Chapter 10

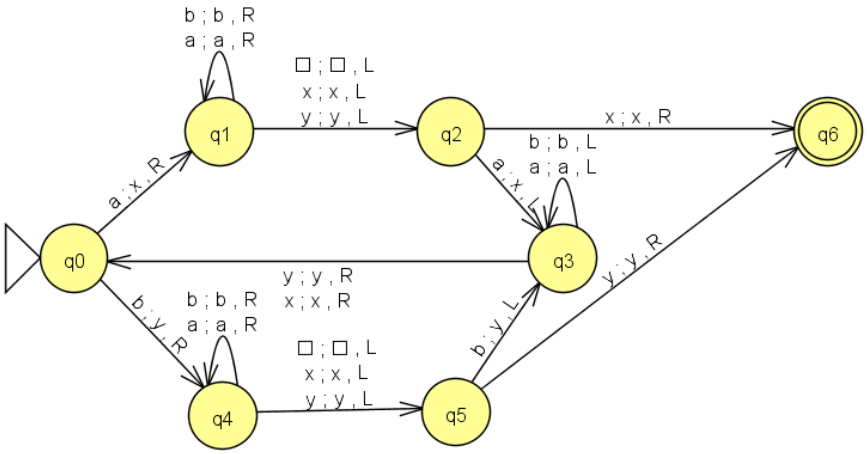
Turing Machines

A. Construct a Turing machine to perform each of the functions given below. Also show an accepting sequence of configurations for each of the following example strings.

Note: Accepting sequences of configurations can be obtained by running the jff files provided for each of the following Turing Machines in JFLAP and providing the specified input string in "Step" mode. Many of the traces are too long to be included here.

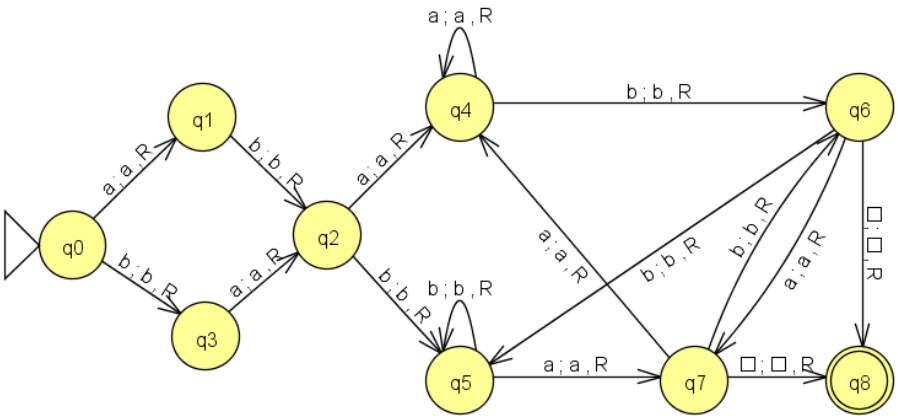
1. Accept all odd palindromes over {*a*, *b*} (where the symbol at the middle is also *a* or *b*). Show an accepting sequence of configurations for the input *baaaaab*.

   Solution: Please see also the VIDEO SOLUTION for this exercise.



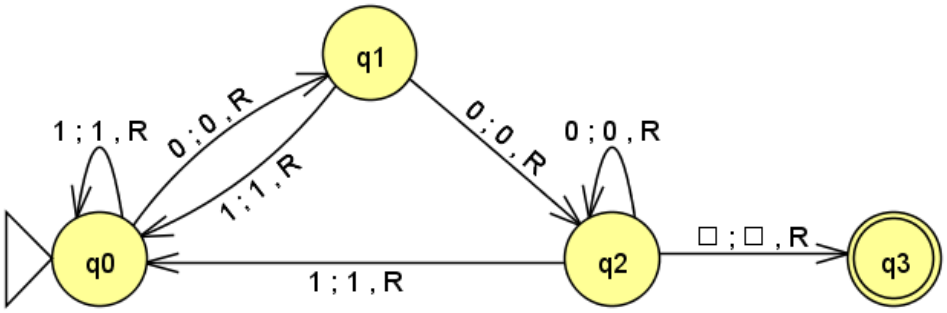
2. Accept the language of the RegEx: (*ab* + *ba*)(*a* + *b*)*(*ba* + *ab*). Show how it rejects *ababaa*.

   Solution:



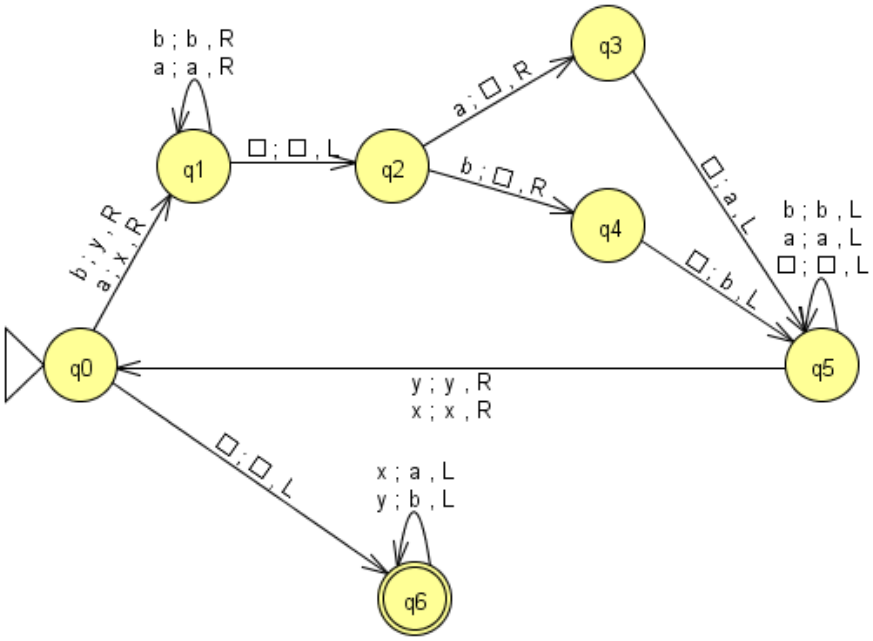
3. Accept all positive binary numbers divisible by 4.

   Solution:



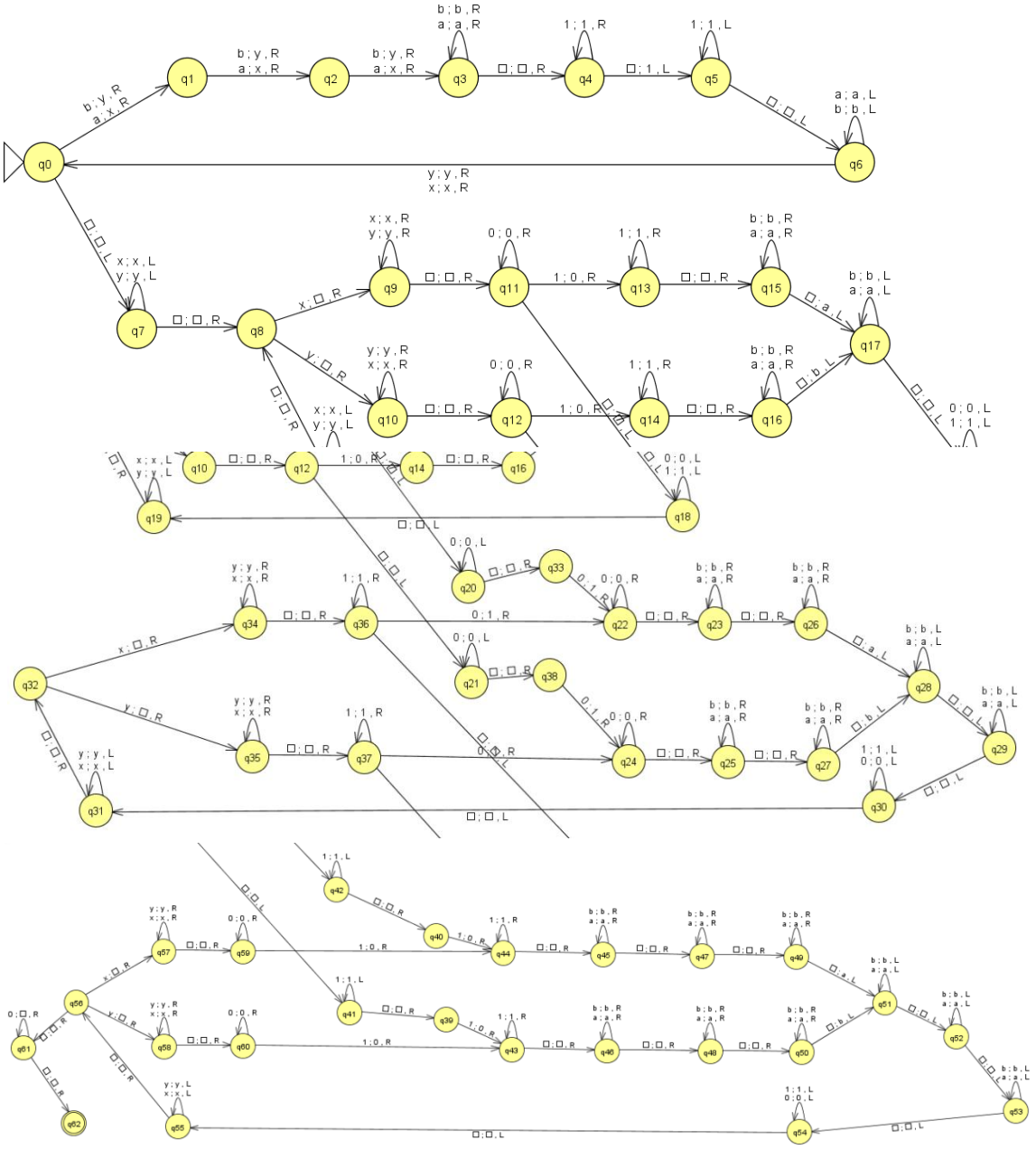
4. Separate a given string into two equal halves by finding the midpoint of the given string and inserting a blank at that point. For example, given *abbaab*, the output shall be *abb aab*.

