Modern Complexity Theory Homework One

The aim of problem set is to help you to test your understanding of Turing Machines.

- **Collaboration:** You can collaborate with other students that are currently enrolled in this course in brainstorming and thinking through approaches to solutions but you should write the solutions on your own and cannot share them with other students.
- **Owning your solution:** Always make sure that you "own" your solutions to this other problem sets. That is, you should always first grapple with the problems on your own, and even if you participate in brainstorming sessions, make sure that you completely understand the ideas and details underlying the solution. This is in your interest as it ensures you have a solid understanding of the course material, and will help in the midterms and final. Getting 80% of the problem set questions right on your own will be much better to both your understanding than getting 100% of the questions through gathering hints from others without true understanding.
- **Serious violations:** Sharing questions or solutions with anyone outside this course, including posting on outside websites, is a violation of the honor code policy. Collaborating with anyone except students currently taking this course or using material from past years from this or other courses is a violation of the honor code policy.
- **Submission Format:** The submitted PDF should be typed and in the same format and pagination as ours. Please include the text of the problems and write **Solution X:** before your solution. Please mark in gradescope the pages where the solution to each question appears. Points will be deducted if you submit in a different format.

By writing my name here I affirm that I am aware of all policies and abided by them while working on this problem set:

Your name: Rishabh Singhal, rishabh.singhal@research.iiit.ac.in

Questions

For a non bonus question, you can always simply write **"I don't know"** and you will get 15 percent of the credit for this problem.

If you are stuck on this problem set, you can use this discussion board to send a private message to all instructors under the `e-office-hours` folder.

This problem set has a total of **50 points** and **5 bonus points**. A grade of 50 or more on this problem set is considered a perfect grade.

0.1 Short Problems

Following are short problems worth 4 points each.

Question 1.1: For a Turing Machine, can the input alphabet Σ be equal to the tape alphabet Γ?

Solution 1.1: No, this can't be true. As the tape alphabet contains blank alphabet to know when to stop, but consider the situation in which input alphabet contains the same blank alphabet then Turing Machine won't be able to recognize when the input ends.

Question 1.2: Can there be a Turing Machine that recognizes a non trivial language (i.e. not the empty language and not $\{0,1\}^*$) with a single state? Why/why not?

Solution 1.2: For recognizing a non trivial language there must be two outcomes for sure, i.e. either the given input is accepted or rejected. But to represent these two conditions, there should be at least two distinct states (rather than one as given in the question) for reject and accept. Or there will be only one state that is only rejecting or accepting which does not do any good. Hence, there shouldn't exist any Turing Machine that recognizes a non-trivial language with a **single state**.

Question 1.3: Consider a Turing Machine with 3 tapes such that it can read, write and move the head on all the tapes simultaneously. Write the formal specification (7-tuple) for such a Turing Machine.

Solution 1.3: Consider the Turing Machine (7-tuple) $(Q, \Gamma, B, \Sigma, \delta, q_0, F)$ where,

1. Q is a set of finite states
2. Γ is the tape alphabet
3. B is blank symbol
4. Σ is the input alphabet
5. $\delta : Q \times \Gamma^3 \to Q \times \Gamma^3 \times \{L, R, S\} \times \{L, R, S\} \times \{L, R, S\}$
6. q_0 is the initial state
7. F is the set of final states.

Question 1.4: Give an example of a language that can be recognized by a Turing Machine but not by a Context Free Grammar.

Solution 1.4: he language $L = \{b^n a^n b^n | n \geq 0\}$, it is recognized by Turing Machine but not by a Context Free Grammar. The given language is not a Context Free Grammar or can't be be recognized by a push-down automata because it only have one stack, let's say we push all the b's inside and then pop according to the occurrence of a but there is surely no way to check if later b's can counted. But surely, using a Turing Machine we can keep count using a state for each b, a and b and hence can verify if the lengths of all three segments i.e. of b, a and b are same.

Question 1.5: Consider a variant of a push-down automata where the stack is replaced by a FIFO queue. Does this change give it more power compared to usual push down automata in terms of the languages that can be recognized?

Solution 1.5: Yes, a push-down automata where stack is replaced by a FIFO queue have more power compared to usual one. As it can be considered same as a Turing Machine. Intuitively, in the FIFO version there is a way to rotate information so that any position can be reached which is now equivalent to the tape in Turing Machine. Consider a language $L = \{ww | w \in \Sigma^*\}$, this can be recognized by the FIFO variant (as by Turing Machine) but not by normal push-down automata.

0.2 Conceptual Problems

Question 2.1 (3+7 Points)

(a) Construct a single tape Turing machine that given a number as input computes its quotient with 2. Assume the input is present on the tape in binary.

(b) Construct a single tape Turing machine that reverses its input. Assume the input is present on the tape in binary (e.g., produces "0010111" from "1110100").

