Student Information

Full Name : Anıl Eren Göçer
Id Number : 2448397

Answer 1)

Screenshot of the Designed Machine

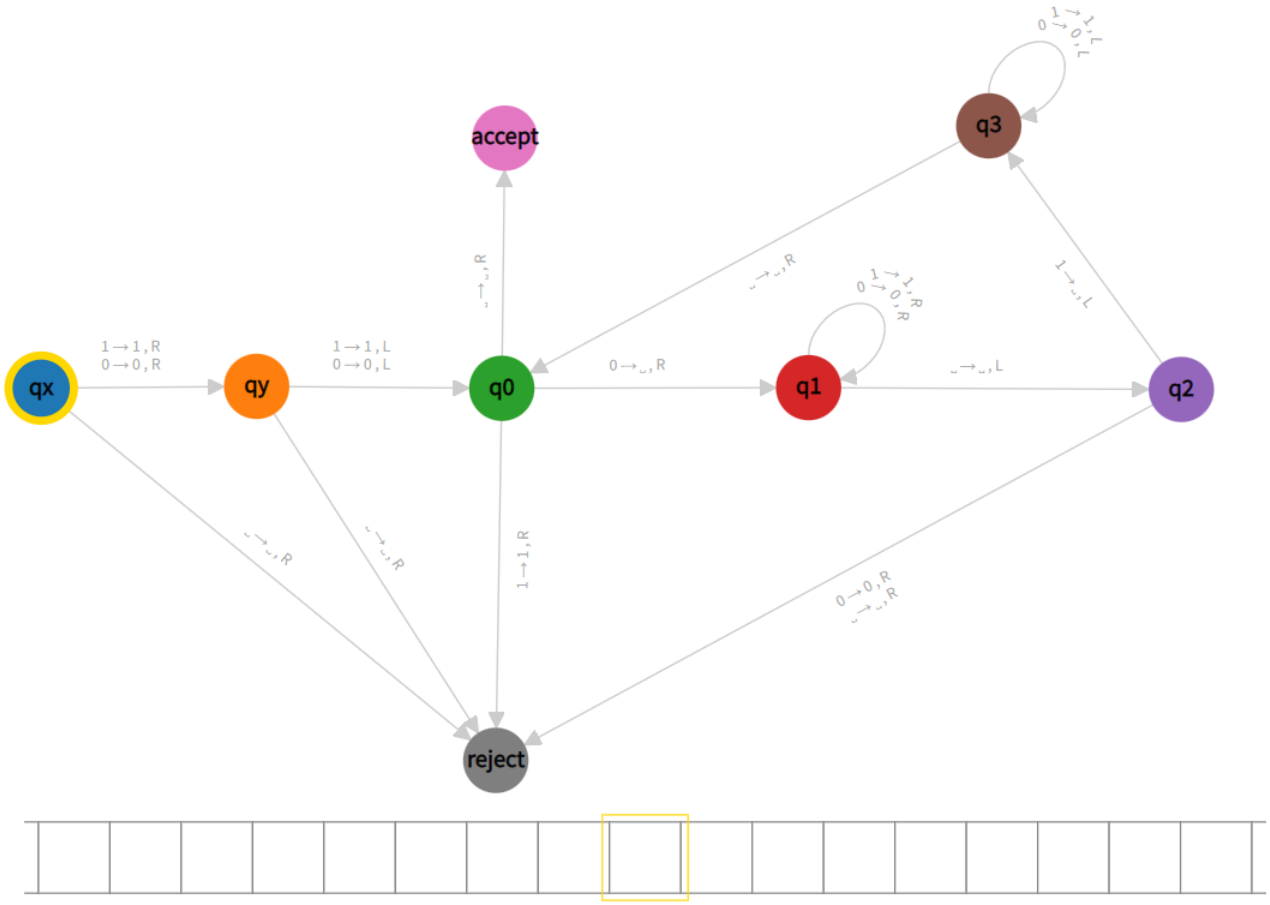

Figure 1.1: Turing machine recognizing the language L = $\{0^N 1^N | N \geq 1\}$

Description of the States Used In the Machine

The machine and General Idea

The logic that I applied is to match the left symbol (which should be 0) with the right-most symbol (which should be 1), then remove both symbols replacing them with the blank symbols, and then iterate starting at the new left-most symbol.

The Turing Machine I have designed above has the tape alphabet $\Gamma = \{0, 1, \sqcup\}$, and the set of states $Q = \{q_x, q_y, q_0, q_1, q_2, q_3, accept, reject\}$, and halting states $H = \{accept, reject\}$, and start symbol q_x. Actually, this Turing Machine consists of two Turing Machines. While the first part of the Turing Machine which includes q_x and q_y functions to handle empty string case, the second part which includes q_0, q_1, q_2, q_3 is designed to satisfy core functionality which is detecting the language L = $\{0^N 1^N | N \geq 1\}$.

State Descriptions

The state q_x is for rejecting the empty string. The state q_y is for rolling-back the tape which has been shifted in the state q_x. In the state q_0, we erase the left-most 0, and go to state q_1 which bring us to the right end of the input string. Once we pass the last symbol, we come one symbol left and enter state q_2. In state q_2, scanning the right-most symbol, we check if this is a 1. If not, we reject; otherwise, we delete that 1, and start moving left all the way to the new left-most symbol using the state q_3. Once we determine the left-most symbol, we enter q_0 and repeat the same algorithm on the shorter input string. If the machine halts in the *accept* state, then this means the input string is in the language L. If the machine halts in the *reject* state, then this means the input string is not in the language L.