   Solution: Please see also the VIDEO SOLUTION for this exercise.



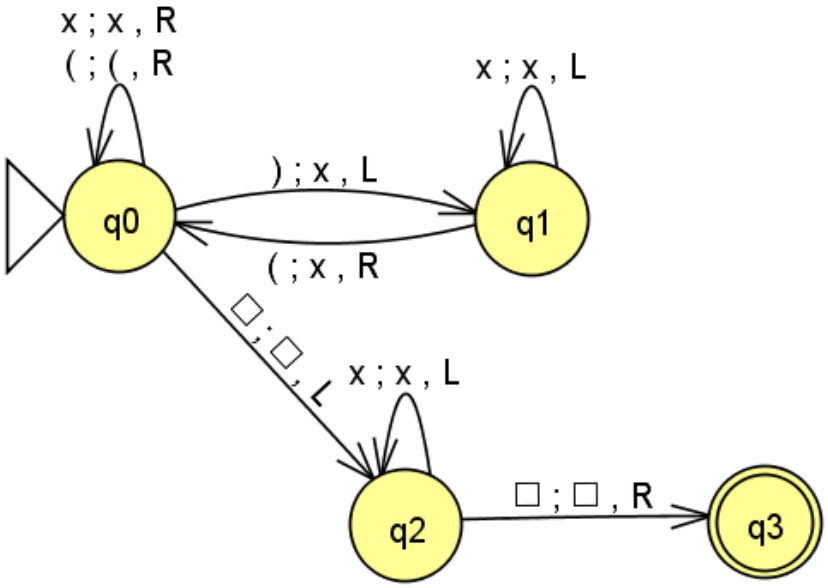
5. Break a given string into three equal parts. Show an accepting sequence of configurations for the input *abbbaabba*, the expected output being *abb baa bba*.

   Solution: Note that this Turing machine is too complex to show as a single readable diagram. Please load the accompanying jff file in JFLAP to see how this machine works,



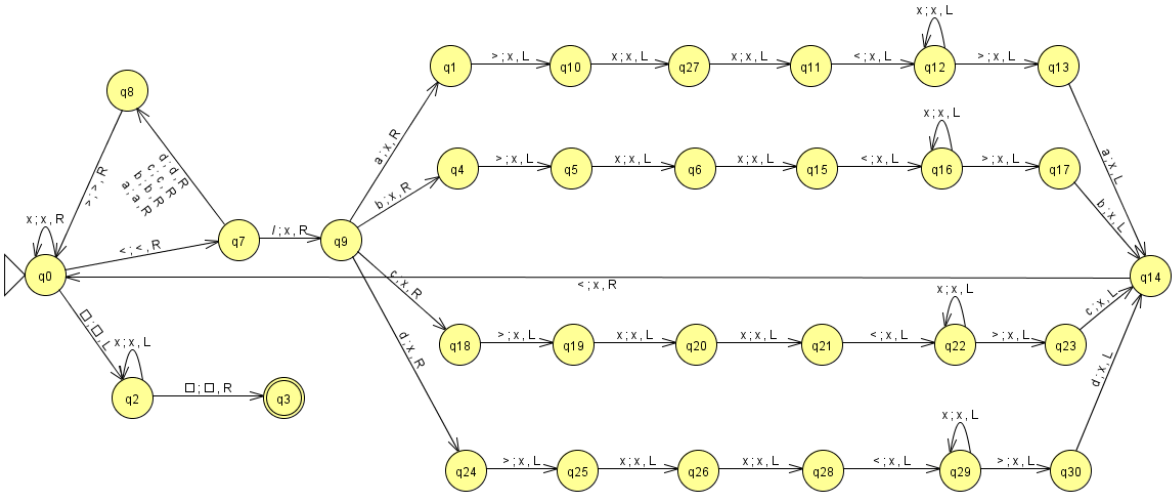
6. Match properly nested parentheses. Show an accepting sequence of configurations for (((()(()))())).

Solution:



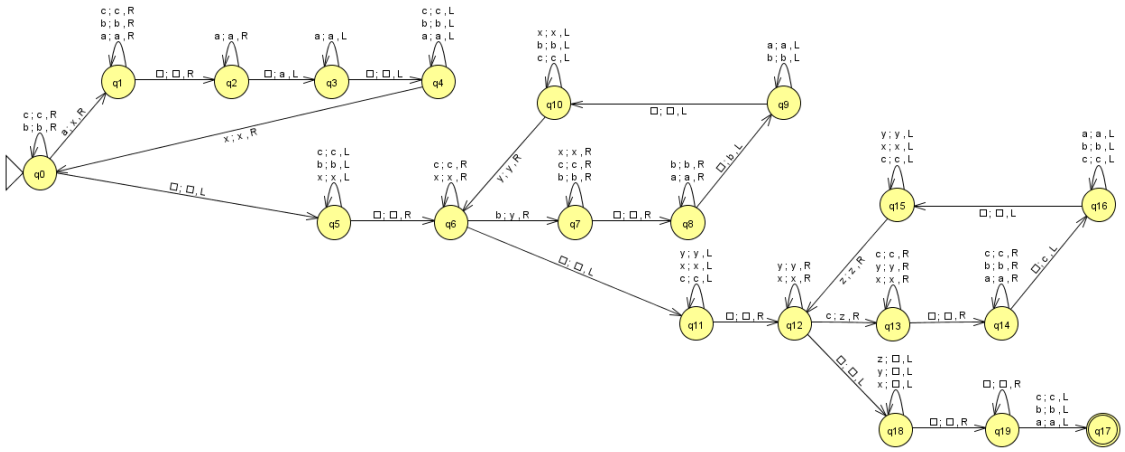
7. Match properly nested XML tags (as described in Exercise 71 in Chapter 7 and Exercise 27 in Chapter 8), assuming that the only possible tag names are {*a*, *b*, *c*, *d*}.

