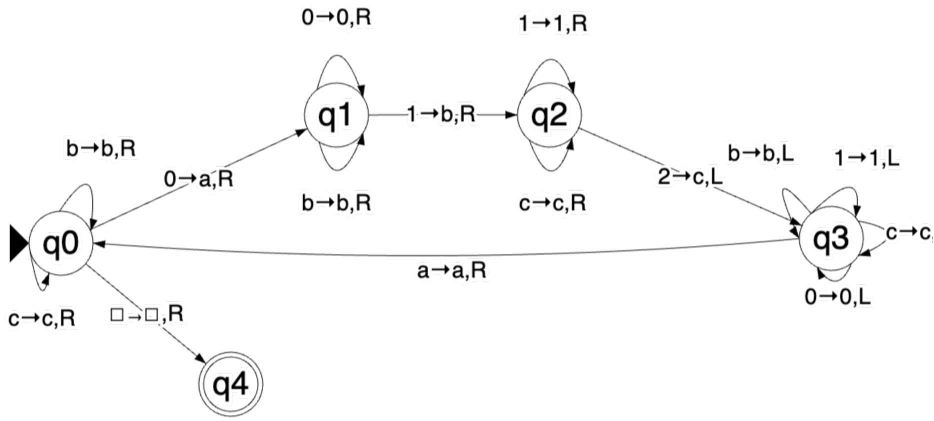
Implementation-level Description:

The idea is to cross off a 0, a 1, and a 2 in every loop, and accept if nothing goes wrong. Otherwise, reject.

Tapehead starts out at the first cell of the tape. If the first cell is blank (tape is blank), accept. If the first cell is a 1 or 2, reject. Otherwise, for the first 0 the tapehead reads, overwrite it with a. Keep moving right on the tape until a 1 is hit, then overwrite the 1 with b. Then keep moving right on the tape until a 2 is hit, then overwrite it with an c. Then move left until you find the first a (the most recent 0 that we crossed off) and move right. Repeat the process until the starting tape are all a, b, c's and there are no more 0's,1's, **and** 2's. If this is satisfied, accept. Otherwise, reject.

(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 $qrej$ and any missing transitions in the state diagram have value $(qrej, \square, R)$; $b \to R$ label means $b \to b, R$).

Solution:



Trace on 012

$$q_0 012$$

$$a q_1 12$$

$$ab q_2 2$$

$$a q_3 bc$$

$$q_3 abc$$

$$a q_0 bc$$

$$ab q_0 c$$

$$abc q_0 \_$$

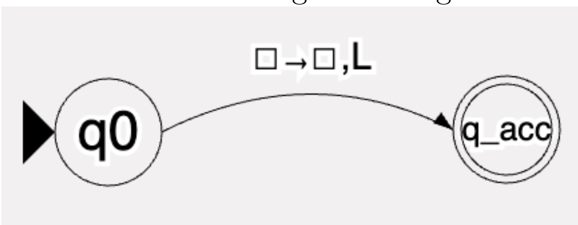
accept

2. **Deciders, Recognizers, Decidability, and Recognizability** (15 points): For this question, consider the alphabet $\Sigma = \{0, 1\}$.

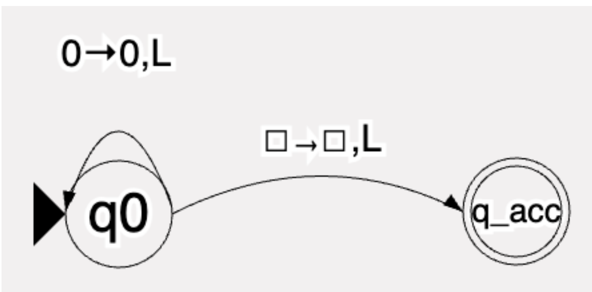
(a) (*Graded for correctness*) Give an example of a finite, nonempty language over Σ and two different Turing machines that recognize it: one that is a decider and one that is not. A complete solution will include a precise definition for your example language, along with **both** a state diagram and 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.

Solution:
Consider the language $L = \{\varepsilon\}$. It is finite, and nonempty, as its only element is ε. For the state diagrams below, recall the convention that if a transition is missing, it is implicitly going to the reject state.

Consider the following state diagram of a decider of L



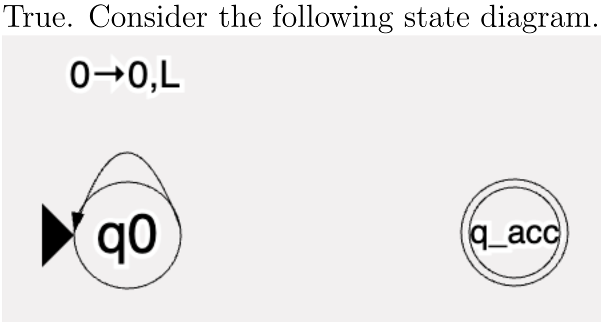
This machine checks the first character of a string. If it is a blank, the string must be the empty string ε, so we accept it and halt. Otherwise, we reject and halt.