Initial and End State Screenshots for Sample Inputs

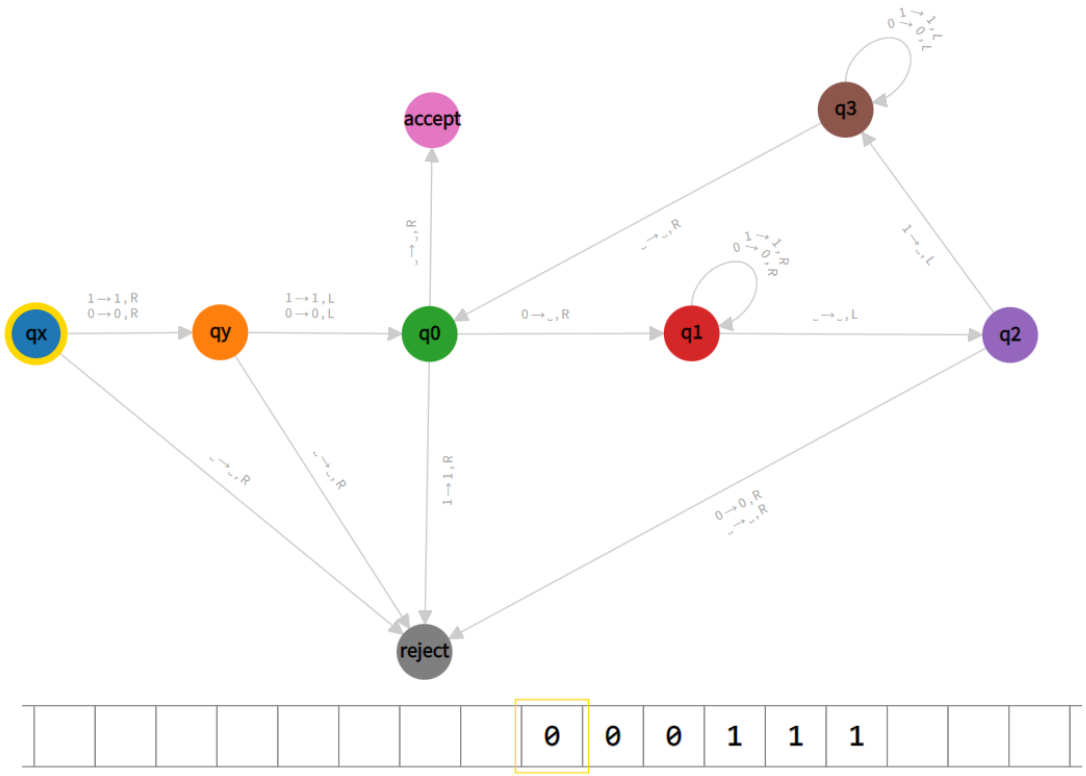

Figure 1.2.1: Initial state of the machine for the input **000111**

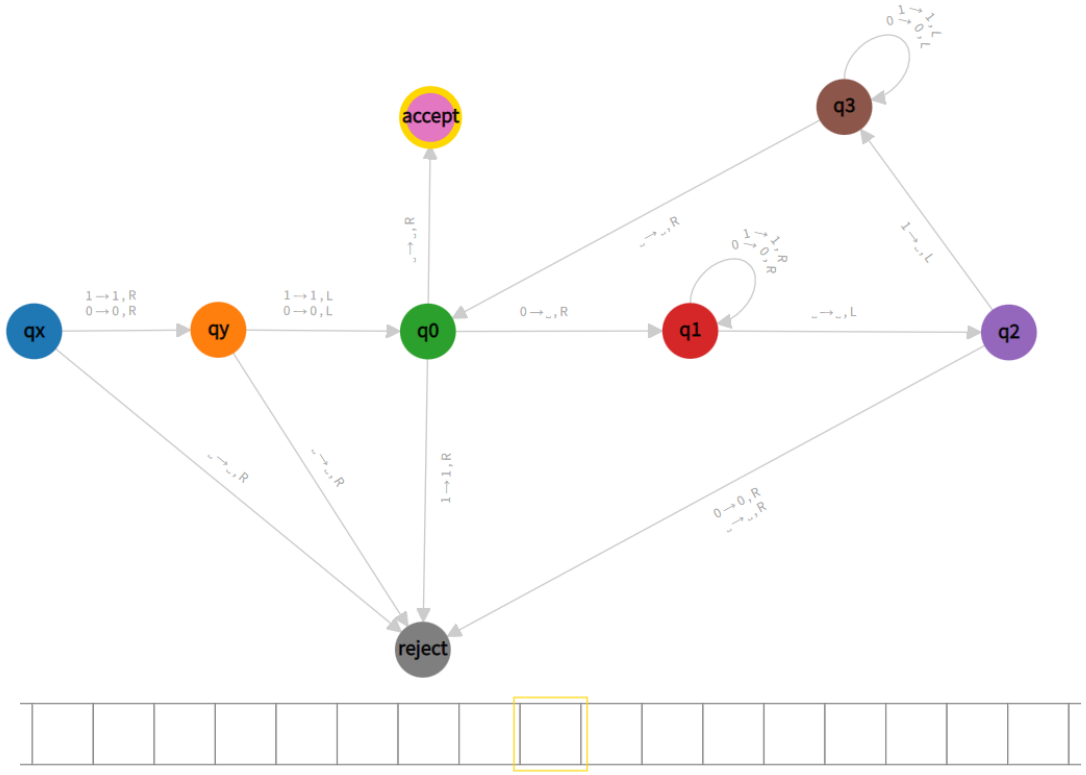

Figure 1.2.2: End state of the machine for the input **000111**

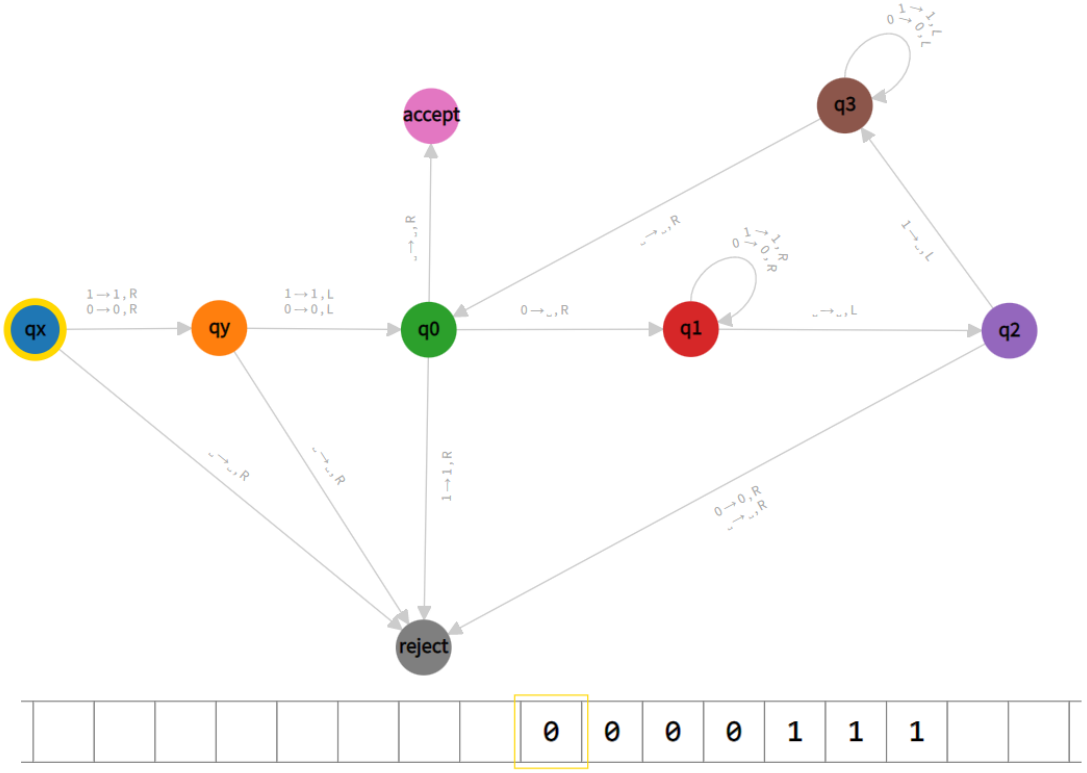

Figure 1.3.1: Initial state of the machine for the input **0000111**

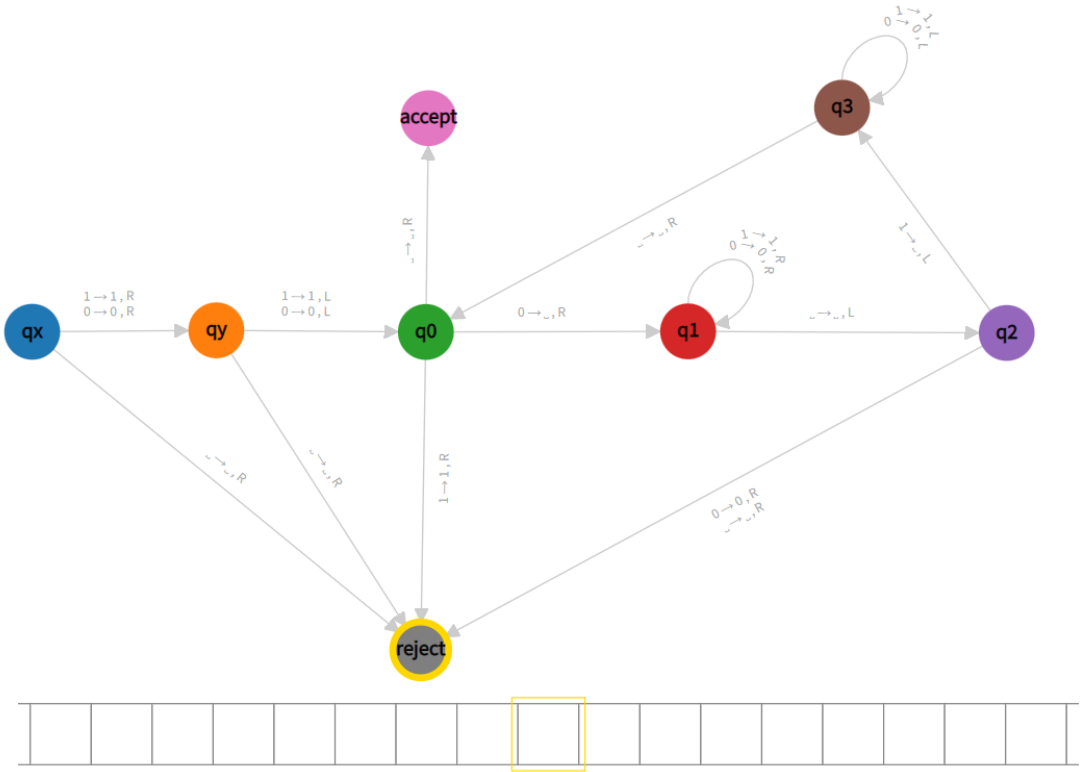

Figure 1.3.2: End state of the machine for the input **0000111**

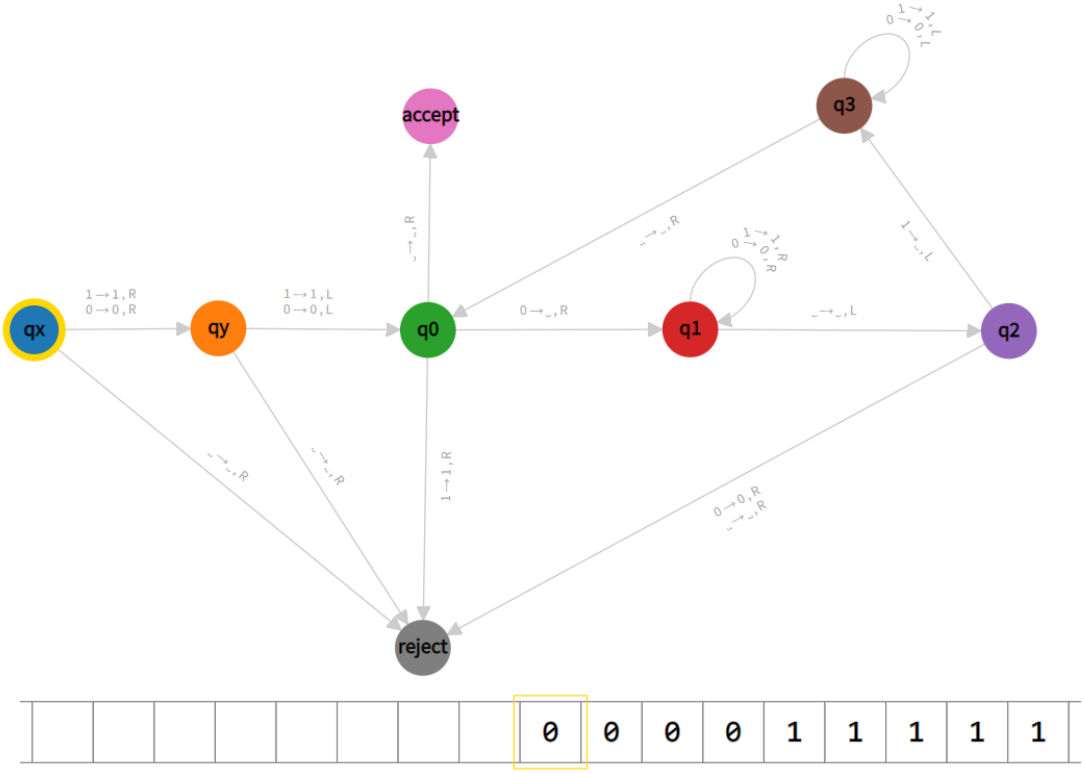

Figure 1.4.1: Initial state of the machine for the input **000011111**

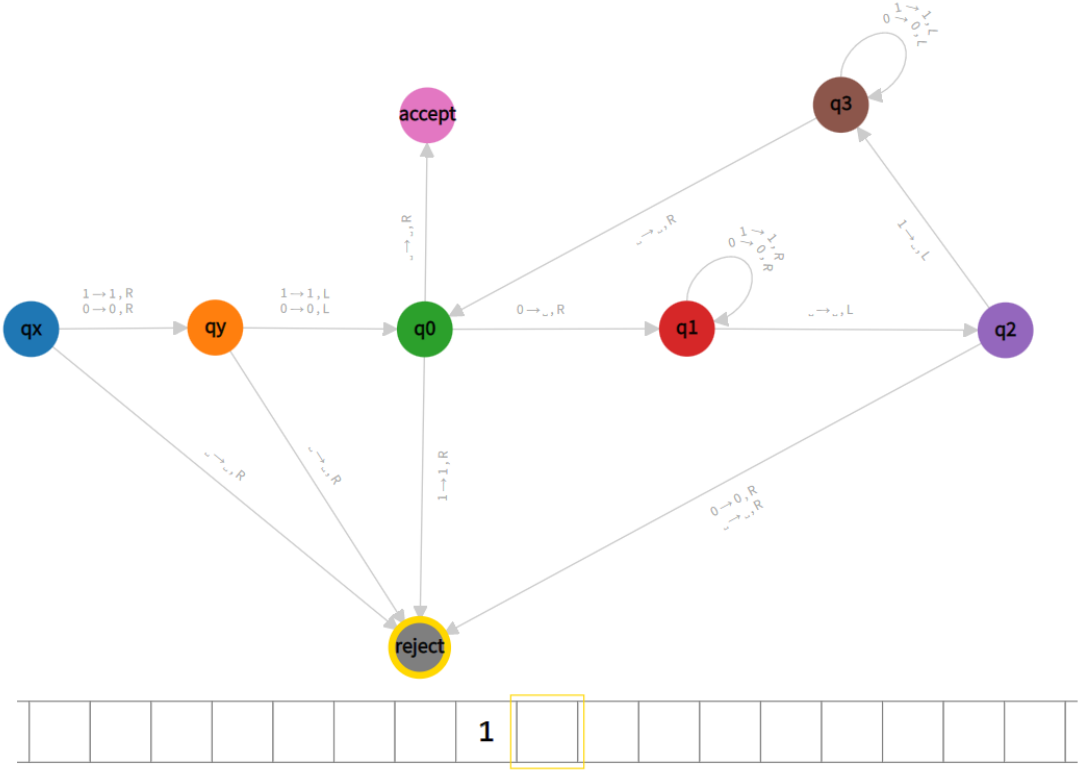

Figure 1.4.2: End state of the machine for the input **000011111**

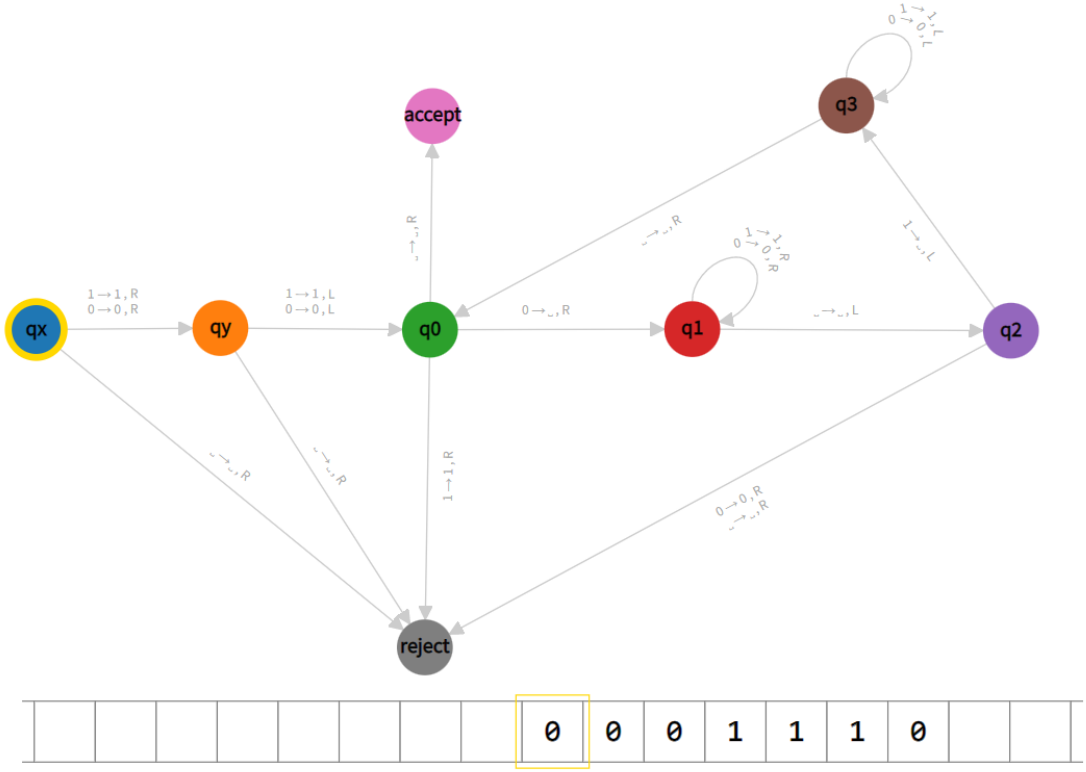