Solution:



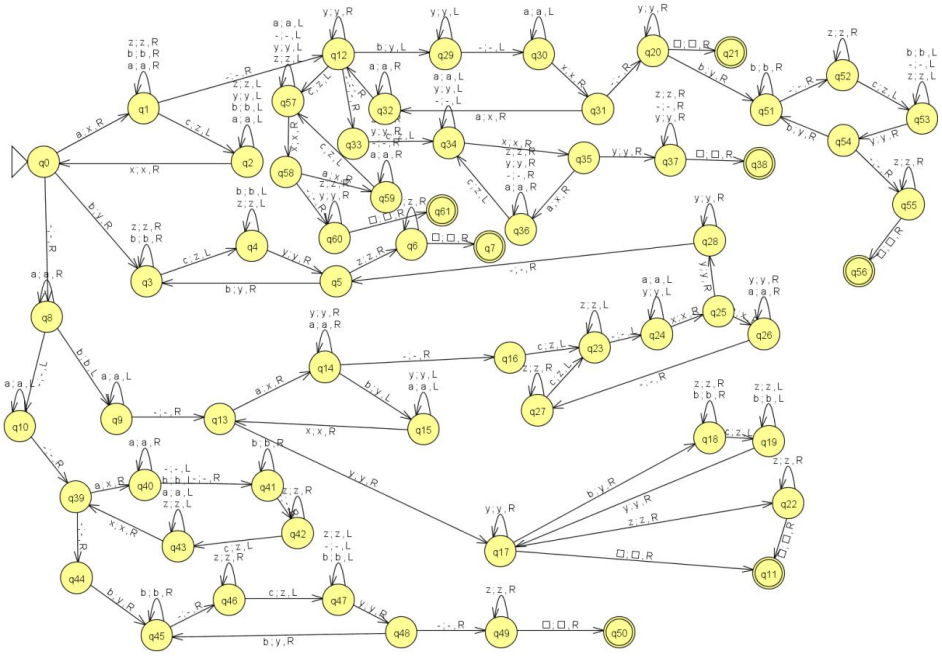
8. Sort the symbols in the input string: for example, given *babcacba*, the output shall be *aaabbbcc*.

Solution: Note: This Turing machine does not work for null input.



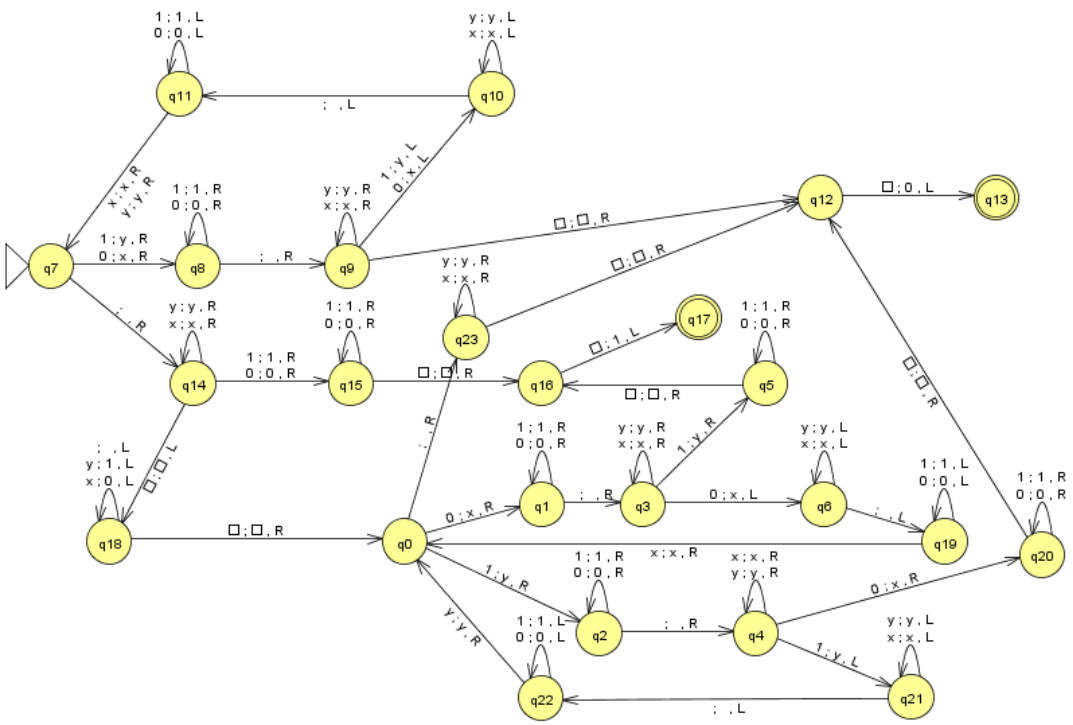
9. Enhance Example 10.2 by including addition with positive or negative numbers, that is, strings over the alphabet {*a*, *b*, *c*, -} of the form:
   a) $a^n b^m c^k$ where $k = n + m$ and both numbers are positive; for example, *aabbcccc*;
   b) $-a^n b^m c^k$ where $k = m - n$ if $m \geq n$ and the first number is negative;
      i. for example, *-aabbbbcc*;
   c) $-a^n b^m -c^k$ where $k = n - m$ if $n > m$ and the first number is negative;
      i. for example, *-aaaabb-cc*;
   d) $a^n -b^m c^k$ where $k = n - m$ if $n \geq m$ and the second number is negative;
      i. for example, *aaaa-bbcc*;
   e) $a^n -b^m -c^k$ where $k = m - n$ if $m > n$ and the second number is negative;
      i. for example, *aa-bbbb-cc*; or
   f) $-a^n -b^m -c^k$ where $k = n + m$ and both numbers are negative;
      i. for example, *-aaa-bbb-cccccc*.

Solution: It may be noted that the Turing machine shown here works well only when all three symbols are present. Also, it is possible to simplify it by re-using many of its states.



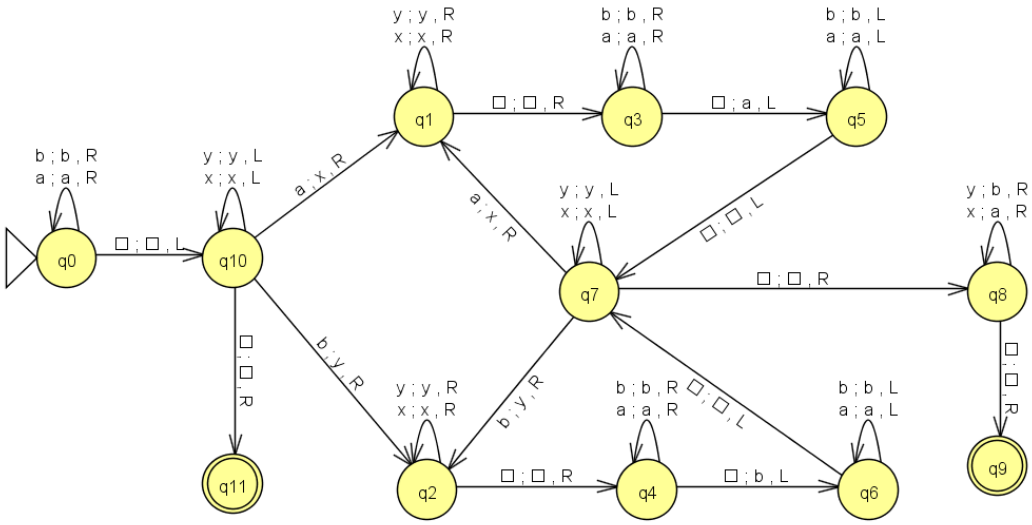
10. Find the maximum of two positive binary integers. For example, given 1001 11000 as input, the output is the second number, which is indicated by writing a 1 onto the tape: 1001 11000 1 (similarly a 0 if the first number is the maximum of the two or if the two numbers are the same).

Solution: Note: Blank is a character, not the tape blank symbol.



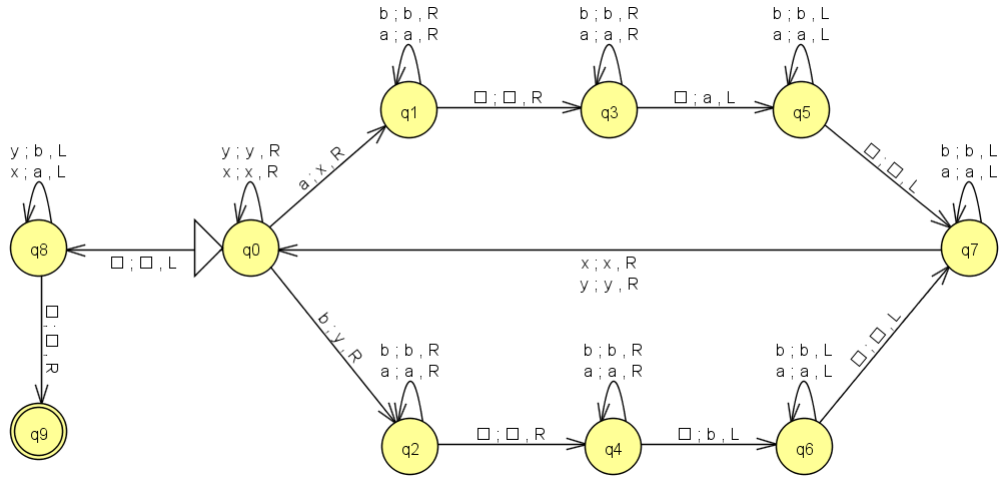
11. Reverse a given string. Show a sequence of configurations for the input strings *ababbb*.

