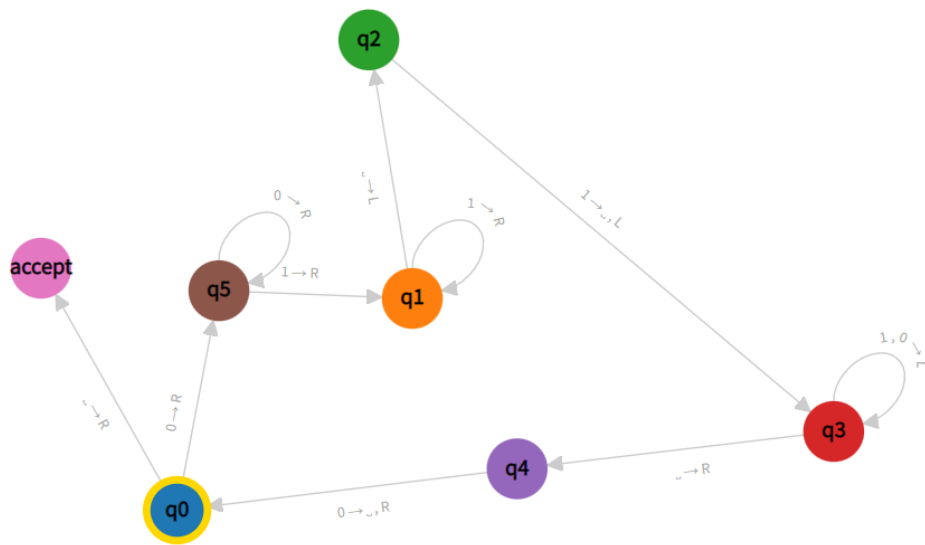


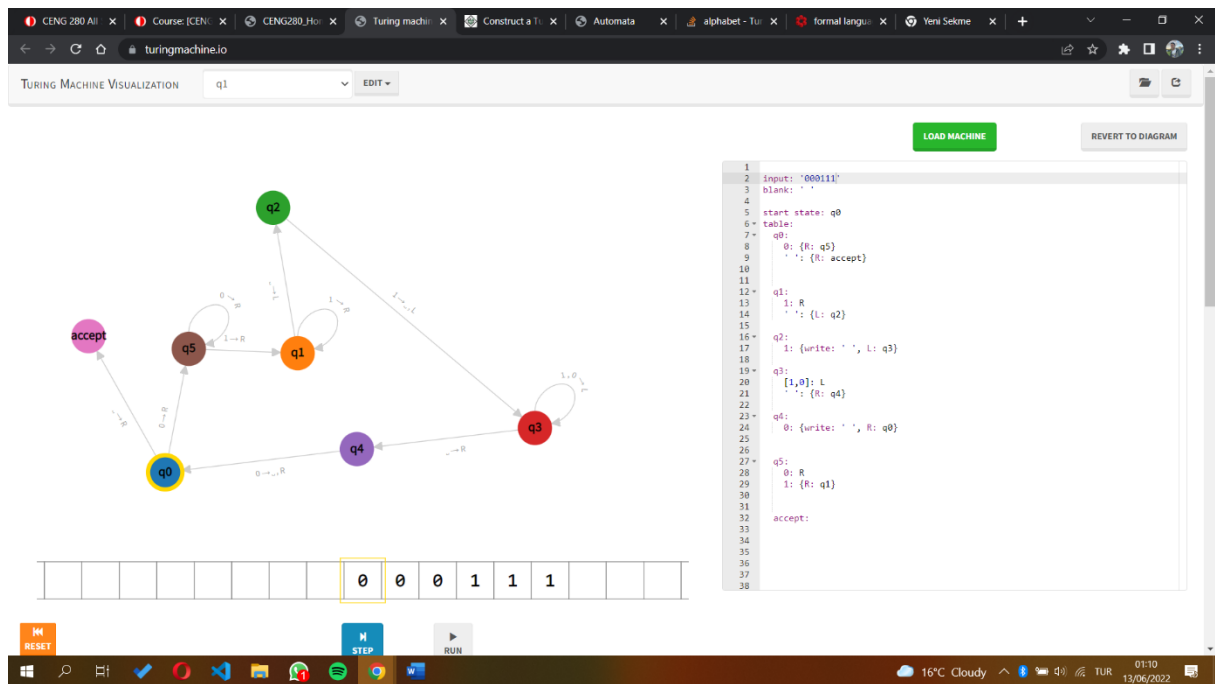
**Name:** Cem