Figure 1.5.1: Initial state of the machine for the input **0001110**

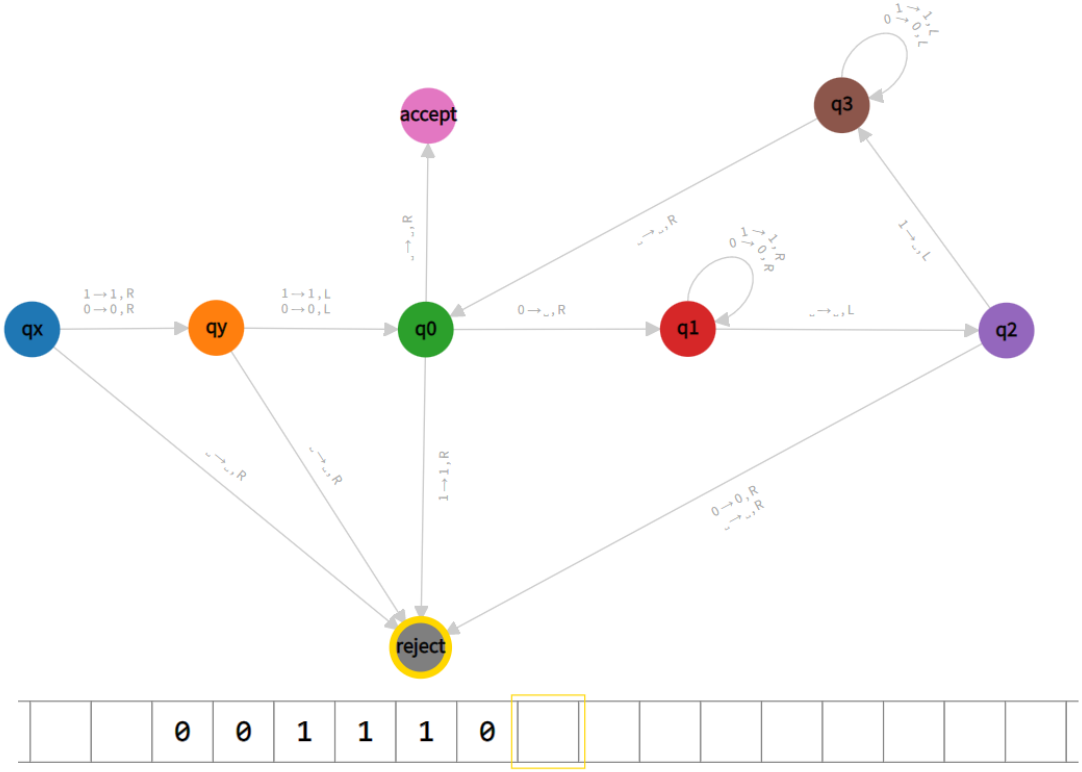

Figure 1.5.2: End state of the machine for the input **0001110**

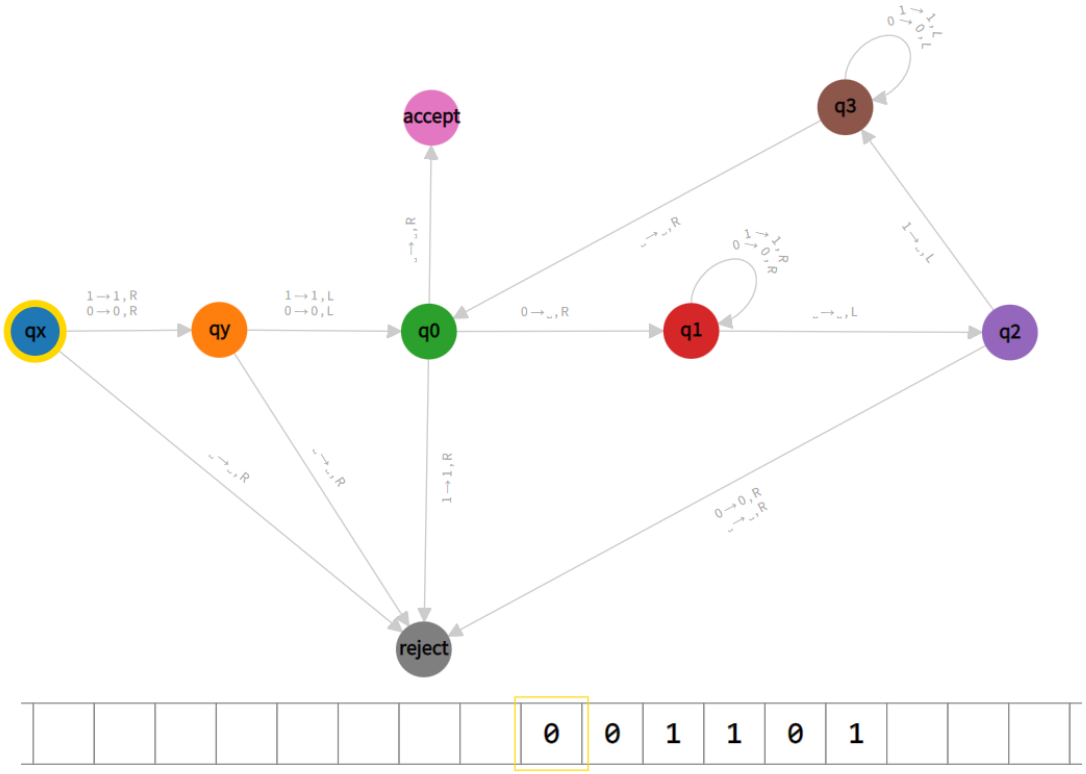

Figure 1.6.1: Initial state of the machine for the input **001101**

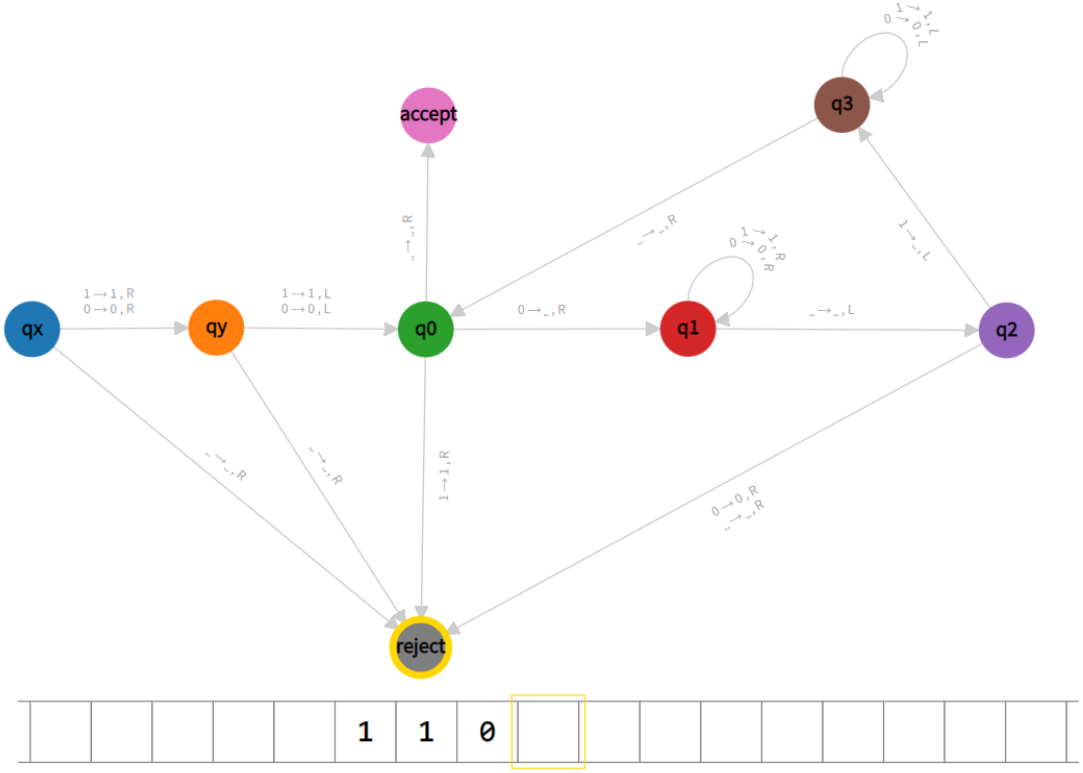

Figure 1.6.2: End state of the machine for the input **001101**

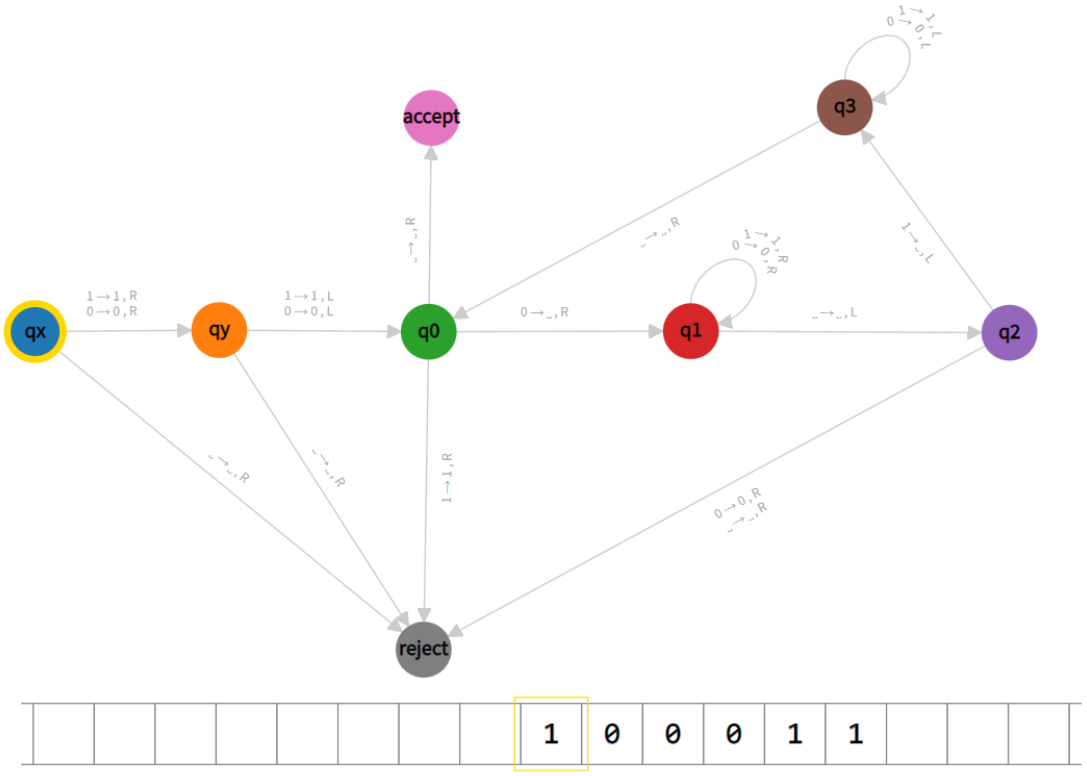

Figure 1.7.1: Initial state of the machine for the input **100011**

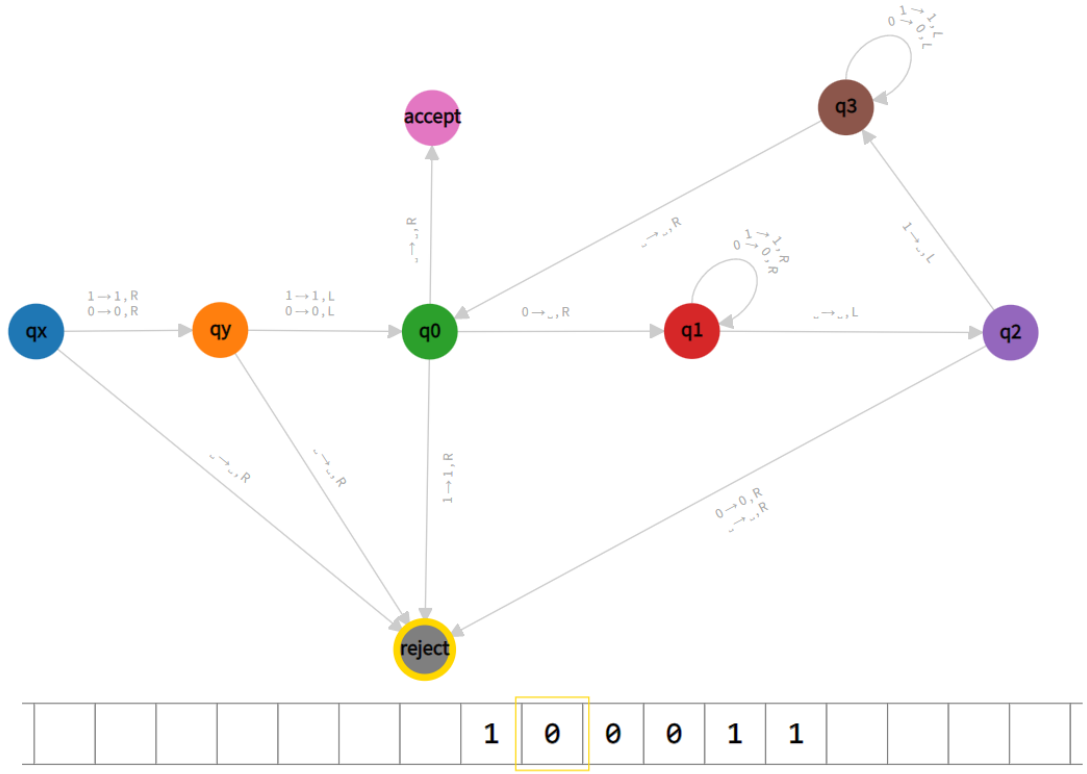

Figure 1.7.2: End state of the machine for the input **100011**

Answer 2)

Screenshot of the Designed Machine

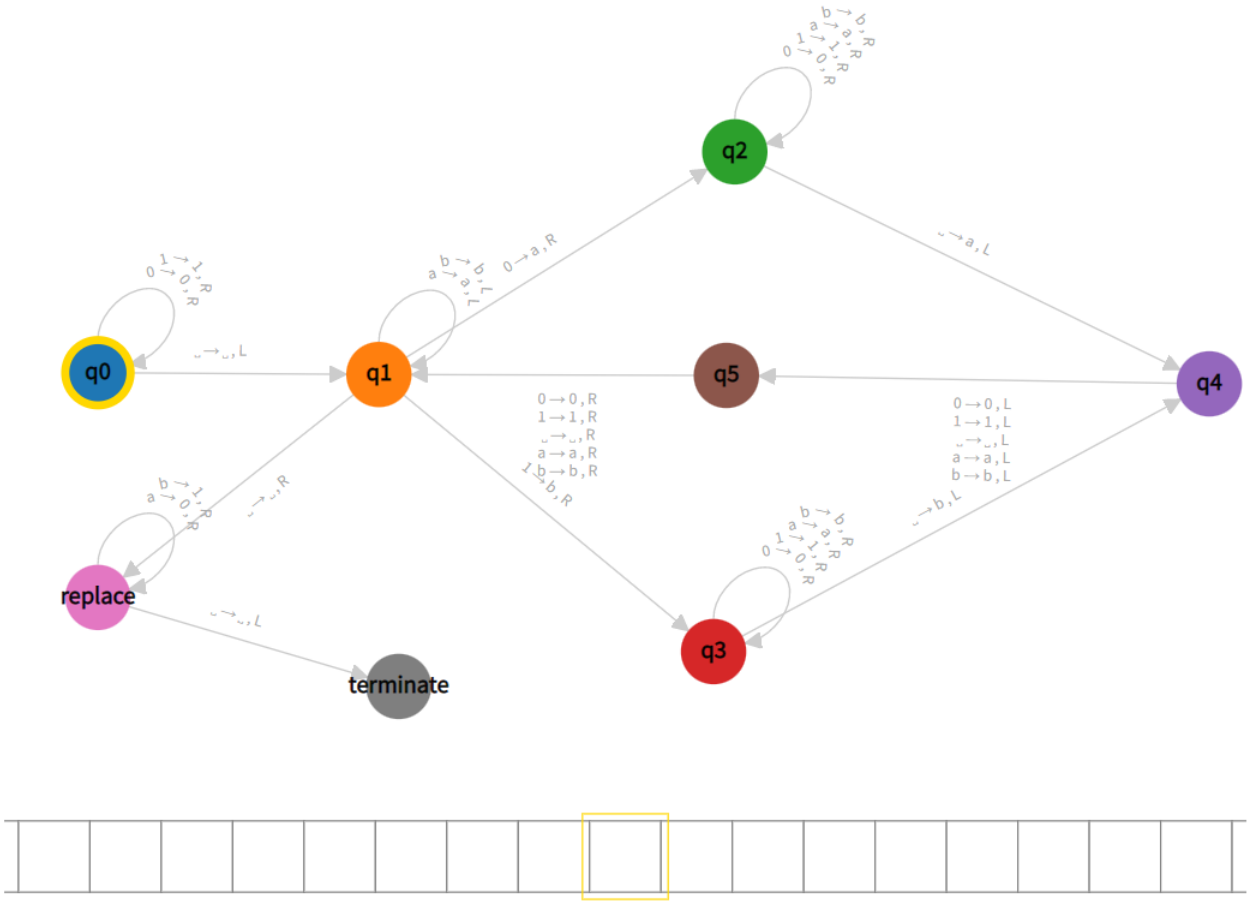

Figure 2.1: Turing machine computing the function $f(w) = ww^R, w \in \{0,1\}^+$

Description of the States Used In the Machine

The Machine and General Idea

The Turing Machine I have designed above has the tape alphabet = $\{0, 1, a, b, \sqcup\}$, and the set of states Q = $\{q_0, q_1, q_2, q_3, q_4, q_5, replace, terminate\}$, and the halting state $\{terminate\}$, and start symbol q_0. The idea that I applied is to copy the last symbol of the input string and marking them as processed by a if it is 0, and by b if it is 1, then paste a if it is 0 and b if it is a to the end of the total string. Then by going to new right-most 0 or 1, we apply the same algorithm iteratively until there is not 0 or 1 on the tape. Then we will replace each a with 0 and each b with 1. Then, we terminate the machine.

