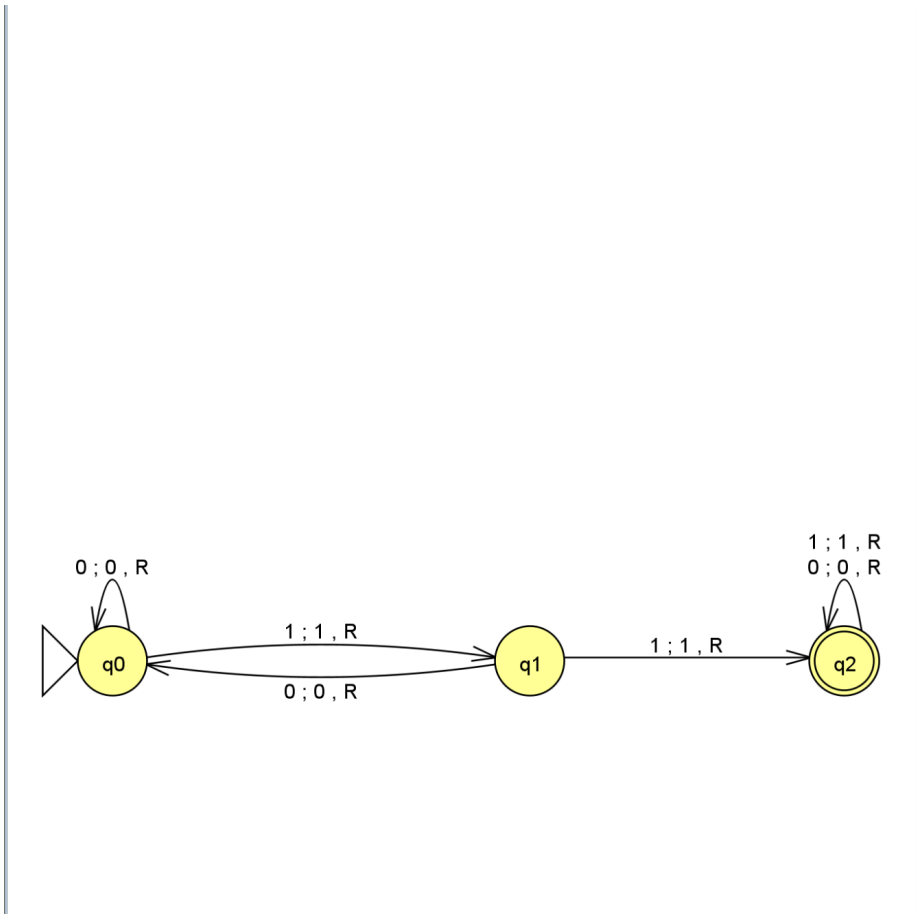
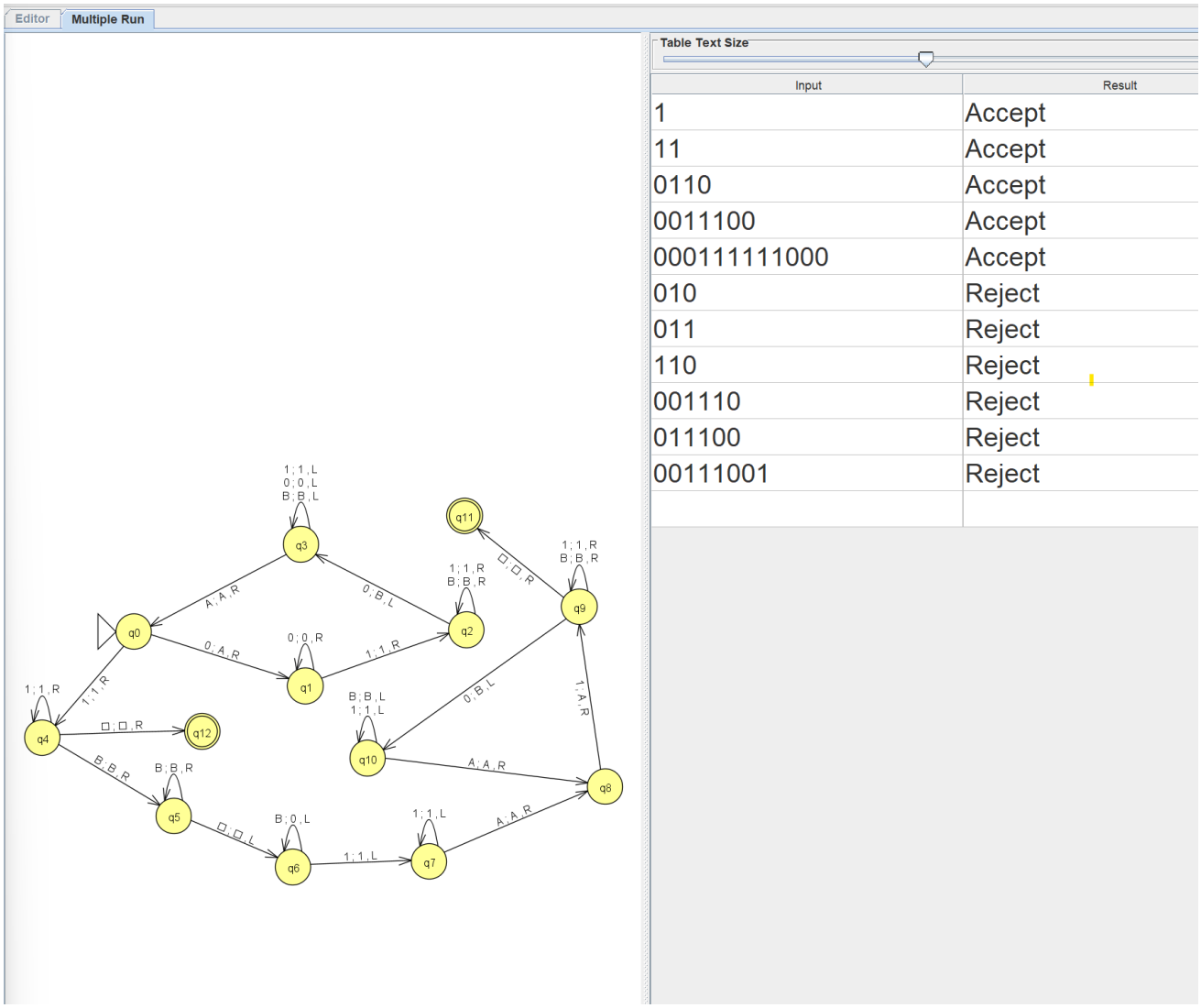
1. For each of the following languages, using JFLAP (https://www.jflap.org/jflaptmp/) to draw the state diagram of an ordinary Turing machine (deterministic, and not any of the variants discussed in class) that recognizes it, then run it with the given input strings. Screen copy the Turing machine's state diagram and the running results of these strings, paste it as your answer to the question. 10 points each