Solution 2.1:

(a) For the given program, the quotient of a number in binary with 2 will be all the bits except the last bit i.e. for output, the last bit of the input can be removed. So, consider the Turing Machine (Q, Γ, δ), such that

1. Q is the set of states which is $\{q_{start}, q_0, q_1, q_{halt}\}$
2. Γ is the tape alphabet which is $\{\triangleright, 0, 1, \square\}$ where $\square$ is blank symbol and $\triangleright$ is start symbol.
3. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$ is the transition function as follows:

Transition Function (δ)				
Q	Γ	Q	Γ	{L, R, S}
q_{start}	$\triangleright$	q_0	-	R
q_0	0	q_0	-	R
q_0	1	q_0	-	R
q_0	$\square$	q_1	-	L
q_1	0	q_{halt}	$\square$	S
q_1	1	q_{halt}	$\square$	S

Here initially the state transits into q_0 and it moves right till the end until blank symbol is hit, then it moves one step backward and removes the last bit to find the quotient. **Now, if the input is 1 or 0 then the output will be blank which represents 0 quotient.**

(b) For the given program, we can first directly copy the input in reverse order after the given input replacing the given input by X character then after everything is done X can be replaced by blank symbols. So consider the Turing Machine (Q, Γ, δ), such that

1. Q is the set of states which is $\{q_{start}, q_0, q_1, q_2, q_3, q_{10}, q_{11}, q_{halt}\}$
2. Γ is the tape alphabet which is $\{\triangleright, 0, 1, X, \square\}$ where $\square$ is blank symbol and $\triangleright$ is start symbol.
3. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$ is the transition function as follows:

Transition Function (δ)				
Q	Γ	Q	Γ	{L, R, S}
q_{start}	$\triangleright$	q_0	-	R
q_0	0/1	q_0	-	R
q_0	$\square$	q_1	-	L
q_1	0	q_{10}	X	R
q_1	1	q_{11}	X	R
q_1	X	q_1	-	L
q_1	$\square$	q_3	-	R
q_{10}	0/1	q_{10}	-	R
q_{11}	0/1	q_{11}	-	R
q_{10}	$\square$	q_2	0	L
q_{11}	$\square$	q_2	1	L
q_2	0/1	q_2	-	L
q_2	X	q_1	-	L
q_3	X	q_3	$\square$	R
q_3	0/1	q_{halt}	-	S

Here, initially the state is in q_0 after start symbol, it moves into q_1 after it reaches the end of the input, moving into the state of copying input alphabet. q_1 when sees 0/1 moves to q_{10}/q_{11} and then moves till the blank character is seen to copy the alphabet. After copying it moves back "left" until it hits X and changes to q_1 again doing the same thing iteratively. And once, q_1 sees a blank symbol it moves to q_3 for replacing all X with blank and then terminating on reach first 0/1 alphabet. Thus, reversing the input string.

Question 2.2 (10 Points): Consider a Turing Machine with an infinite 2-D tape. The head can now move not only left and right but also up and down. The input is initially written to the right of the head position.

(a) Write a detailed formal specification for this Turing Machine. How will the transition function change? What is a configuration?

(b) Does the set of languages recognized by such a 2D Turing Machine differ from that of the 3-tape Turing machine mentioned in Question 1.3? If so, give an example of a language that is recognized by one but not the other. If not, why? Would there be some sort of trade off in such a case?

Solution 2.2:

(a) Consider the 2-D Turing Machine (7-tuple) $(Q, \Gamma, B, \Sigma, \delta, q_0, F)$ where,

1. Q is a set of finite states in which Turing Machine can be
2. Γ is the tape alphabet
3. B is blank symbol
4. Σ is the input alphabet
5. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, U, D, S\}$, transition function will also have an additional option of going up (U) and down (D)
6. q_0 is the initial state
7. F is the set of final states.

A configuration of such Turing Machine will be $(x, q, k) \in \Sigma_*^2 \times Q \times K$, where x denote a 2-D matrix of alphabet on the tape, q denotes the machine's current state and k denotes the position of the machine on the tape.

(b) The set of languages recognized by such a 2D Turing Machine won't differ from that of the three tape Turing Machine mentioned in Question 1.3 as both such Turing Machines can be reduced to a single tape Turing Machine.

Claim: 2D Turing Machine can be simulated by a single tape Turing Machine. Proof: As the Turing Machine at any point of time has to look only a finite portion of 2D tape, consider such a construction. All the rows (of 2D Turing Machine) are written on single tape TM with a separator (or "$" symbol) between each row. Now,

1. If there is a need to move **left or L**, we can directly move left on the single tape Turing Machine as the row is represented simply as a contiguous block. When the machine is at the bottom-right most corner then there won't be any movement as it will be a wall hit.
2. Similar to left, if there is a need to move **right or R**, the pointer head can be moved right. When the machine is at the up-left most corner then there won't be any movement as it will be a wall hit.
3. If there is need to move **down or D**, it can be done by moving right until a separator (or "$" symbol) is not hit, and then moving the exact length as the starting position was from it's first left separator (then we will reach down cell). Also, if the current row is the last row then there won't be any move as it will be a wall hit.
4. If there is need to move **up or U**, similar to down it can be done by moving left until a separator (or "$" symbol) is not hit, and then moving the exact length as the starting position was from it's first right separator (then we will reach up cell). Also, if the current row is the first row then there won't be any move as it will be a wall hit.