This machine does essentially the same as above, except in one case. If the first character is 0, it replace it with a 0, and move to the left. This is the same as staying in the same cell. Thus, this machine will loop on any string that starts with a 0. So this machine recognizes the same language, but is not a decider, since it loops on certain inputs.

(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.

Solution:
True. Consider the following state diagram.



There is no transition to the accept state, so this machine will not accept any string, so it recognizes the empty set. Just like the machine above, this machine loops on any input string that starts with 0, so it is not a decider.

(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.

Solution:
False. In order to recognize the set of all strings Σ^*, the computation of the Turing Machine on each and every input must end in the accept state, where the computation halts. Thus, the computation on each string must eventually halt, so the Turing machine must be a decider.

3. **Closure** (15 points): 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 $L_1 \circ L_1$.

Note: there was a bug in the version of this assignment that was first released.

(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.

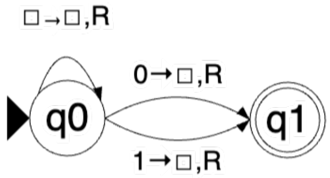
Solution:

For a Turing machine M where a string w is not in $SUBSTRING(L(M))$ or not in $L(M) \circ L(M)$, M never accepts $s_j w s_k$ for any s_j, s_k. The computation of M_{new} on w would not halt because each combination of $s_j w s_k$ is tried and there is no step at which M_{new} would reject. Since M_{new} is not a decider, it cannot witness that $SUBSTRING(L(M))$ or $L(M) \circ L(M)$ is decidable (even if it recognized the right language) and so cannot be used to prove that the class of decidable languages under $SUBSTRING$ or self-set-wise concatenation.

(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 $SUBSTRING(L_1)$.

Solution:

On line 4, we run M indefinitely until it accepts or rejects. So if somehow it never halts, M_{new} doesn't halt either. This is a problem if a later choice of s_j, s_k could witness membership in $SUBSTRING(L(M))$. In particular, let $L = \Sigma^* \setminus \{\varepsilon\}$ which is recognized by the Turing machine M_1 with state diagram



We will show that the constructed M_{new} does not recognized $SUBSTRING(L_1)$. We know that $\varepsilon \in SUBSTRING(L_1)$ since $0\varepsilon 0 \in L_1$. However, when ε is input to the machine, the first iteration of the nested loops has $j = k = 1$ and $s_1 = \varepsilon$ so in step 4 M_new glues a copy of ε on each end of ε and runs M_1 on $\varepsilon\varepsilon\varepsilon = \varepsilon$. This step never halt so we do not accept ε.

(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*"

Solution:

All we need to do is to limit the number of steps M can run. Change line 4 to

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

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 $L(M_{new}) = SUBSTRING(L_1)$

$\subseteq$ Let $w \in L(M_{new})$, i.e. M_{new} accepts the string w. The only way for M_{new} to accept is in step 5. Tracing the definition of M_{new}, arriving in step 5 means there exists some s_j, s_k such that $s_j w s_k$ is accepted by M. This means $s_j w s_k \in L(M_1)$, so $w \in SUBSTRING(L(M_1))$ by definition of $SUBSTRING$.

$\supseteq$ Consider arbitrary $w \in SUBSTRING(L(M))$. By definition of $SUBSTRING$, there exists a, b such that $awb \in L(M)$. Moreover, let's say a is the j^{th} string in string order and b is the k^{th} string in string order and the computation of M_1 on awb takes N steps to get to the accept state. When $n = \max(\{i, j, N\})$, the nested for loop in lines 2 and 3 of the definition of M_{new} has an iteration where $a = s_j$ and $b = s_k$ are considered and in step 4 feed awb to M_1 and get accepted after N steps. (We are guaranteed to get to this step because each prior iteration of the loop takes no more than $N \cdots N$ steps. Thus, M_{new} will accept w.

4. **Computational problems** (10 points): Recall the definitions of some example computational problems from class

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

Solution:

i. Let N be an NFA that accepts every string. $< N, \varepsilon > \in A_{NFA}$

ii. $< 101^*, 10111 > \in A_{REX}$

iii. $< (\{S\}, \{a, b\}, \{S \to S\}, S) > \in E_{CFG}$

iv. $< 101^*, 10 \cup 1011^* > \in EQ_{REG}$

v. Let P_1, P_2 both be PDA with empty language. $< P_1, P_2 > \in EQ_{PDA}$

(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 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"

Solution:

$L(M_{DFA})$ is the set of all DFA encodings such that the DFA has non-empty language. Suppose we have a non-empty DFA D, feeding $< D >$ to M_{DFA} would result in M_{DFA} trying all possible strings on D. Eventually, D will accept one, and M_{DFA} will also accept D. On the other hand, if the input is not the encoding of a DFA, M_{DFA} will reject. Finally, if the input is the encoding of a DFA but the language is empty, M_{DFA} will loop forever.

The same reasoning applies to M_{TM}. We do need to watch out for infinite loops when running a TM, hence the difference in construction compared to M_{DFA}.