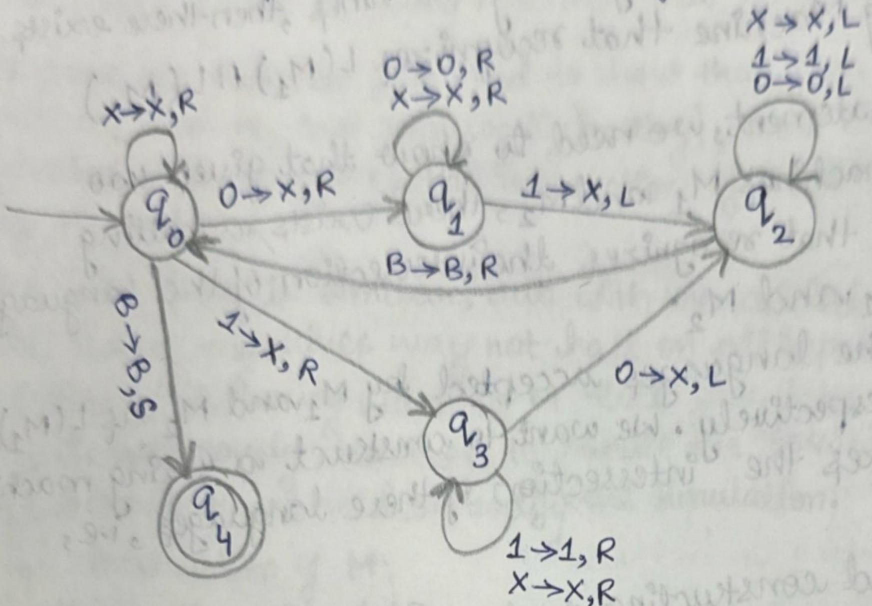
Theory of Computation
Homework-3

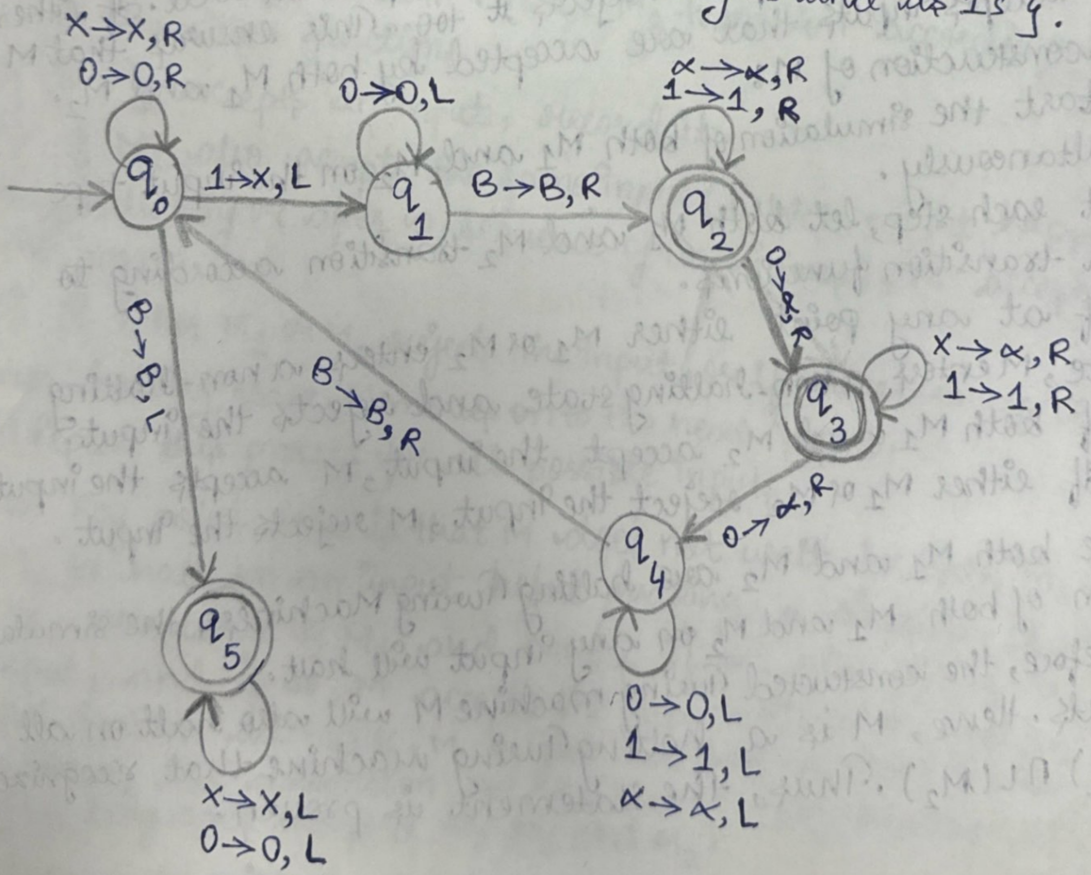
Vadlamudi Manogna
A20551908

1)

a) {W | W contains an equal number of 0's and 1's }.



b) {W | W does not contain twice as many 0's and as 1's }.