Solution: Assuming that the alphabet is {*a*, *b*}, the Turing Machine is:



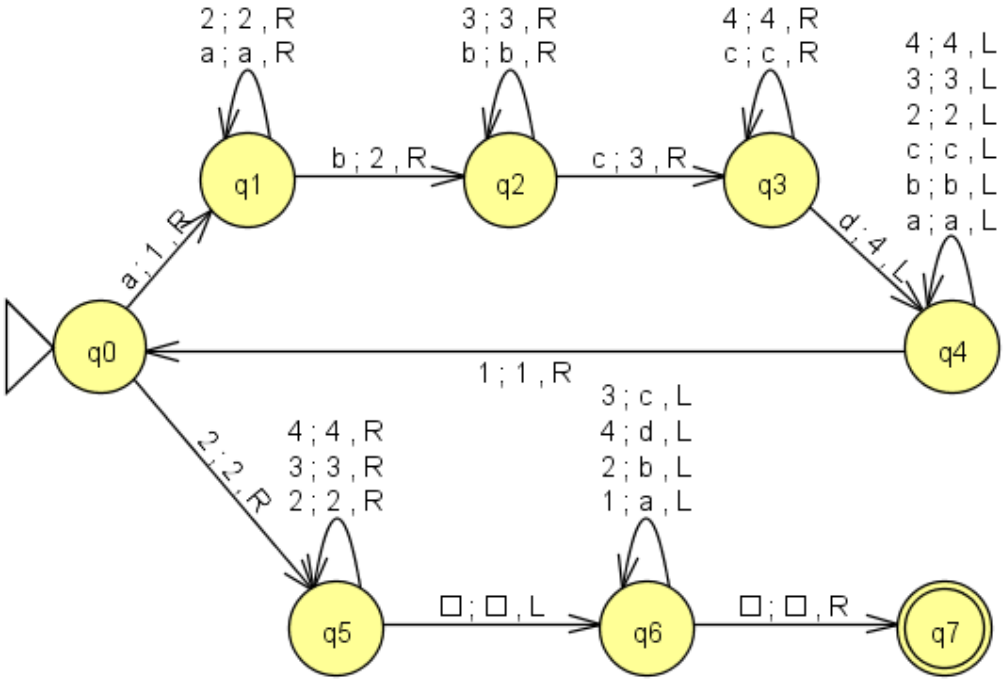
12. Copy a given string, that is, the tape should contain a second copy of the input string separated by a single blank cell from the given string.

Solution: Assuming that the alphabet is {*a*, *b*}, the Turing Machine is:



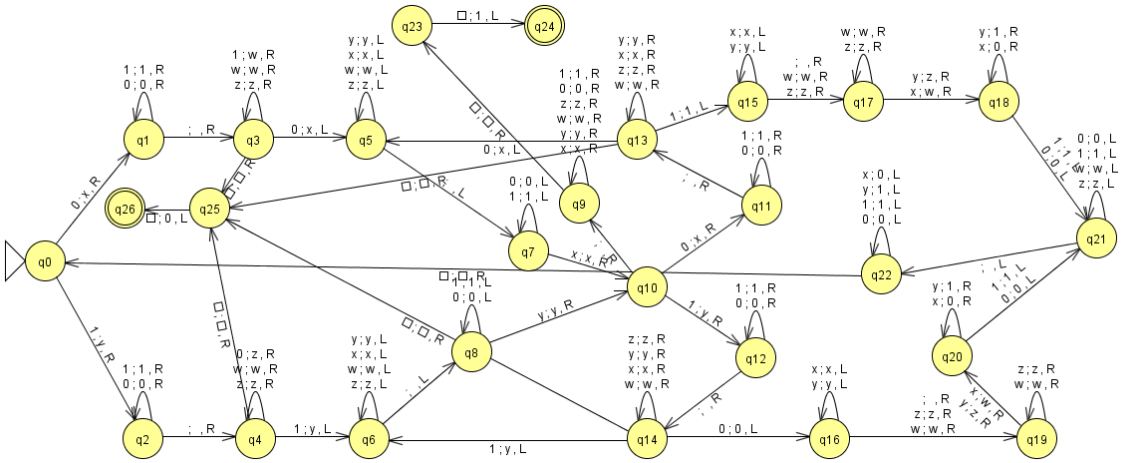
13. Accept strings of the form $a^n b^n c^n d^n$. Show an accepting sequence of configurations for the input *aaabbbcccddd* and show how *aaabbbccccdd* is rejected.

Solution: Please see also the VIDEO SOLUTION for this exercise.



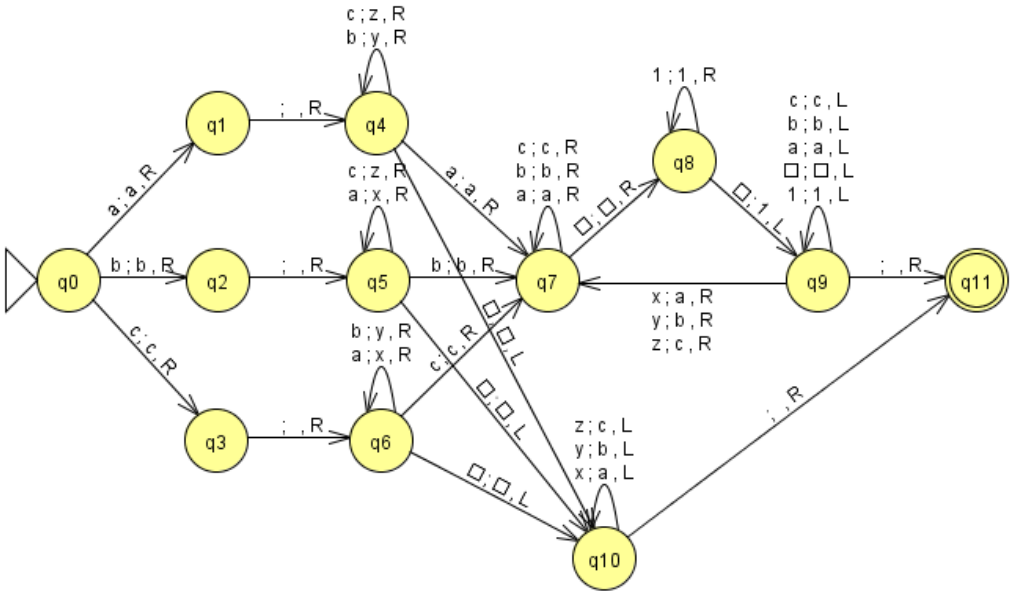
14. Check if the first part of the string is present as a sub-string of the second part of the string, the two parts being separated by a single blank cell. The output shall be a 0 or 1 indicating the presence or absence of the sub-string in the given string, respectively. Show the computation for the input 011 001011011.

Solution:



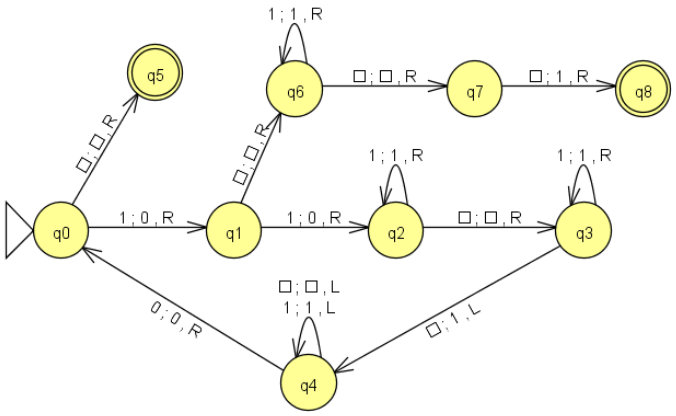
15. Find the index of a symbol in a given string, that is, given a symbol followed by a blank and a string, for example, *a bcbaaba*, it finds the first position where the given symbol occurs in the string, for example, 4 in the example, and outputs the number in unary. The final contents of the tape in the example shall be *a bcbaaba* 1111.