a) $L1$ = {All binary strings that contains two consecutive 1s}
Input strings (accepted): 11, 0110, 00111, 11110, 010110110,
Input strings (rejected): 1, 000, 0001, 101, 1010, 00101, 10101,




| Input | Result |
|---|---|
| 11 | Accept |
| 0110 | Accept |
| 00111 | Accept |
| 11110 | Accept |
| 010110110 | Accept |
| 1 | Reject |
| 000 | Reject |
| 0001 | Reject |
| 101 | Reject |
| 1010 | Reject |
| 00101 | Reject |
| 10101 | Reject |

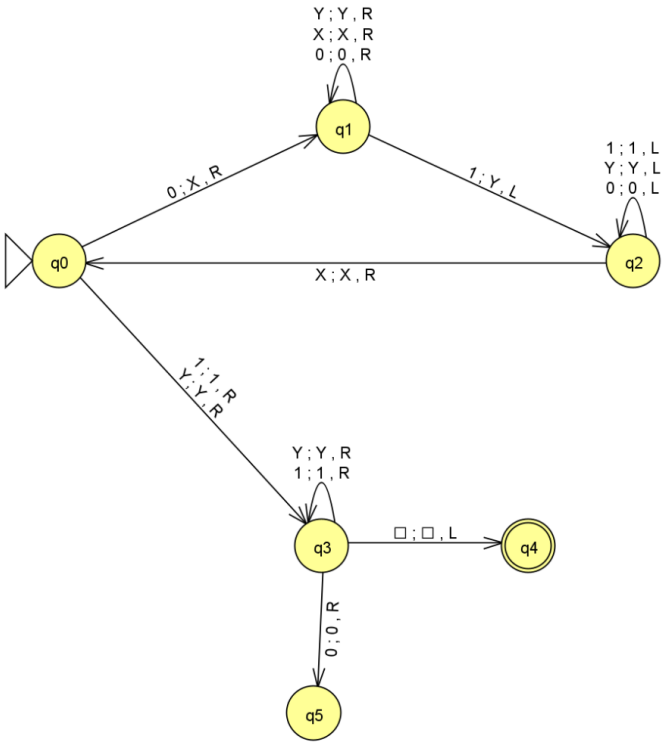
b) $L2$ = {0^$n$ 1^$m$ 0^$n$ | $m > n \geq 0$}
Input strings (accepted): 1, 11, 0110, 0011100, 000111111000,
Input strings (rejected): 010, 011, 110, 001110, 011100, 00111001,




c) $L3$ = {0^$i$ 1^$j$ | $j \geq i \geq 0$}

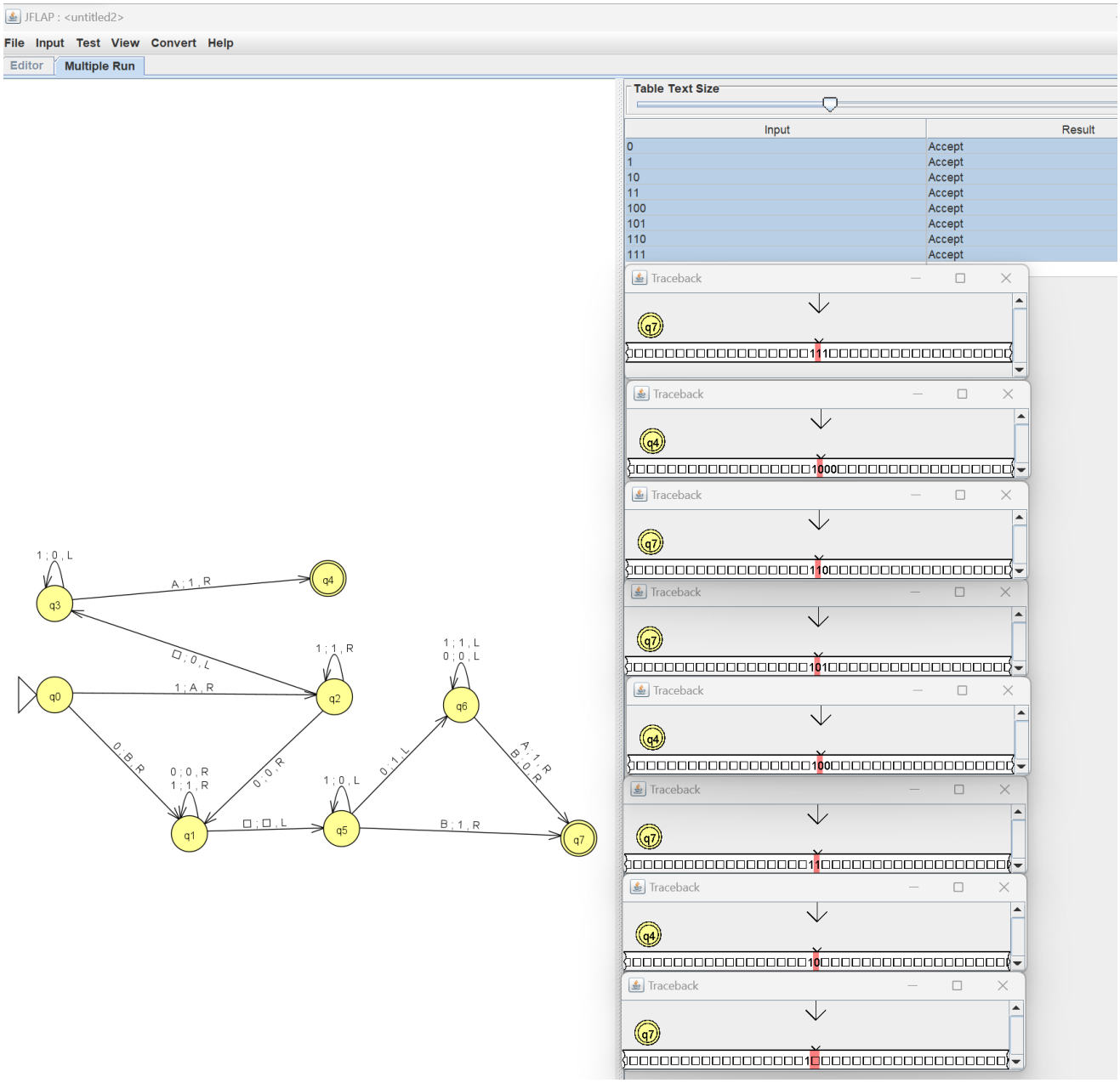
Input strings (accepted): 1, 11, 01, 011, 001111, 000001111111,