State Descriptions

When the machine starts, we go to the right of the right-most symbol by using the start state q_0. Then, we go to left until finding the first 0 or 1 symbol, and mark it as a if it is 0, and mark it as b if it is 1, and based on the symbol that we mark we go to q_2 or q_3. We go to right of the rightmost symbol using q_2 or q_3 and paste there a if we are in the state q_2, or paste there b if we are in the state q_3, and then go to state q_4. By using q_4 and q_5 we are adjusting the tape so that we are at the end of the current string and in the state q_1. Now, we are in the state q_1, we apply the same algorithm until there will be no 0 or 1 on the input tape. Then, we go to *replace* state, and we replace each a with 0 and each b with 1. Then, we go to *terminate* state and halt the machine.

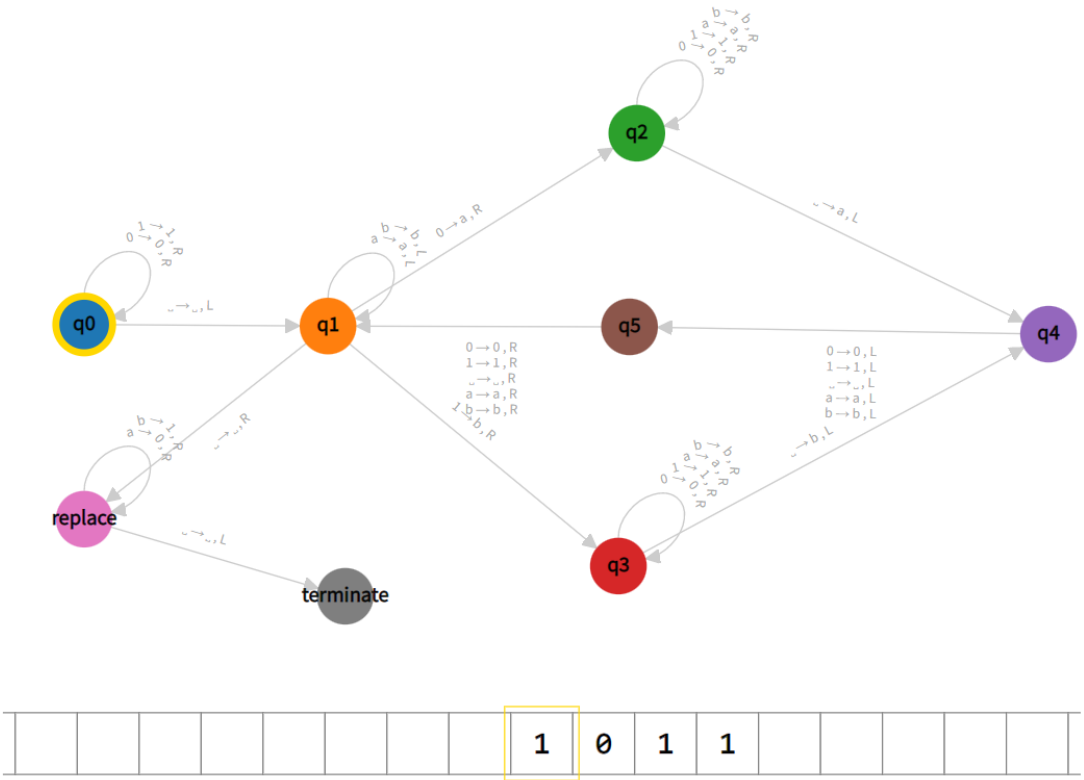

Figure 2.2.1: Initial state of the machine for the input **1011**

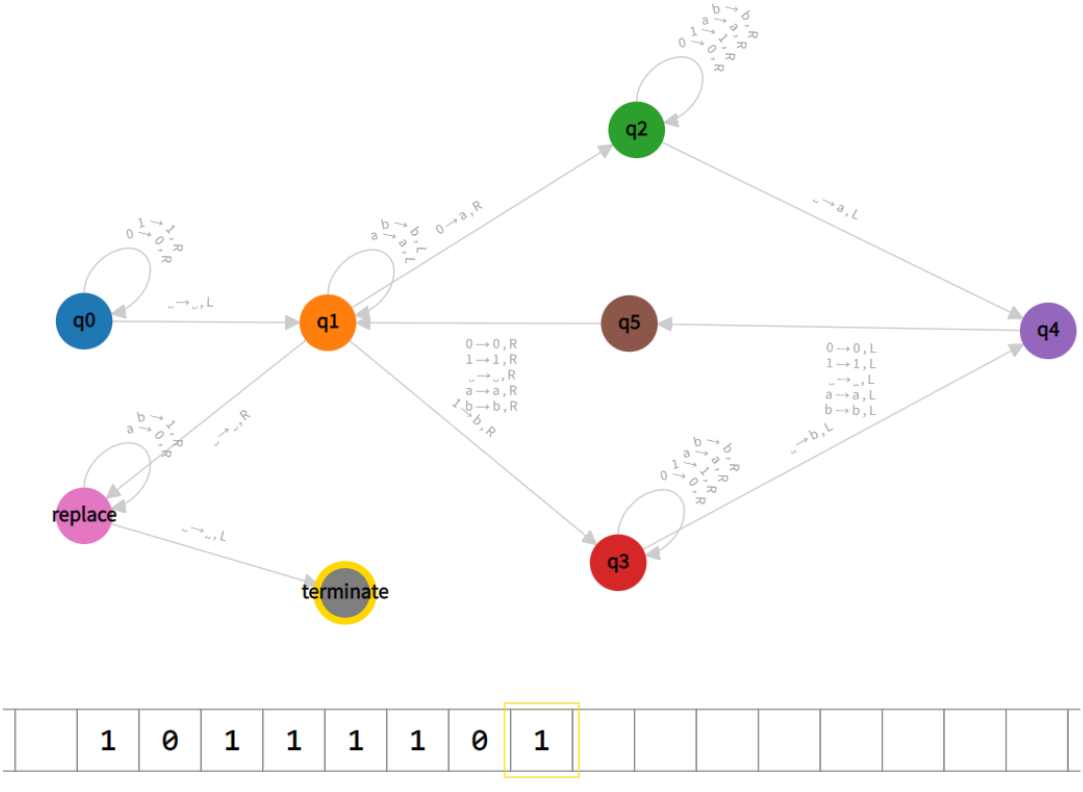

Figure 2.2.2: End state of the machine for the input **1011**

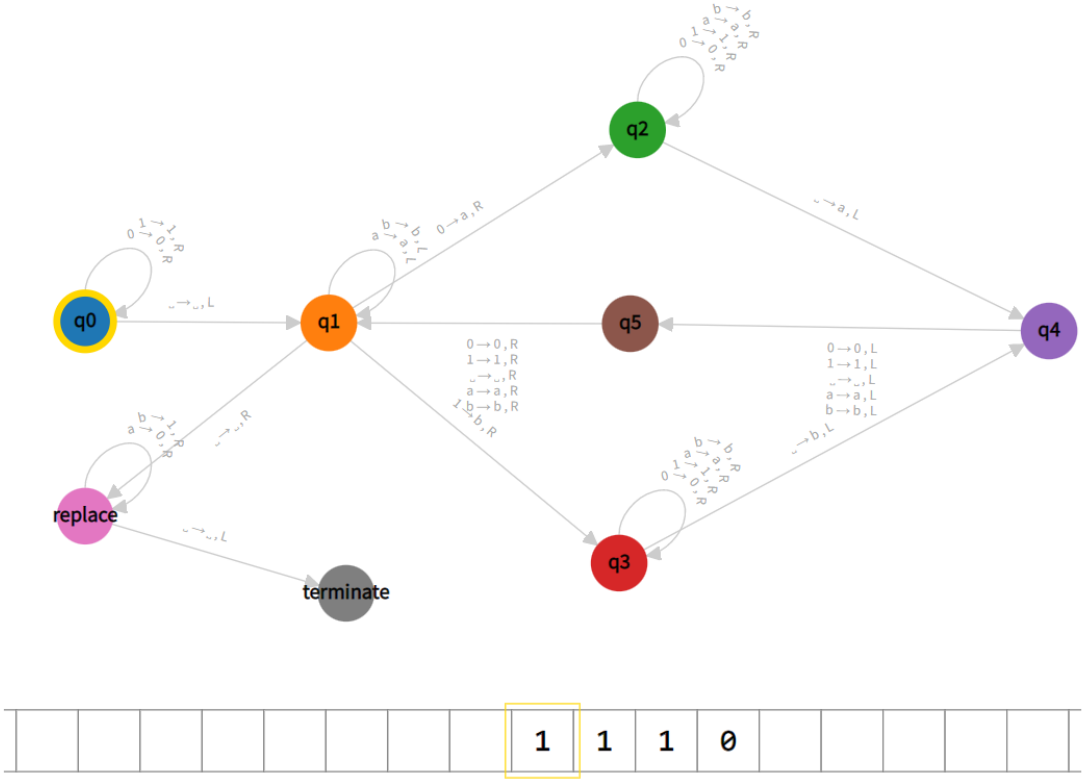

Figure 2.3.1: Initial state of the machine for the input **1110**

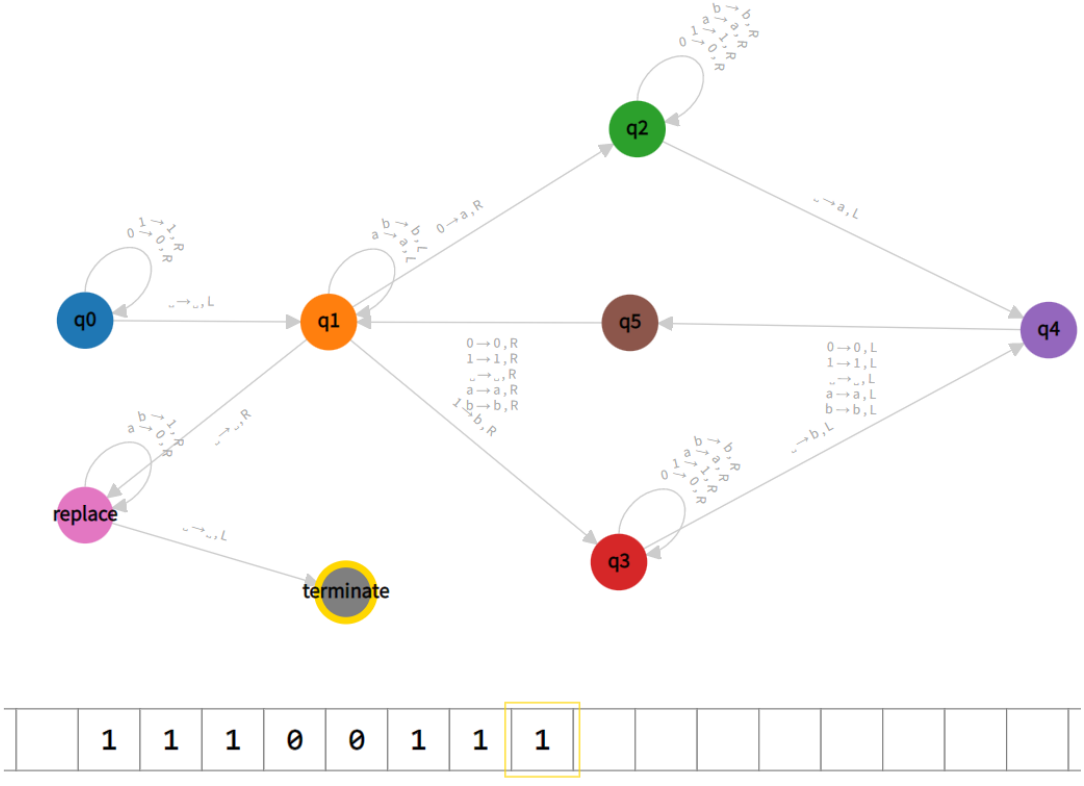

Figure 2.3.2: End state of the machine for the input **1110**

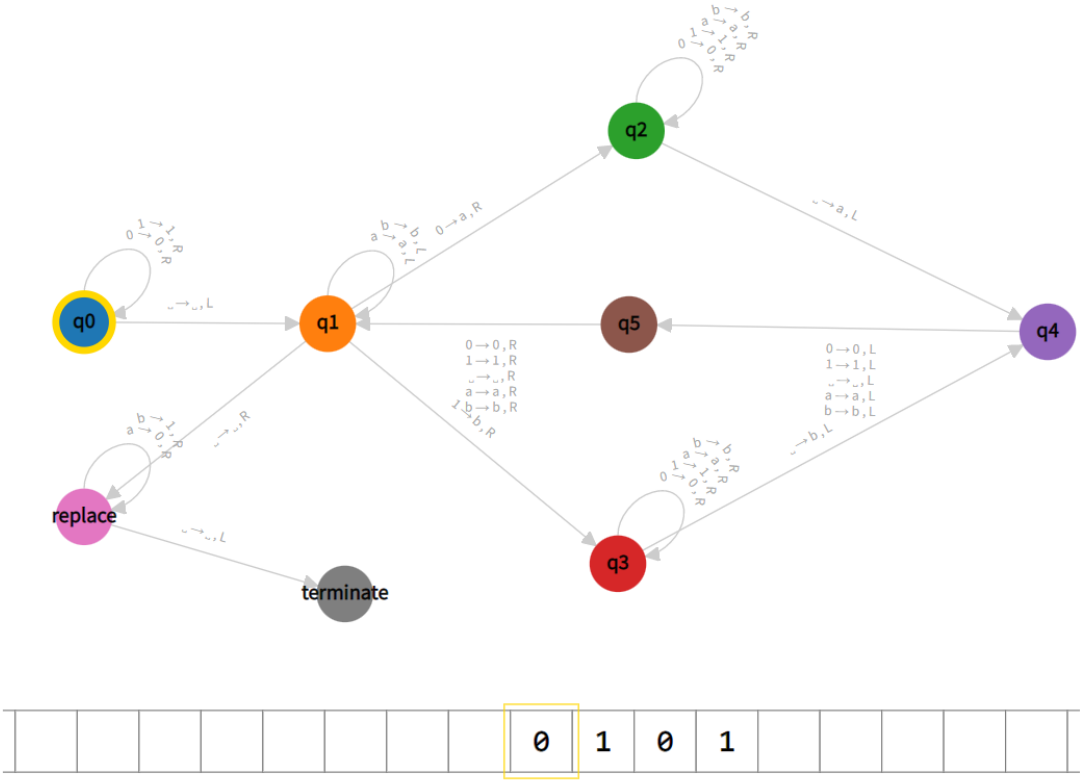

Figure 2.4.1: Initial state of the machine for the input **0101**