Meriç Şefikoğulları

**ID:** 2448850

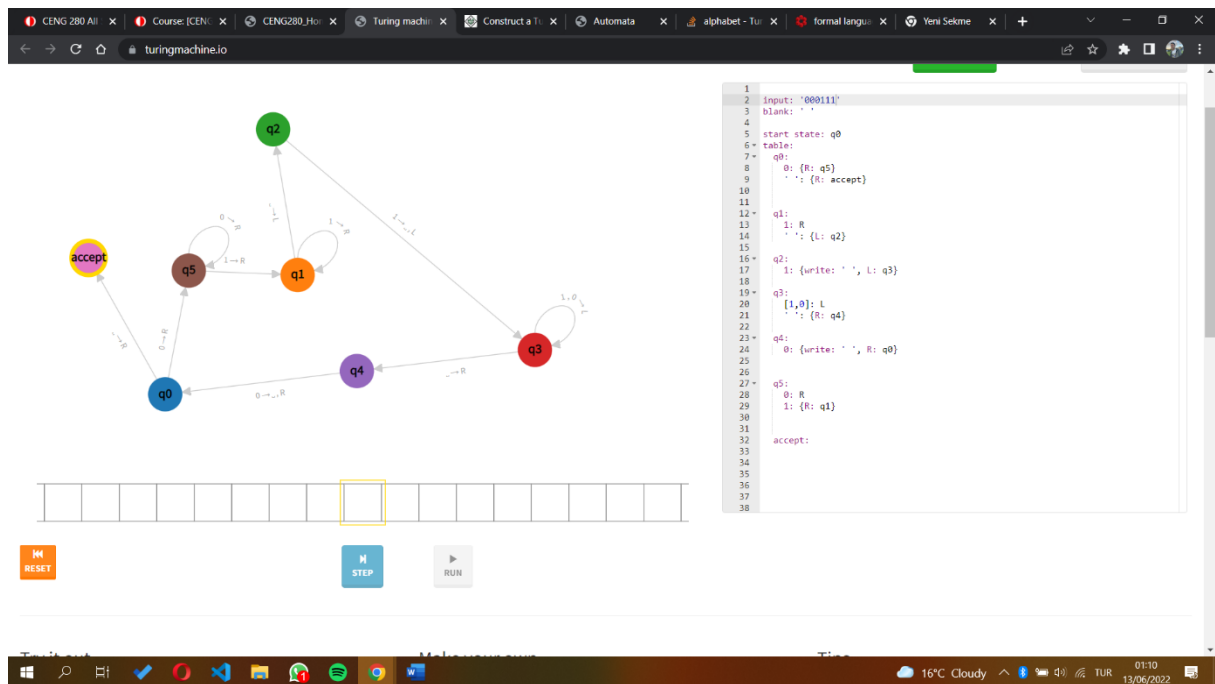
**Q1)**