Input strings (rejected): 0, 10, 001, 010, 0110, 001, 0000111,




| Input | Result |
|---|---|
| 1 | Accept |
| 11 | Accept |
| 01 | Accept |
| 011 | Accept |
| 001111 | Accept |
| 000001111111 | Accept |
| 0 | Reject |
| 10 | Reject |
| 001 | Reject |
| 010 | Reject |
| 0110001 | Reject |
| 0000111 | Reject |

2. using JFLAP (https://www.jflap.org/jflaptmp/) to draw the state diagram of an ordinary Turing machine (deterministic, and not any of the variants discussed in class) that increments a binary number by 1. For example, if the input string is 11 (decimal number 3), then the output should be 100 (decimal number 4). Run the TM with the given input strings. Screen copy the Turing machine's state diagram and the running results of these strings ("Input"=>"Multiple Run (Transducer)". To screen copy the running result of the tape, pick each input, then press the button "View Trace. Paste it as your answer to the question. 10 points
Input strings: 0, 1, 10, 11, 100, 101, 110, 111



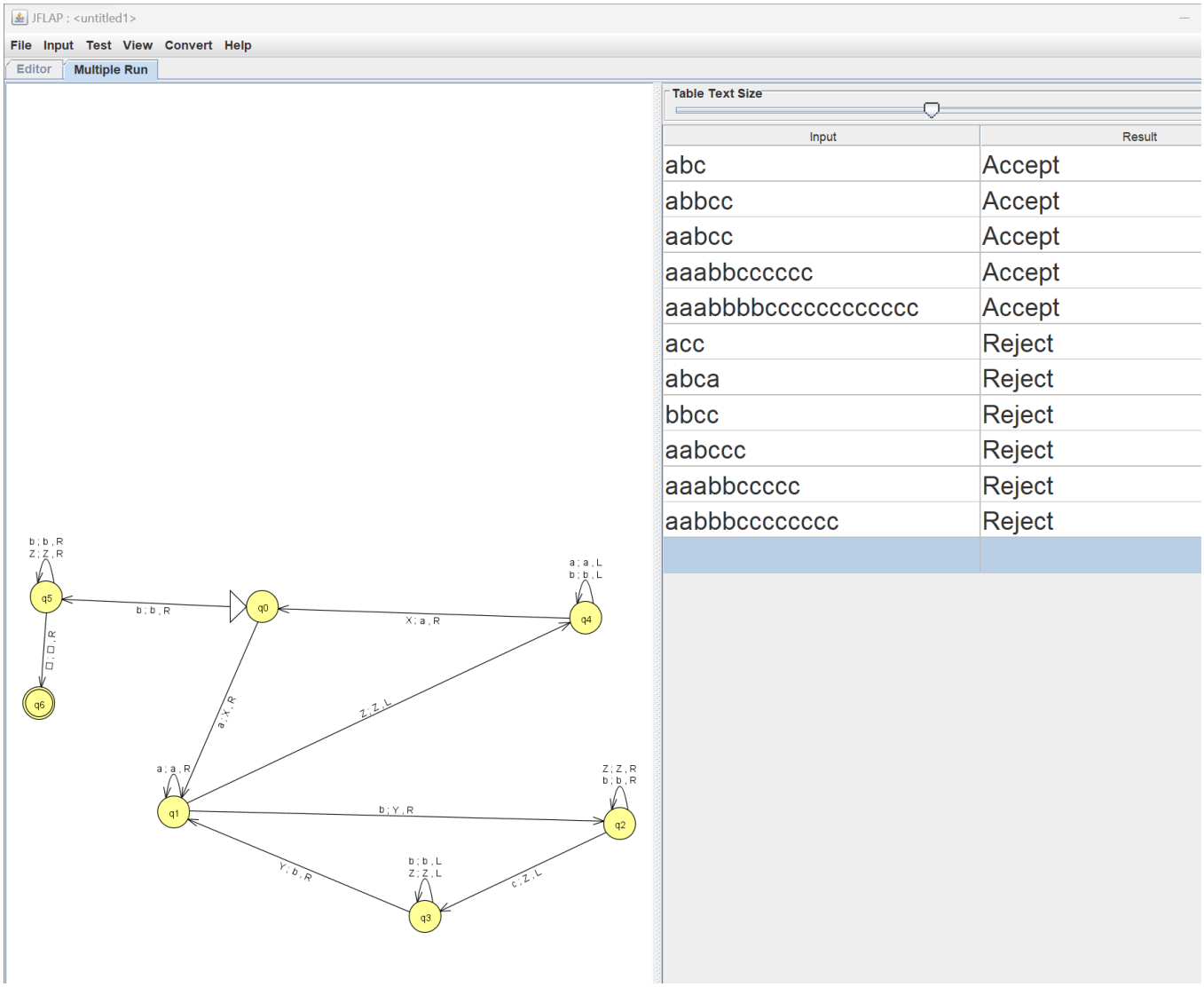