Solution:



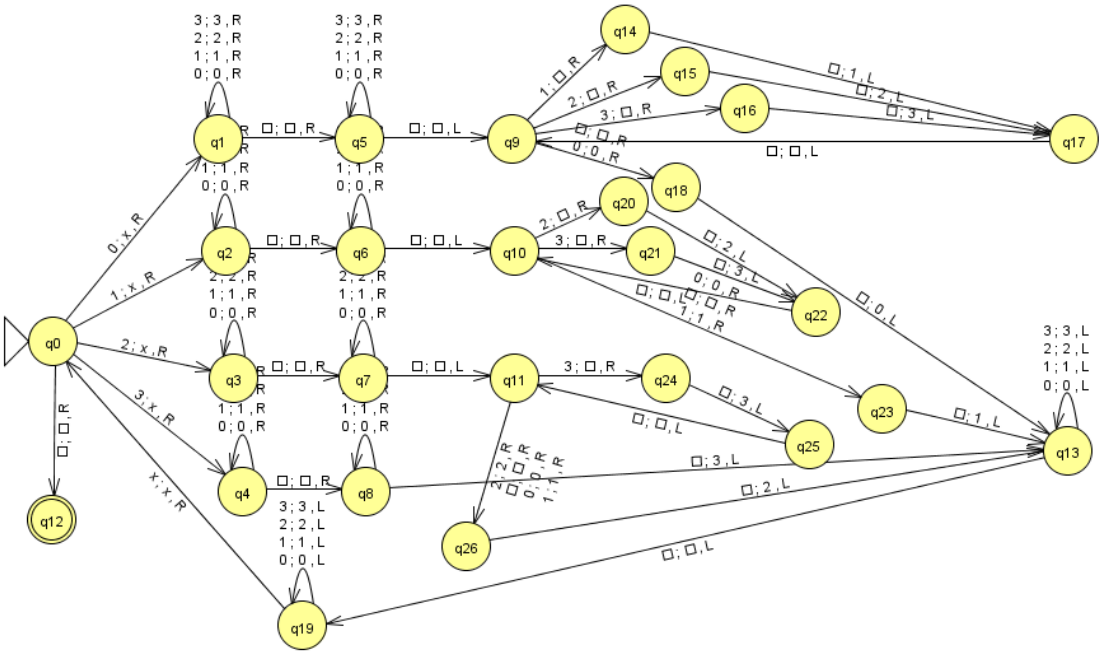
16. Divide a given number by two in the unary number system. The quotient and the remainder (separated by a blank) should be written on the tape when the machine halts. For example, given 1111 on the tape, the output should be 11 (i.e., 4/2 = 2 and there is no remainder). As another example, given 1111111, the output should be 111 1 (i.e., 7/2 = 3 and the remainder is 1).

Solution:



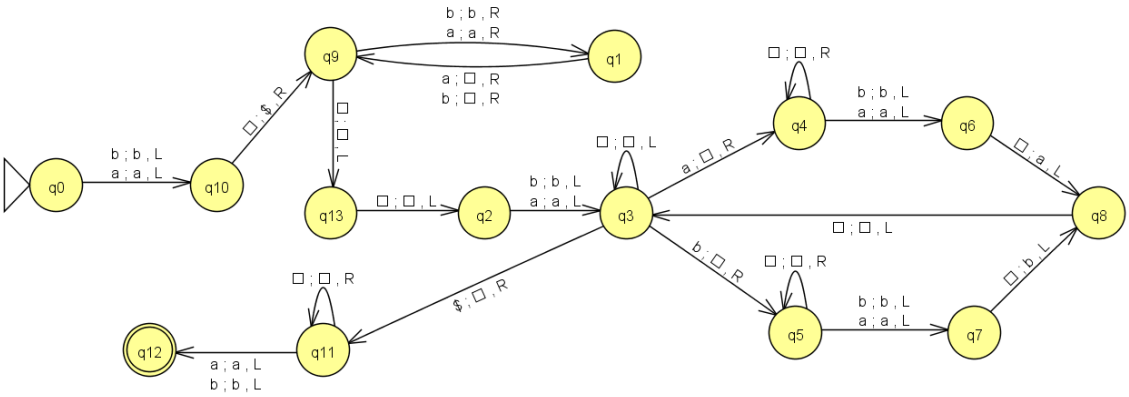
17. Perform insertion sort, that is, given a string over {0, 1, 2, 3}, write the symbols in increasing order on the tape by repeatedly inserting the next symbol in its proper place in the sorted output. For example, given 3021312, the output shall be 0112233 (with the intermediate contents of the tape being 3, 03, 023, 0123, 01233, 011233, 0112233).

Solution:



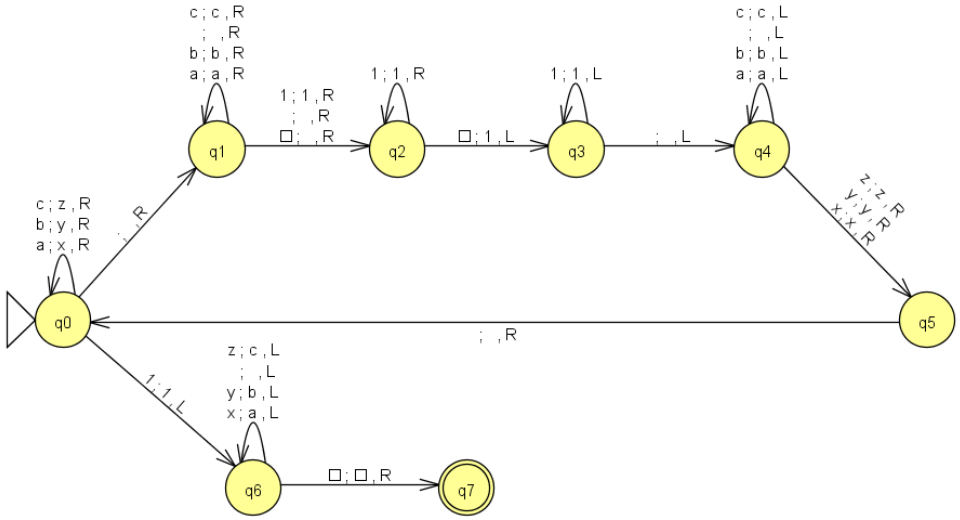
18. Extract all symbols in odd-numbered positions. If the input is a string $w = w_1w_2w_3\ldots w_n$ where each w_i is *a* or *b* and *n* is an even number, the machine halts with the contents of the tape being $w_1w_3w_5\ldots w_{n-1}$.

Solution:



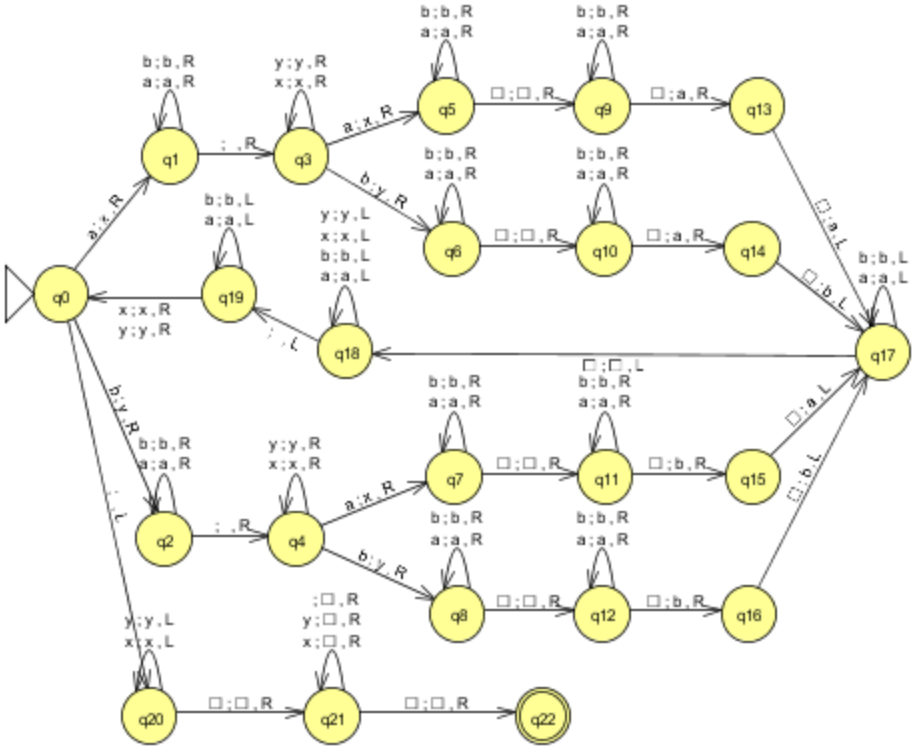
19. Implement a simple tokenizer: given a string with single blank spaces separating other symbols, count the number of tokens in the string and output this number in unary. For example, given ab bc abc bca abc a b c, the output shall be ab bc abc bca abc a b c 11111111.