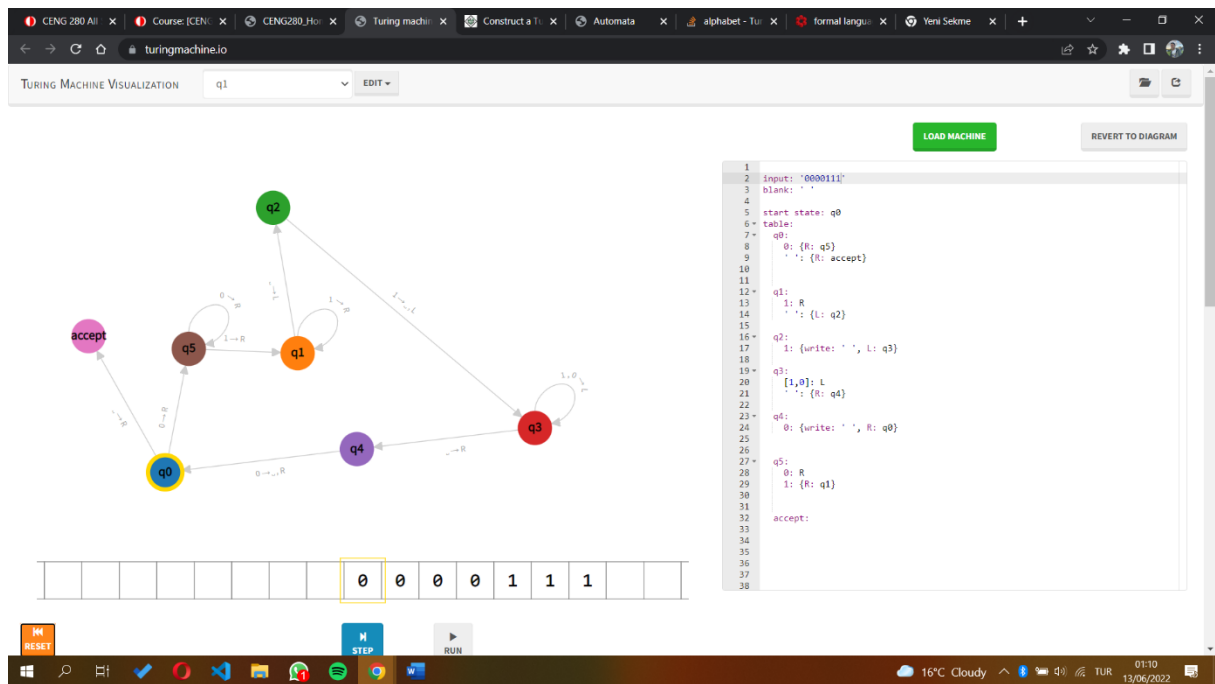
**Figure of machine**



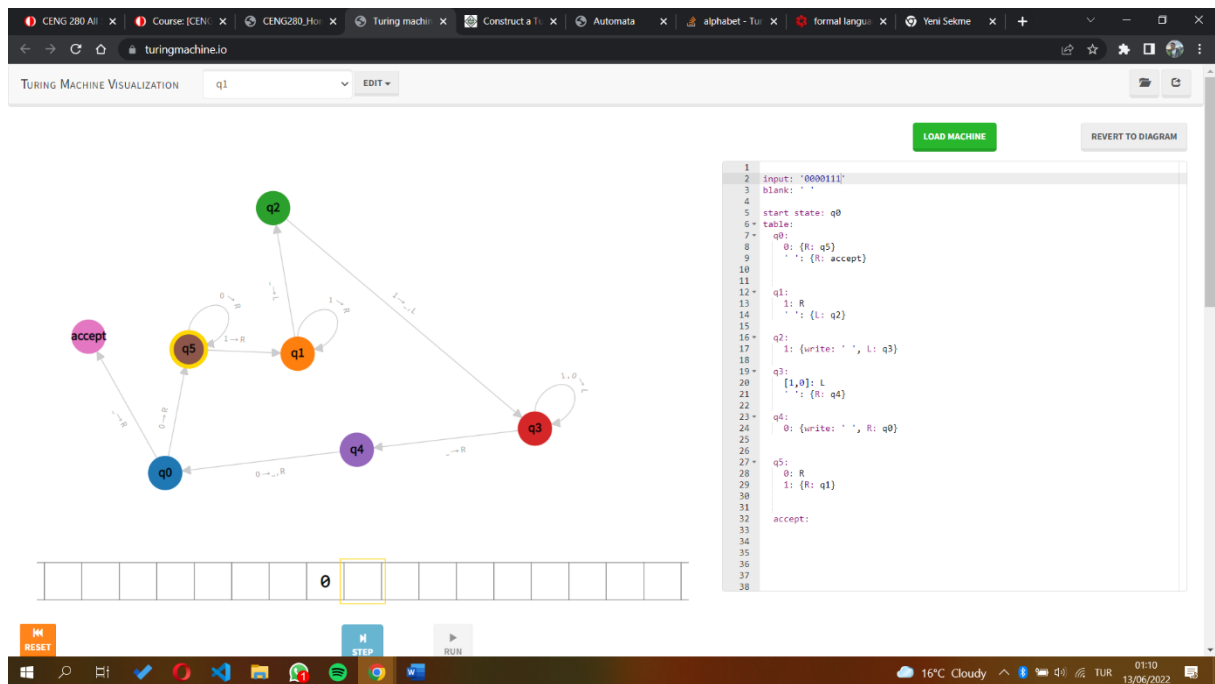