So, the 2D Turing Machine can be simulated by one-tape TM but there will be some trade-offs, the amount of computation as done by one-tape TM will be more as compared to 2D TM, and hence 2D TM will be faster (approx. $O(RowSize)$ movement of tape pointer will be more) or the 2D TM will be faster by a constant factor in best-case, but there will be an additional need of (space) 2D tape.

Question 2.3 (10 Points):

(a) What is a language that is recursively enumerable but not recursive?

(b) What is a language that is recursive but not recursively enumerable?

(c) Read about Enumerators from Pg. 180 of "Introduction to the Theory of Computation, 3rd Edition by Michael Sipser". Read and understand the proof of Theorem 3.21 on the following page. (Write "I certify that I fully read the section on Enumerators and the proof of Theorem 3.21".)

(d) Using your understanding of Enumerators and Theorem 3.21 from the previous question, show that a language is recursive if and only if there is an Enumerator that enumerates the strings of the language in a length increasing fashion.

Solution 2.3:

(a) A language that is recursively enumerable but not recursive does not have a guarantee to halt if the given input (to the Turing Machine) is not in the language. That is it will surely halt with 1 if the given input is in the language, otherwise it will halt with 0 or will not halt forever. For e.g. consider a universal language $L_u = \{(M, w)|M$ is an encoding of a Turing machine and w is an encoding of a binary string accepted by M $\}$. Here L_u can be accepted by a Turing Machine but still undecidable. It is recursively enumerable but not recursive.

(b) A language that is recursive but not recursively enumerable will surely halt even if the input (to the Turing Machine) is not in the language that is if the given input is in the language it will halt with 1, while if the input is not in the language it will halt with 0 and won't go on without halting (unlike recursively enumerable languages).

(c) I certify that I fully read the section on Enumerators and the proof of Theorem 3.21.

(d) If there is an Enumerator that enumerates the string of the language in a length increasing fashion, then let's say we have to check if some input is in the language or not. Let the Enumerator enumerate till the length of the string which is being enumerated is greater than the input string. If the input string occurred before then halt with output 1 (accept) i.e. the given input is in the language otherwise, halt with output 0 (reject) as there was no string of given length in the list enumerated by the Enumerator as it is enumerating in order of increasing length. Therefore, whatever string is enumerated from now will be greater length than the input and hence they won't be equal to the input surely. Hence, as the language is able to halt in either accept or reject for every possible input, the given language is **recursive.**

Now in the other direction, if a Turing Machine is recursive and recognizes a language A, we can construct the following Enumerator for A. Say that $s_1, s_2, s_3, ...$ is the list of all possible strings in Language.

Then E = for i = 1, 2, 3,.... if the string $s_1, s_2, ...$ is of length i then output i.

In such a way, if s_1 exists in the recursive language it will eventually appear on the output and hence can be decided, otherwise the length of the string will be greater and TM can reject the given input.

Bonus Problem (5 Points):

Construct a single tape Turing machine that adds two numbers written in binary. (Assume that the numbers are separated by a special symbol "+" that belongs to the external alphabet of the TM.)

Bonus Solution: For adding two binary numbers one approach can be incrementing first number by 1 and decrementing the second number by 1 till the second number becomes 0. Turing Machine following same algorithm can be constructed as follows:

1. Turing Machine, $M = (Q, \Gamma, \delta)$ a 3-tuple.
2. Q is the set of states which is $\{q_{start}, q_0, q_1, q_2, q_3, q_4, q_5, q_{halt}\}$
3. Γ is the tape alphabet which is $\{\triangleright, 0, 1, +, \square\}$ where $\square$ is blank symbol and $\triangleright$ is start symbol.
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$ is the transition function as follows:

Transition Function (δ)				
Q	Γ	Q	Γ	{L, R, S}
q_{start}	$\triangleright$	q_0	-	R
q_0	0	q_0	-	R
q_0	0/1	q_0	-	R
q_0	+	q_1	$\square$	R
q_1	0/1	q_1	-	R
q_1	$\square$	q_2	-	L
q_2	0	q_2	1	L
q_2	1	q_3	0	L
q_2	$\square$	q_5	-	R
q_3	0	q_3	-	L
q_3	1	q_3	-	L
q_3	$\square$	q_4	-	L
q_4	0	q_0	1	R
q_4	1	q_4	0	L
q_4	$\square$	q_0	1	R
q_5	1	q_5	$\square$	R
q_5	$\square$	q_{halt}	-	S

Here it is assumed that the two numbers are separated by "+", which for the convenience of adding the binary number is replaced by the blank symbol, also it is assumed that the tape have blank symbols before the first number and after the second numbers. So initially, the given TM traverse and moves at the right-end and transitions into state 2, after which it subtracts 1 from second number and adds 1 to the first number. Subtracting by taking changing 0s to 1s and changing first 1 (from right) to 0. And similarly adding 1 to the first number.

This goes on until the second number becomes 0. In the given Turing Machine the second number is changed to 0s and then to blank symbols.