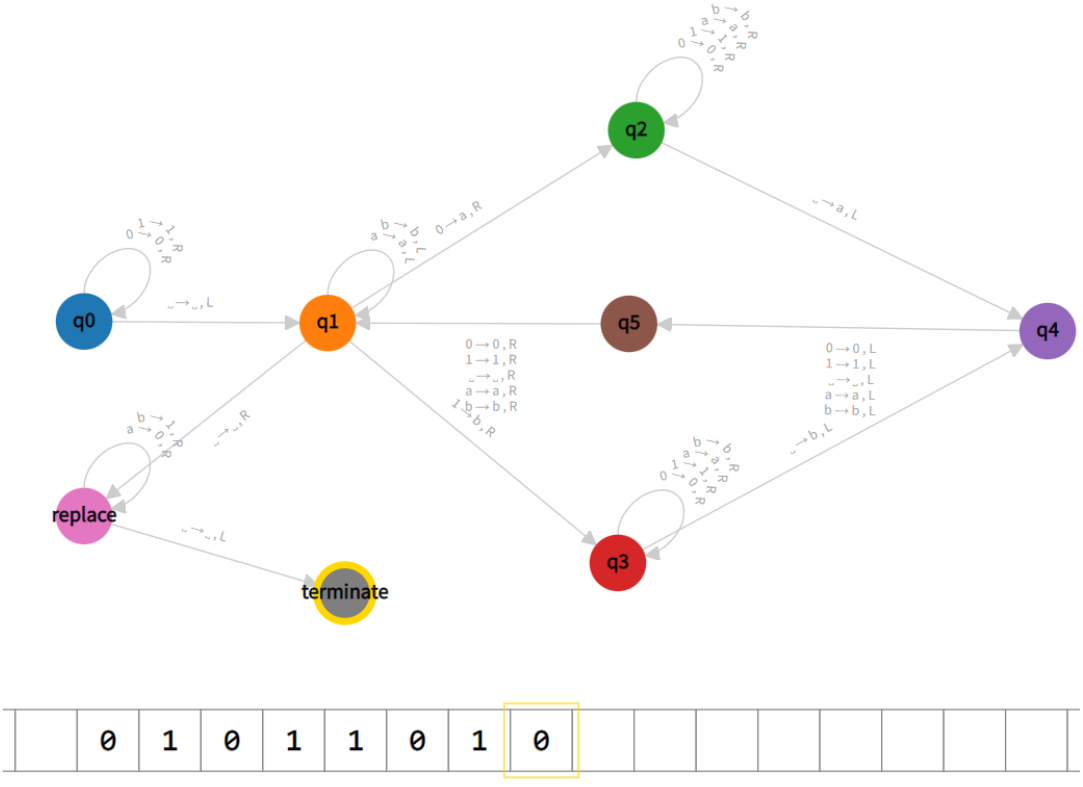

Figure 2.4.2: End state of the machine for the input **0101**

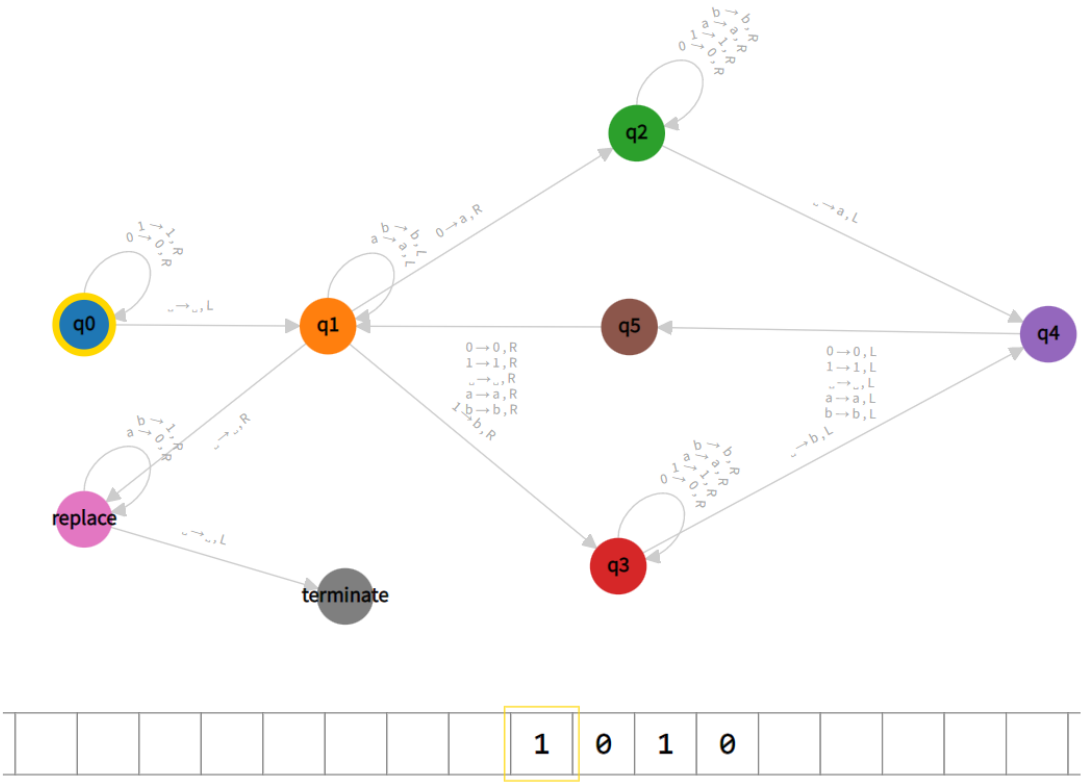

Figure 2.5.1: Initial state of the machine for the input **1010**

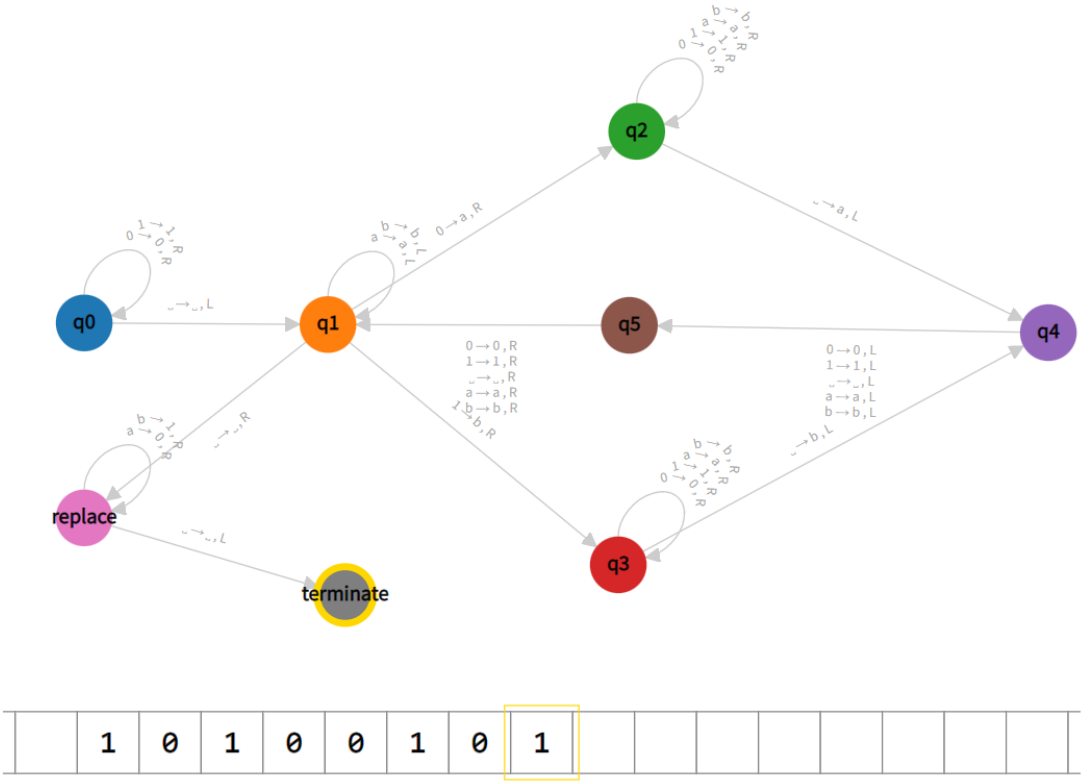

Figure 2.5.2: End state of the machine for the input **1010**

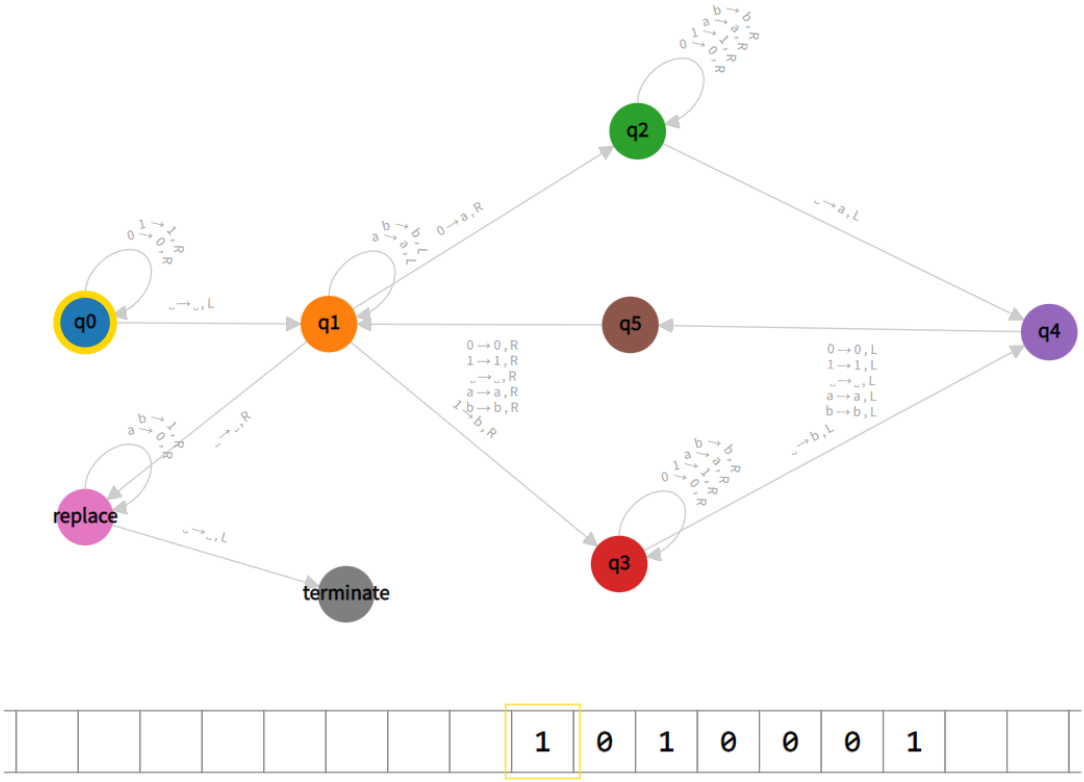

Figure 2.6.1: Initial state of the machine for the input **1010001**

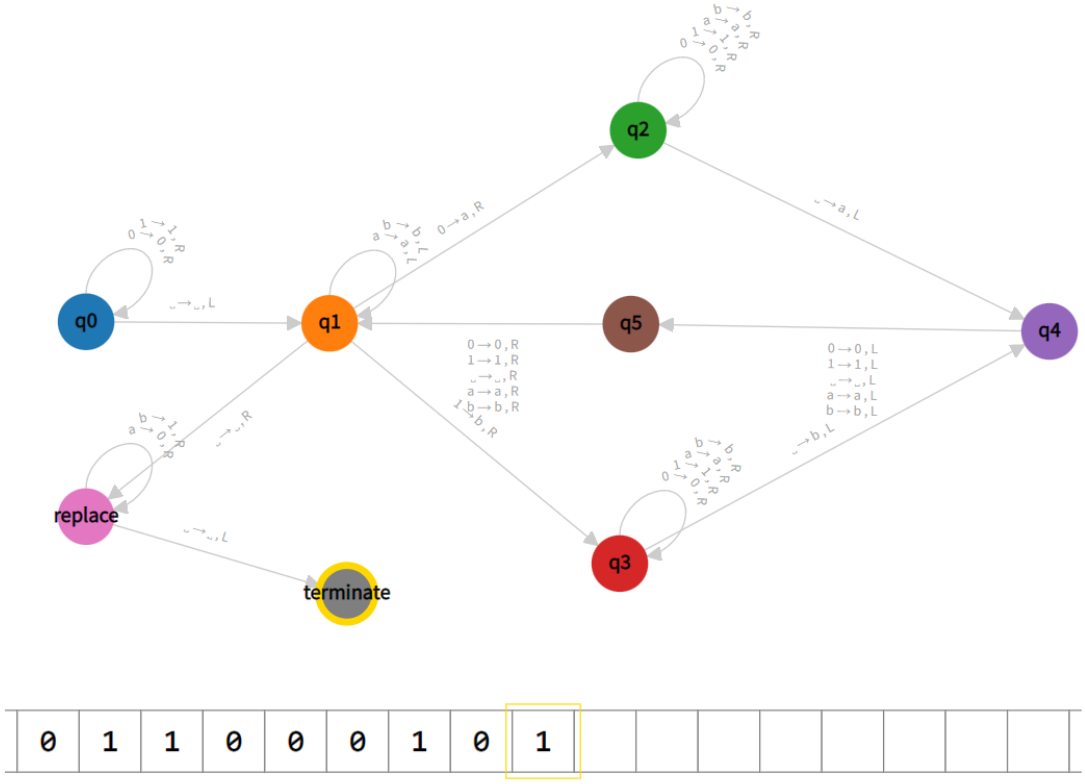

Figure 2.6.2: End state of the machine for the input **1010001**

Whole tape couldn't fit into the screenshot. This is the part of the tape which can be seen to us. If you want to see the whole tape, you can see on the simulator.