Solution:



20. Interleave two strings. Given two strings of equal length (separated by a single blank cell), such as $a_1a_2a_3…a_n$ $b_1b_2b_3…b_n$, the machine halts after writing on the tape an interleaved string composed of the first elements of the two strings, followed by the second elements of the two strings, and so on, ending with the last elements of the two strings, in that order: $a_1b_1a_2b_2a_3b_3…a_nb_n$.

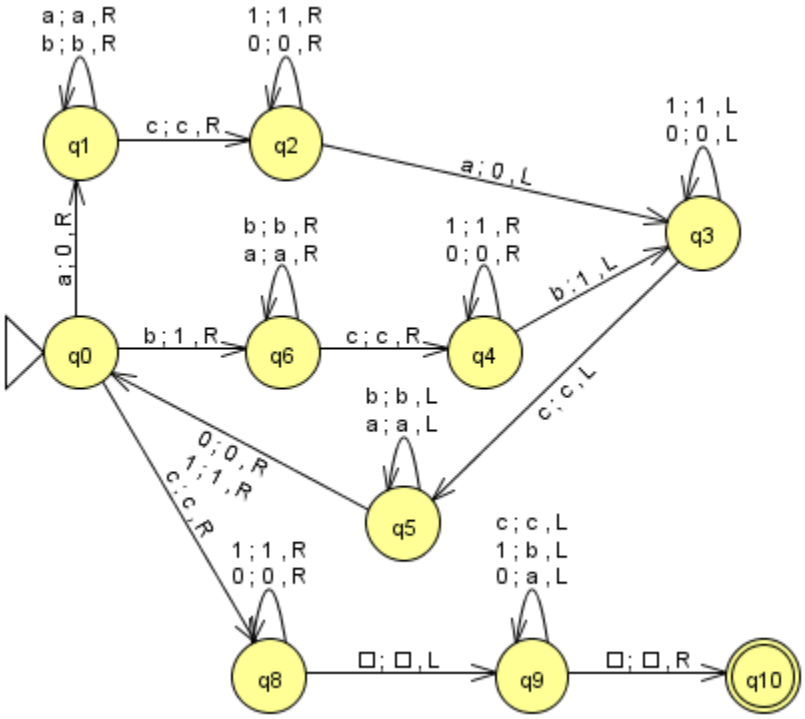
Solution: Please use JFLAP to load the accompanying jff file if this diagram is not readable:



B. Make changes to the given Turing machine.

21. Modify the Turing machine shown in Figure. 10.13 to accept *wcw* where *w* = (*a* + *b*)* and c is a special character separating the two parts.

Solution: There is no need to discover the midpoint now. The modified Turing machine is:



22. Show what happens when the Turing machine in Figure. 10.9 is given the input 000011112211.

Solution: The Turing machine halts in state q_2 and rejects the input string. It does, however, go beyond the 1 s until the blank cells looking for a matching 2.

C. Debug and fix the following Turing machines.

23. The Turing machine to compute the weight of a binary string, that is, the number of 1 s in the string (to be written as a unary number on the tape) shown in Figure-10.15:

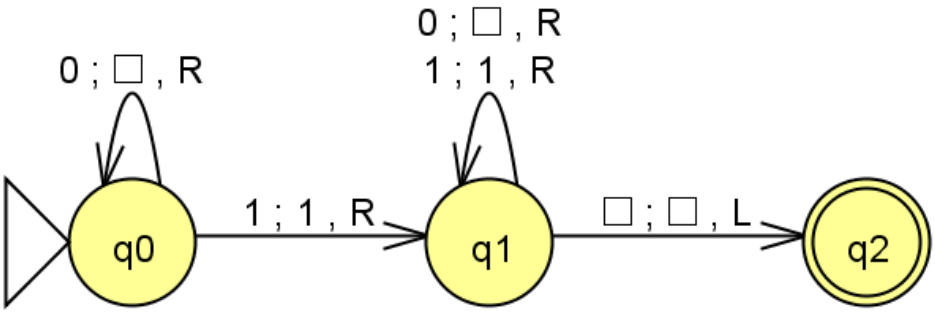
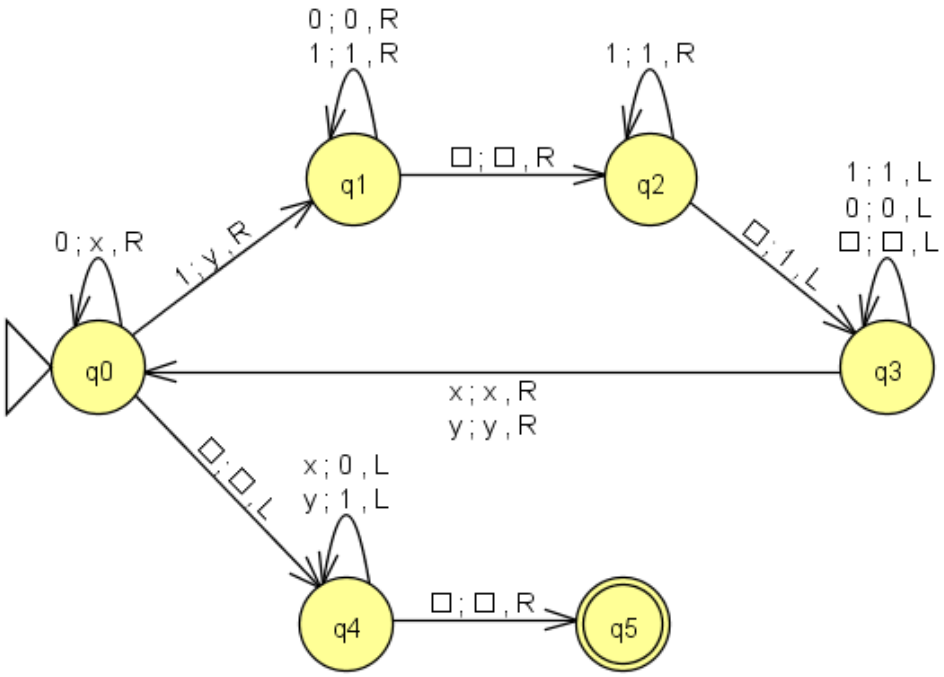


Figure-10.15

Solution: Please see also the VIDEO SOLUTION for this exercise.



24. The Turing machine to accept a given binary string if it has an unequal number of 0 s and 1 s shown in Figure. 10.16:

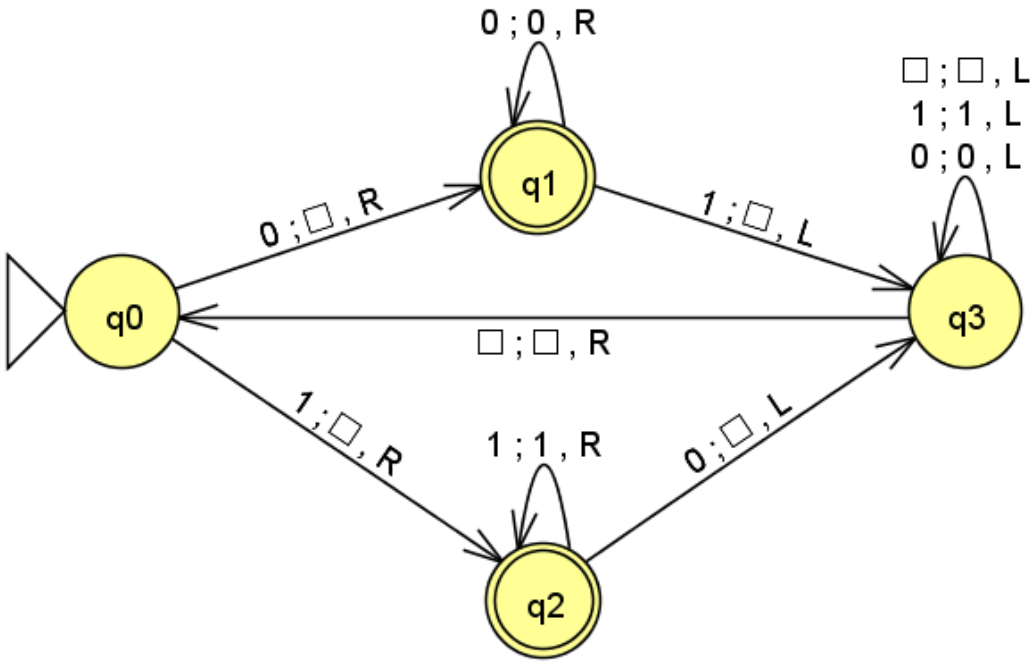
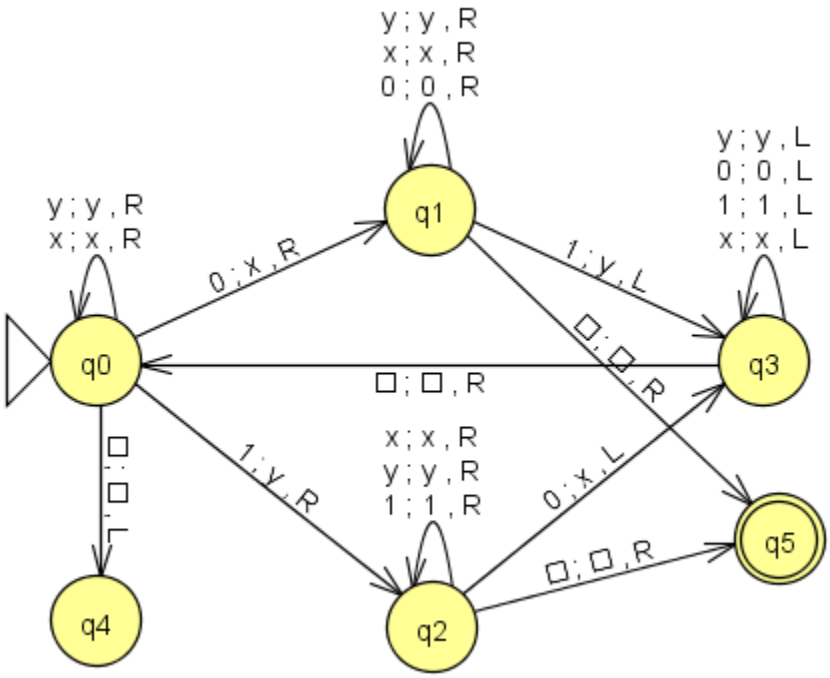


Figure-10.16

Solution:



25. The following Turing machine to add 2 to a given binary number:

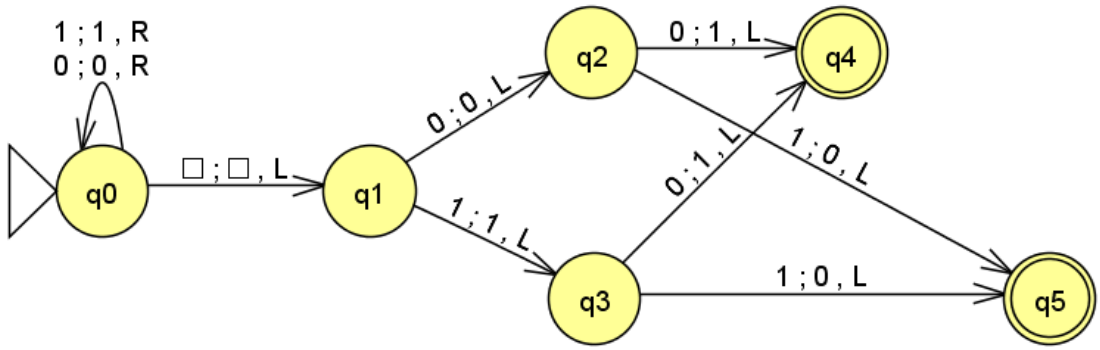
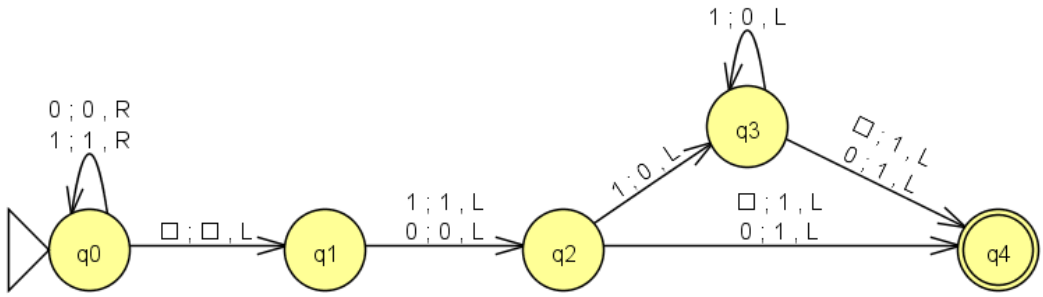


Figure-10.17

Solution:



26. The following (incomplete) Turing machine to search for its first argument in its third argument and replace every occurrence of it with its second argument, for example, given <ED> please ensure no line break in the middle of example strings </ED> *b p bababbabbabba*, the ouptput is *papappappappa*, the alphabet being {*a*, *b*, *p*}:

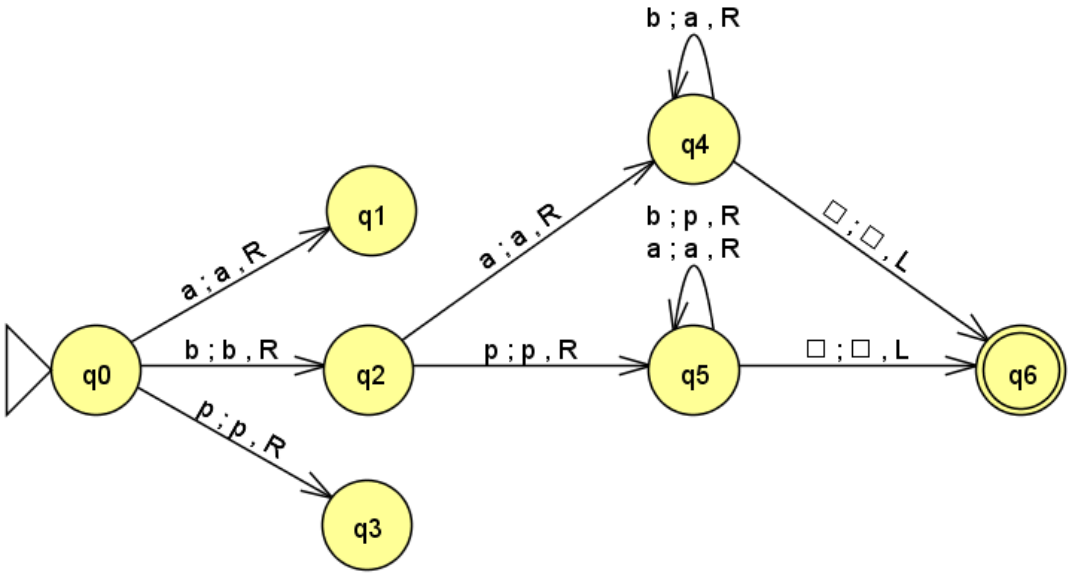
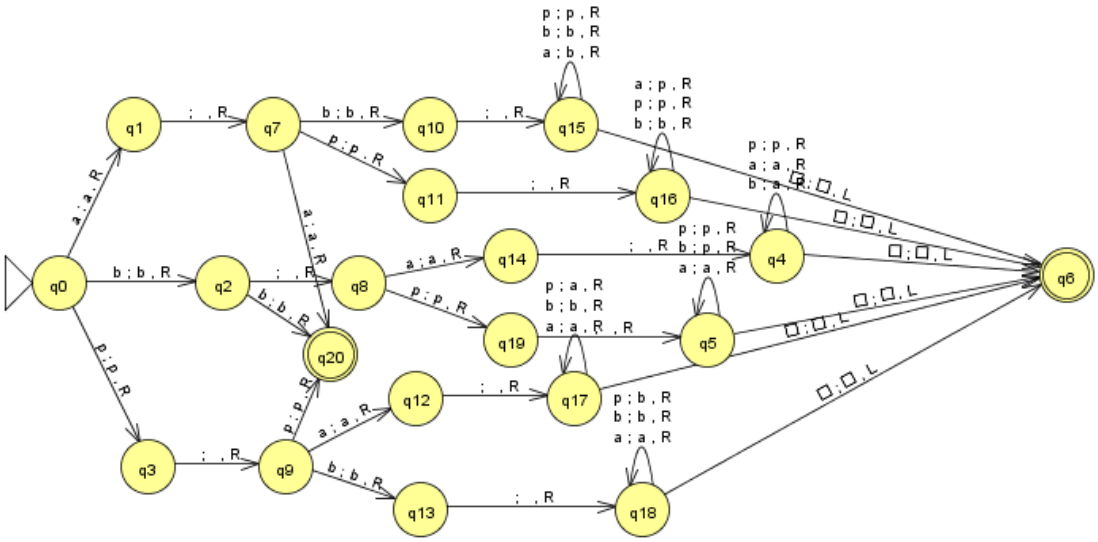


Figure-10.18

Solution:



D. Show that each of the following variations of a Turing machine is equivalent to the standard Turing machine.

27. Multi-track machine: The tape has multiple parallel tracks. *Hint*: Interleave the contents of the multiple tracks on a single tape (similar to Exercise 20 above).