3. Using JFLAP (https://www.jflap.org/jflaptmp/) to draw the state diagram of a Turing machine that accepts the following language. Run the TM with the given input strings. Screen copy the Turing machine's state diagram and the running results. 20 points

$L3$ = {$a$^$i$ $b$^$j$ $c$^$k$ , $i \times j = k$ *and* $i, j, k \geq 1$}

Note: the implementation level description of TM that accepts $L3$ is given in Example 3.11 of the textbook pp.167. Use it as a reference.

Input strings (accepted): abc, *abbcc*, aabcc, *aaabbccccc*, *aaabbbbccccccccccc*,

Input strings (rejected): acc, abca, *bbcc*, aabccc, *aaabbccccc*, *aabbbccccccc*,

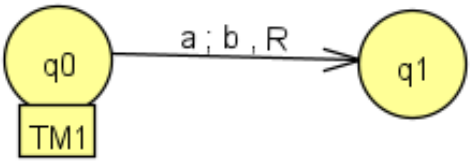


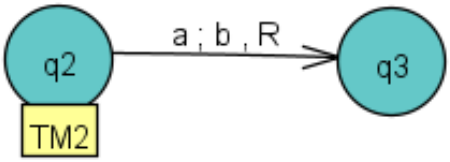

4. A Turing machine with double infinite tape is like an ordinary Turing machine, but its tape is infinite in both directions, to the left and to the right. Prove that Turing machine with double infinite tape is equivalent to the ordinary Turing machine. (Note: you can assume that the tape is initially filled with blanks except for the portion that contains the input) 10 points

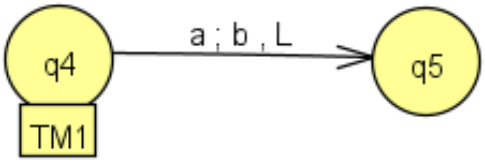
**TM1 = classic Turing machine**

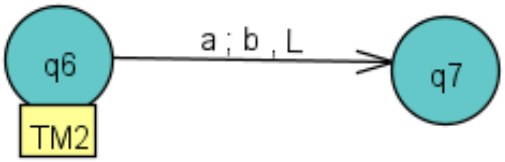
**TM2 = doubly infinite tape Turing machine**