2)

a) If M_1 and M_2 are two halting Turing machines, then there exists a halting Turing Machine that recognizes $L(M_1) \cap L(M_2)$

To prove this statement, we need to show that given two halting Turing machines M_1 and M_2, there exists a halting Turing machine that recognizes the intersection of the languages accepted by M_1 and M_2.
let's denote the languages accepted by M_1 and M_2 as $L(M_1)$ and $L(M_2)$, respectively. We want to construct a Turing machine M that recognizes the intersection of these languages, i.e, $L(M_1) \cap L(M_2)$.
The idea behind constructing such a Turing machine is to simulate both M_1 and M_2 simultaneously. Whenever either M_1 or M_2 accepts an input, M accepts the input as well. If either M_1 or M_2 rejects the input. M rejects it too. This ensures that M only accepts inputs that are accepted by both M_1 and M_2.
The construction of M:
1) Start the simulation of both M_1 and M_2 on the input tape simultaneously.
2) At each step, let both M_1 and M_2 transition according to their transition functions.
3) If at any point either M_1 or M_2 enters a non-halting state, M enters a non-halting state and rejects the input.
4) If both M_1 and M_2 accept the input, M accepts the input.
5) If either M_1 or M_2 reject the input, M rejects the input.
Since both M_1 and M_2 are halting Turing Machines, the simulation of both M_1 and M_2 on any input will halt.
Therefore, the constructed Turing machine M will also halt on all inputs. Hence, M is a halting Turing machine that recognizes $L(M_1) \cap L(M_2)$. Thus, the statement is proven.

2)
b) If M_1 and M_2 are two (not necessarily halting) Turing machines, then there exists a Turing machine that recognizes $L(M_1) \cap L(M_2)$
To prove this statement, we need to show that given two Turing machines M_1 and M_2 (not necessarily halting), there exists a Turing machine that recognizes the intersection of the languages accepted by M_1 and M_2.
The idea here is similar, but with the added complexity that the Turing machines may not halt on all inputs. We can construct a Turing machine M that simulates both M_1 and M_2 simultaneously. However, to handle the possibility of non-halting behaviour, we need to modify the simulation.
The construction of M:
1) Start the simulation of both M_1 and M_2 on the input tape simultaneously.
2) At each step, let both M_1 and M_2 transition according to their transition functions.
3) If M_1 accepts an input, record it.
4) If M_2 also accepts the same input (which might happen immediately or after a sequence of transitions), M accepts the input.
5) If either M_1 or M_2 rejects the input, or if either enters a non-halting state, M moves on to the next input.
6) Repeat this process for all possible inputs.
The key point here is that M does not wait for both M_1 and M_2 to halt on an input before making a decision. It records the acceptance of M_1 and checks if M_2 also accepts the same input, and if so, M accepts the input. Since M_1 and M_2 may not halt on all inputs, M might not halt on some inputs as well. However, this construction ensures that M recognizes the intersection of the languages accepted by M_1 and M_2.
Therefore, the existence of such a Turing machine M proves the statement.

3) Prove,

Double infinite tape turing machine that recognizes a class of languages is the same as the class of Turing recognizable languages.

To show this, It has to satisfy two things:

→ Any language L that can be recognized by M can be recognized by D (M is an ordinary turing machine, D is a double infinite tape turing machine)

→ Here any language L that can be recognized by D i.e, a double infinite tape turing machine, can be recognized by M i.e an ordinary turing machine.

This can be shown by simulation -

• Simulating M by D: So, here first we have to mark left hand end of D then prevent D from moving its head to the left of the mark. By this way D simulates M.

• Simulating D by M: To simulate Doubly infinite tape D by an ordinary Turing machine M, Simulate it with a 2-tape Turing Machine.

Here 2 tape Turing Machine is already equivalent in power to an ordinary Turing Machine.

Simulation of doubly infinite tape Turing machine with 2-tape Turing machine 'M_1'.

Here first tape of M_1 is written with input string and the second tape is blank.

Now, cut the tape of double infinite tape Turing Machine into two parts, at the starting cell of the input string.

One part containing input string all the blank spaces to the right, the second part containing left of the input string appears on the second tape, in Reverse order.

In this way Turing machine simulates double infinite tape turing machine.

Thus, from both the simulation, we can say double infinte tape machine is equivalent to turing machine i.e, ~~[illegible]~~ class of languages recognized by TDMITs is the same as the class of Turing recognizable languages.

4) To prove,

A language is decidable if and only if some enumerator prints the strings in the language in lexicographical order.
We have to prove in both directions:
→ language is decidable then enumerator enumerates the language in lexicographic order.
→ Enumerator enumerates the language in lexicographic order and then it is decidable.

• Proof:-
language is decidable then enumerator enumerates the language in lexicographic order.

Let us assume that we have a turing machine M to decide a language L.
Now we can use this M to construct an enumerator E as follows.
We generate the strings in the lexicographic order and input each string into M for L.
If M accepts then print that string. Therefore E prints all strings of L in lexicographic order.

• Proof:-
Enumerator enumerates the language in lexicographic order and then it is decidable.

Now, we need to consider the other direction.
That is, if we have an enumerator E for a language L, then we can use E to construct a Turing machine M that decides L.
They are 2 cases which we can consider.
case 1-
If L is finite language, then it is decidable, because all the finite languages are decidable.

case 2-
If L is infinite then a decider for L operates as follows.

- On receiving the input w, the decider enumerates all strings of L in lexicographic order until a string greater than w appears in the lexicographic order.
- This must eventually occur since L is infinite
- If w has appeared in the enumeration already, then accept.
- If w has not yet appeared in the enumeration then it will never appear and hence we can reject.

So in both cases L is decidable. The theorem proved in both directions.

∴ A language is decidable if and only if some enumerator prints (enumerates) the language in lexicographic order.