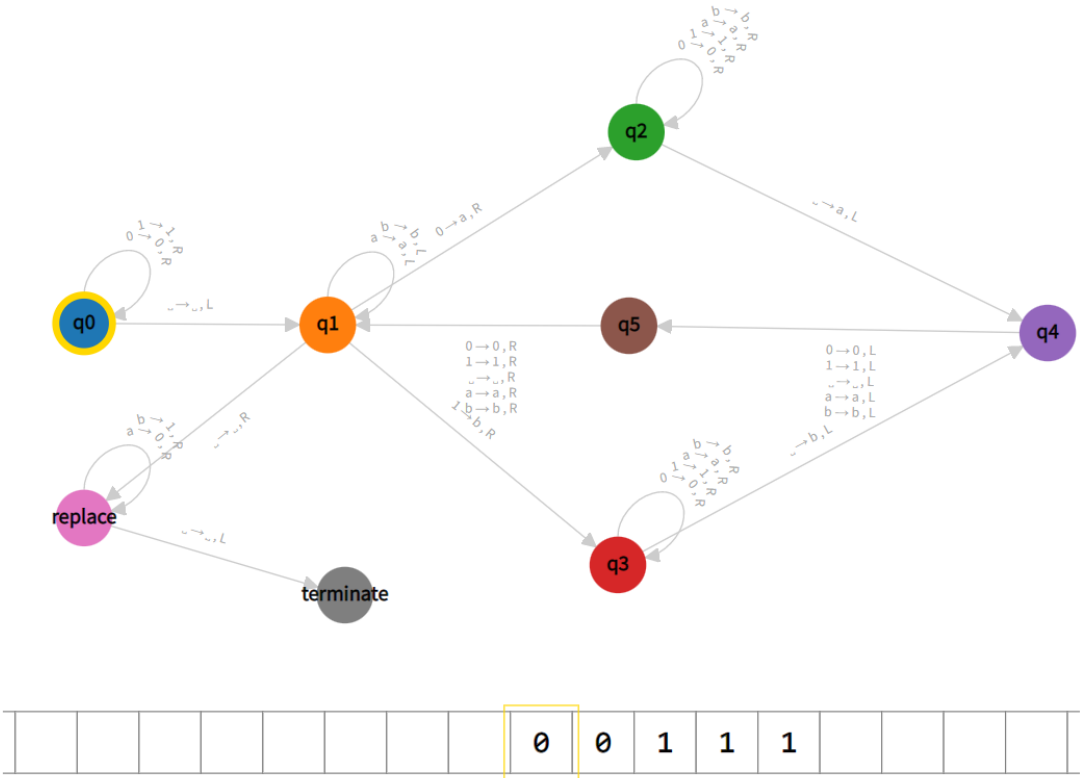

Figure 2.7.1: Initial state of the machine for the input **00111**

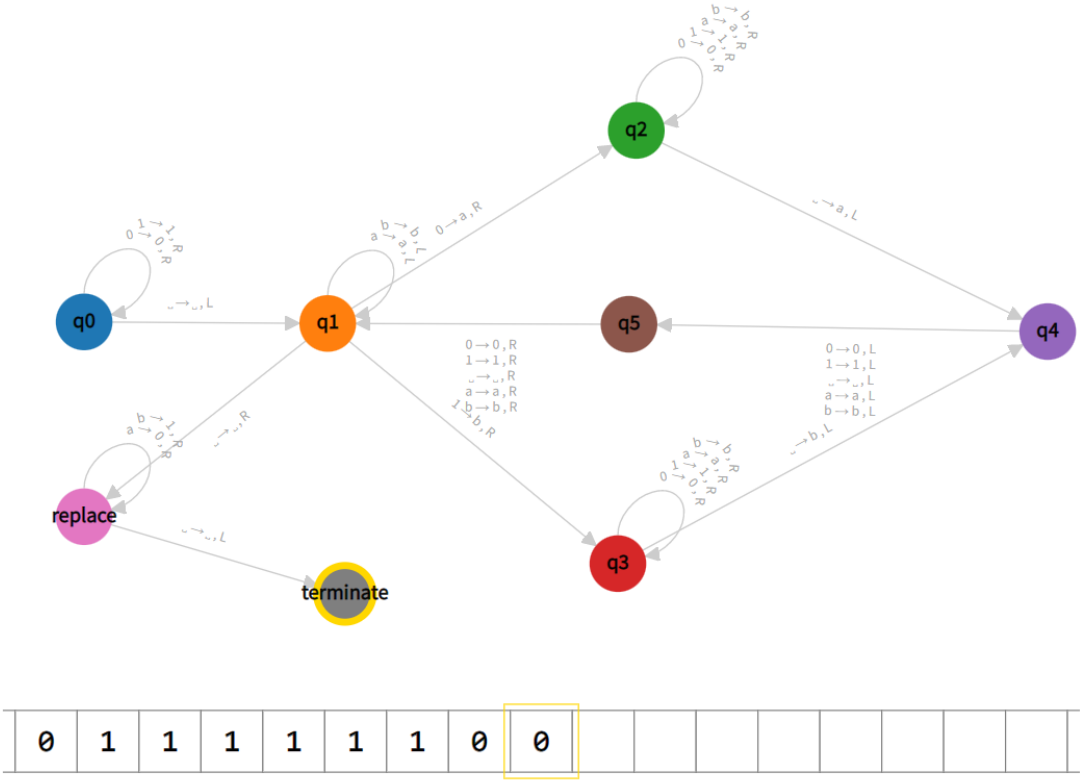

Figure 2.7.2: End state of the machine for the input **00111**

Whole tape couldn't fit into the screenshot. This is the part of the tape which can be seen to us. If you want to see the whole tape, you can see on the simulator.

Answer 3)

Formal Definition of the Machine:

A 2-dimensional Turing Machine is a quintuple $M = (K, \Sigma, \delta, s, H)$ where

* K is a finite set of states.

* Σ is an alphabet containing the blank symbol $\sqcup$ and the left-end symbol $\triangleright$ for marking the left end of the tape, and the bottom symbol $\triangle$ for marking the bottom of the tape.

* $s \in K$ is the start state.

* $H \subseteq K$ is the set of halting states.

* δ is the transition function defined from $(K - H) \times \Sigma$ to $K \times (\Sigma \cup \{L, U, R, D\})$ where:

    * L stands for going left on the 2D-tape.
    * U stands for going up on the 2D-tape.
    * R stands for going right on the 2D-tape.
    * D stands for going down on the 2D-tape.

such that;

* i) For all $q \in (K - H)$, if $\delta(q, \triangleright) = (p, b)$, then $b = R$.
* ii) For all $q \in (K - H)$, if $\delta(q, \triangle) = (p, b)$, then $b = U$.
* iii) For all $q \in (K - H)$ and $a \in \Sigma$, if $\delta(q, a) = (p, b)$, then $b \neq \triangleright$.
* iv) For all $q \in (K - H)$ and $a \in \Sigma$, if $\delta(q, a) = (p, b)$, then $b \neq \triangle$.

Formal Definition of a Configuration of the Machine:

A configuration of a 2-dimensional Turing Machine is a member of the cartesian product $Q \times N \times N \times T$ where Q is the set of states, N is the set of natural numbers, and T is the set of functions defined from $N \times N$ to Σ such that:

i) $t(0, y) = \triangleright$, for all $y \in N$.
ii) $t(x, 0) = \triangle$, for all $x \in N$ and $x > 0$.

Hence, a configuration consists of the current state of the machine, current head position, i.e two indices on the 2-D grid (one for columns, one for rows), and non-blank squares on the 2-D grid which are embedded into the functions $t \in T$.

Formal Definition of a Computation of the Machine:

A computation of a 2-dimensional Turing Machine can be defined as follows using entail $\vdash$ symbol:

$(q_1, x_1, y_1, t_1) \vdash_M (q_2, x_2, y_2, t_2)$ if there exists a transition $\delta(q_1, t_1(x_1, y_1)) = (q_2, \sigma)$ satisfying one of the following:

* $x_1 = x_2$, $y_2 = y_1 - 1$, $t_1 = t_2$, and $\sigma = L$.
* $y_1 = y_2$, $x_2 = x_1 + 1$, $t_1 = t_2$, and $\sigma = U$.
* $x_1 = x_2$, $y_2 = y_1 + 1$, $t_1 = t_2$, and $\sigma = R$.
* $y_1 = y_2$, $x_2 = x_1 + 1$, $t_1 = t_2$, and $\sigma = D$.
* $x_1 = x_2, y_1 = y_2, t_2(x_1, y_1) = \sigma, t_2(x, y,) = t_1(x, y)$ for all (x, y) pairs, and $\sigma \notin \{L, U, R, D\}$.

Meaning for the machine to decide a language:

Meaning for a 2-dimensional Turing Machine to decide language L can be defined as follows:

For a string w, let $t \in T$ be the function such that:

i) $t(j + 1, j) = w[j]$ for $1 \leq j \leq |w|$
ii) $t(0, y) = \triangleright$ for y $\in N$
iii) $t(x, 0) = \triangle$ for all $x > 0$
iv) $t(x, y) = \llcorner$ for all other (x, y) pairs

If a Turing Machine with 2-D tape M has two different halting states $h_{accept}, h_{reject} \in H$ such that for any string w, one of the following probabilities occur:

i) $(s, 1, 1, t) \vdash_M (h_{accept}, n, m, t')$ for some $n, m \in N$ and $t' \in T$
ii) $(s, 1, 1, t) \vdash_M (h_{reject}, n, m, t')$ for some $n, m \in N$ and $t' \in T$

where s: start symbol

Since, any string $w \in L$ can be either accepted or reject (not infinite loop) by the machine M, we say that the machine M decides the language L.

Proof that t steps of the machine, starting on an input of length n, can be simulated by a standard Turing Machine in time that is polynomial in t and n:

Part I: First, we will show that a Turing Machine with a 2-dimensional tape can be simulated by a standard Turing Machine.

Recall that from the formal definition of the Turing Machine with 2D tape, any $t \in T$ includes three information x : row index, y : column index, σ: symbol on the x^{th} row y^{th} column of the 2D-tape. So, we can represent any t as a 3-tuple (x, y, σ) . There exists a unique σ for (x, y) pairs, since t is a function. Therefore, we can linearize 2D-tape into 1D-tape such that t(i,j) is stored in the $\frac{1}{2}[(i+j)^2 + 3i + j]^{th}$ square of the 1D-tape.

Therefore, we are able to simulate 2D-tape Turing Machine M with a three tape 1D-tape Turing Machine M'. M' uses one of its tapes for getting input, one is for calculation, one is for encoding of 2D array of M into 1D array. M' makes the calculations on the first tape to keep track of M's head position on its tape by incrementing, decrementing and other arithmetics with indices i and j.

M' works as follows:

1. It starts with converting 2D-tape of M into its 1D-tape.
2. It moves by updating i and j indices.
3. It reads the input tape and writes to the tape according to the i and j indices.

Now, we achieved to simulate a Turing Machine with 2D-tape by a three 1D-tape Turing Machine. Then, we need to show that this three 1D-tape Turing Machine (actually k-tape Turing Machine k ¿ 1) can be simulated by single tape standard Turing Machine.

Let G be a multi-tape Turing Machine with k-tapes and Sbe a single tape standard Turing Machine. Then S simulates the function of k tapes by storing their information on a single tape side by side. It uses a new symbol as a delimeter to seperate the contents of the k-tapes. In addition to the contents of these tapes, S must keep track of the location of head of G. It does so by writing a tape symbol with a marking symbol it to mark the place where the head left. Then, it can successfully simulate G.

We have proven that 3-tape Turing Machine which can simulate 2D-tape Turing Machine can be simulated by standard Turing Machine.

Hence, a Turing Machine with a 2D-tape can be simulated by a standard Turing Machine.

Part II: Second, we will show that t steps of the 2D-tape Turing Machine, starting on an input of length n, can be simulated by a standard Turing machine in time that is polynomial in t and n.:

At any given time, if indices i and j represent the largest x and y coordinates of the head of M on 2D-tape, then each of i and j cannot be larger than t. Thus, simulating one moves of M on 2D-grid depends on incrementing , decrementing and some arithmetics like summation and multiplication on i and j thus it is bounded by second degree polynomial in t. The increments, decrements and arithmetic operations can be done in first degree polynomial in t. As a result, the time to carry out t steps, after initial setup of the simulation, is $O(t^3)$ .The initial setup of the machine M' can be considered as 2n-step of computation of moves and writes, so again it can be carried out in time $O(n^3)$. Hence, the simulation functions in time polynomial in t and n.