**Use TM2 to simulate any transition from TM1**



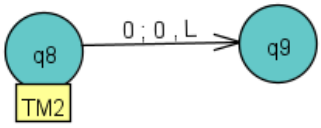



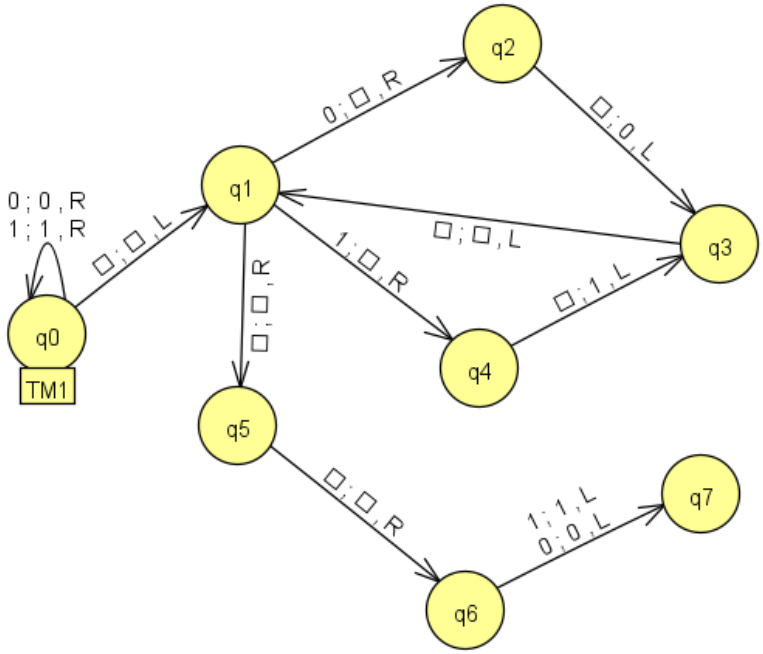




- **We can simulate moving L and R with both TM1 and TM2. We can also simulate adding a blank symbol to the right end of the tape in both TM1 and TM2**

**Use TM1 to simulate any transition from TM2**




- **If you are at the head of the string, TM2 will be able to move L and place a blank symbol at the beginning of the tape. TM1 will be able to do the same procedure, as shown above, by placing a blank symbol at the end of the tape shifting all the digits to the right by 1, and making the first symbol of the tape blank as well. Doing so would make both TM1 and TM2 have the same functionality**

**Both Turing machines have the equivalent because both can complete the same function and transitions.**

5. Prove that Turing-recognizable languages are closed under concatenation operation. (*Hint: prove by constructing a new TM). 10 points

**Consider that there are two languages that are called M and N and an input string 'w'. W will be considered part of the concatenation of M and N if it can be expressed like this w = mn, where 'm' is an element of M and 'n' is an element of N. This shows that the string can be split into two halves. The first half belongs to M and the second half belongs to N.**

**We can make a procedure for the concatenation of M and N using a 2-tape nondeterministic Turing machine. Initially, the machine non-deterministically guesses the point at which to partition the string into the m and n segments. Next, it executes the first tape to check that the first section (m) is in language M. If not, the machine rejects the input. After this, it does the same procedure to the second tape to find if the second segment (n) is in the language of N. If this is not the case, the machine rejects the input.**

**The last case would be if the string (w) was actually split in the correct location and is in the language of the concatenation of M and N, the non-deterministic would continuously guess the partition. In such instances, the corresponding branch of the computation will accept, making an overall acceptance of the input. This approach shows the deterministic nature of the machine to verify a given string in the concatenation of two decidable languages, M and N.**

6. Consider the problem of determining whether a DFA accepts strings containing an equal number of 0s and 1s. Convert this problem into a language, and then prove it is decidable. 20 points.

**We can create a regular language expression that expresses the above requirements. ADFA = {<K> | K is a DFA that accepts all strings containing an equal number of 0s and 1s}**

- **K = 'On the following input <J, w>, where J is a DFA and w is a input string'**
- **Simulate J on the input of w.**
- **The language where the number of 0s and 1s are equal is not regular, so a DFA can not be made for it. However, if we intersect (and) a regular language (RL) with a context-free language (CFL) the result will still be context-free (CF). For this problem, we can create a PDA P and a DFA K that recognizes the language detailed above: w contains an equal number of 1's and 0's. The RL of DFA K would be L(K) and the CFL for the PDA P would be L(P). If we intersect L(K) and L(P), the resulting language we can call L(G). L(G) is CF. If L(G) is not empty, then there is a common string between L(P) and L(K). If we use EDFA on L(G), if L(G) is empty then the DFA will be rejected and if it ends in an accepting state then the DFA will be accepted. This means that the language that accepts strings containing an equal number of 0's and 1's is decidable.**