    Solution: See Theorem 20.2 in Appendix B.

28. Multi-tape machine: There is more than one tape (as in the universal Turing machine). *Hint*: Interleave the contents of the multiple tapes on a single track (similar to Exercise 20 above).

    Solution: See Theorem 20.4 in Appendix B.

29. Multi-dimensional tape machine: The tape has two, three, or higher dimensions. *Hint*: Linearize or serialize the contents of the multi-dimensional tape on a single, one-dimensional tape (the same way address calculations are done in a multi-dimensional array in a modern programming language).

    Solution: See Theorem 20.5 in Appendix B.

30. One-sided tape machine: The tape is semi-infinite, with a definite left-most cell. *Hint*: Split it into a two-track tape with the second track serving to make up for the missing infinitely long tape to the left.

    Solution: See Theorem 20.3 in Appendix B.

31. Turing machine with stay option: Where it is not mandatory that a transition makes the head move left or right.

    Solution: See Theorem 20.1 in Appendix B.

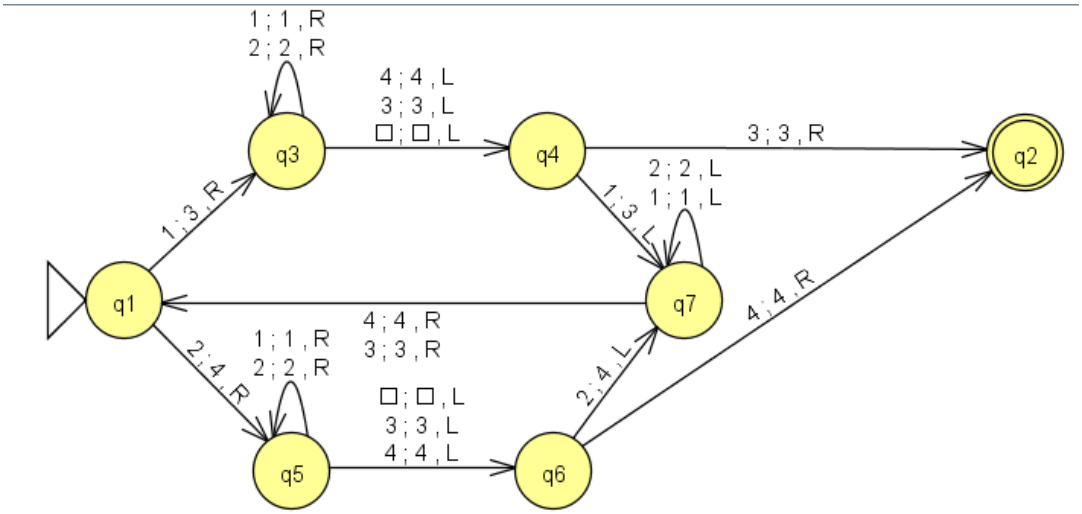
32. Non-deterministic Turing machine: Where there can be more than one transition for the same current state and the same symbol on the tape. *Hint*: Use the idea of a backtracking algorithm.

    Solution: See Theorem 20.6 in Appendix B.

E. Universal Turing machine.

33. Encode the Turing machine you constructed in Exercise 1 as a binary string.

    Solution: The Turing machine after renaming states and symbols for encoding purposes is:



    The encoding is:

    00101011101110100111010111010100111011011101101001110011110011001110111011110111011001111
    11101111011110110011111011101101110100111101011111110111011001111111010111111101011001111111
    10110111111101101100111111101110101110100111111101111010111101001011011111011110100111110
    01111101010011111011011111011010011111001111110011001111101110111101111011101100111110111011
    11110111101100111111011011111110111101100111111011110110111010100

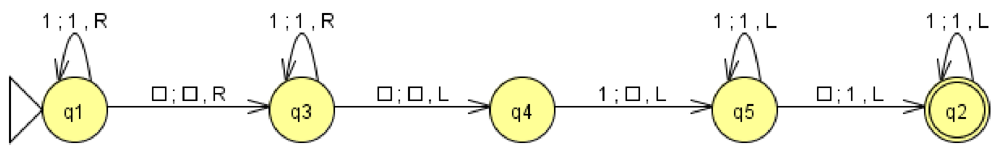
    It may be noted that the blank symbol is a null in the unary number system.

34. Show how a universal Turing machine can simulate the computation of the above encoded Turing machine for the input string *abbba*.

    Solution: The encoded string from Exercise 33 is loaded on to the tape of the universal Turing machine. Its start state is 1. On seeing the first symbol (with the string encoded as 12221), the machine finds a matching transition and changes to state 3. It reaches the end of the input string in this state and reaches state 4 from which it reaches state 7 after matching the two 1 s at the two ends of the input string. Similarly, it matches the two 2 s inside. Finally, it processes the 2 at the centre and reaches the final state: state 2.

35. Encode the unary adding Turing machine from Example 10.2 (Figure-10.5) as a binary string. Also construct a Turing machine for subtracting a given (larger) unary number from another (smaller) unary number. Encode this too as a binary string. Show that the same universal Turing machine can simulate the computations of both the adding and the subtracting Turing machines.

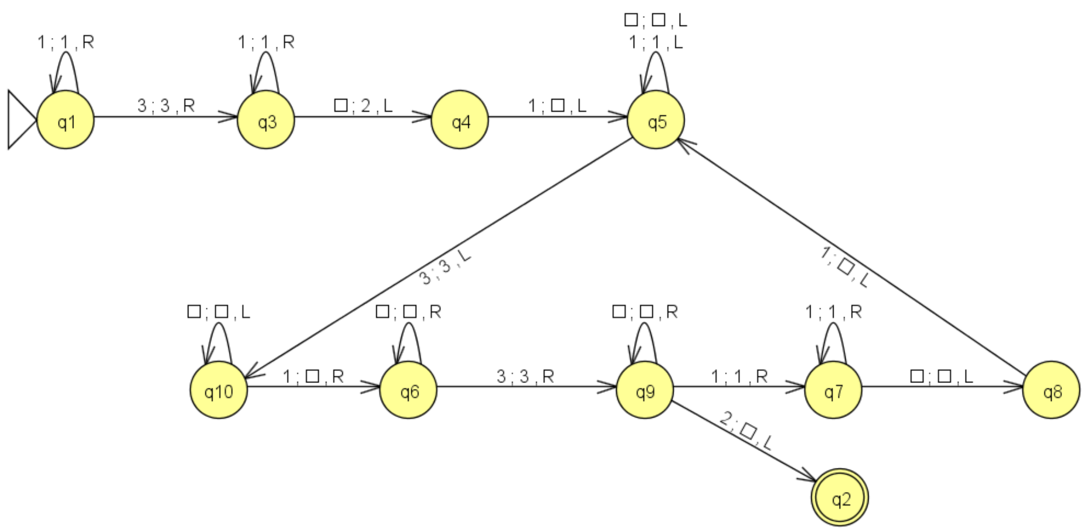
    Solution: The adding Turing machine after renaming is:



Its encoding is:

0010101010100100111001001110101110101001110011110011001111010111110011001111101011111101011
00111110011010110011010110101100

The Turing machine for proper subtraction which uses the symbol 2 to mark the end of the input and the symbol 3 to separate the two given numbers is:



The encoding for this machine is:
0010101010100101110111011101001110101110101001110011110110110011110101111100110011111101011
1110101100111110011111001100111110111011111111110111011001111111111001111111111100110011111
1111101011111100100111111001111110010011111101110111111111101110100111111111001111111110010
0111111111011011001100111111111010111111101010011111110101111111010100111111100111111111001
1001111111101011111001100

Either of the two encodings can be loaded and executed by a universal Turing machine in exactly the same way.