Initial state: 000111,q0



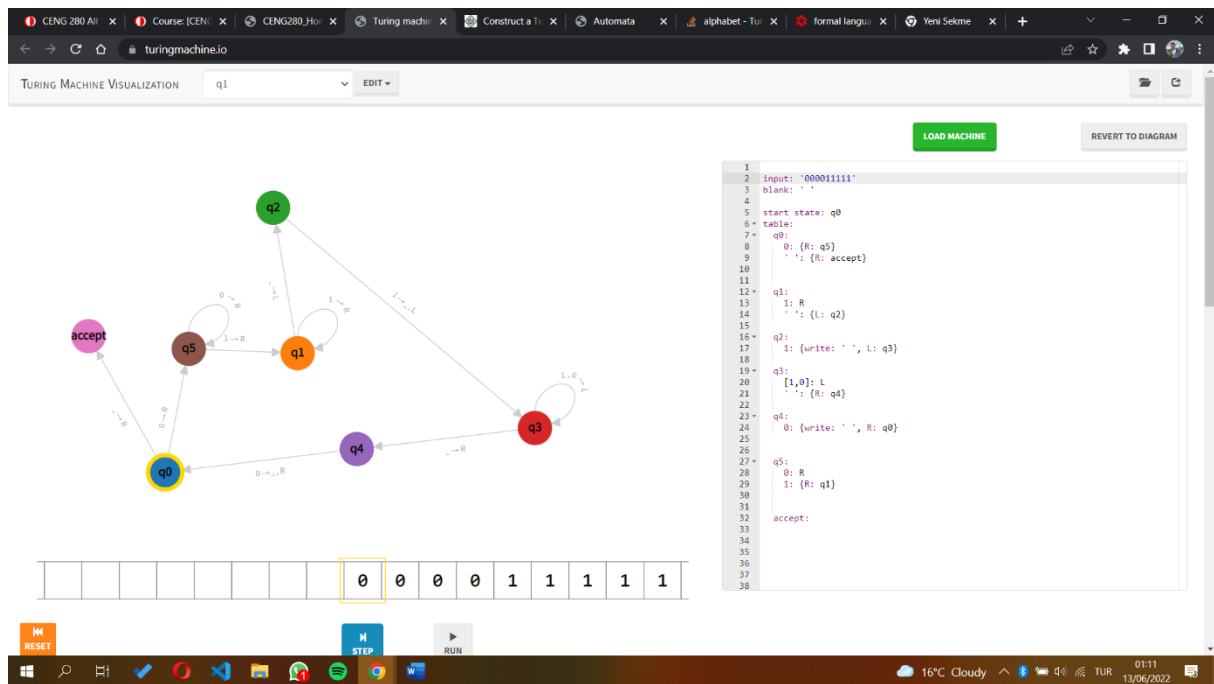
End state: \_accept state accepted



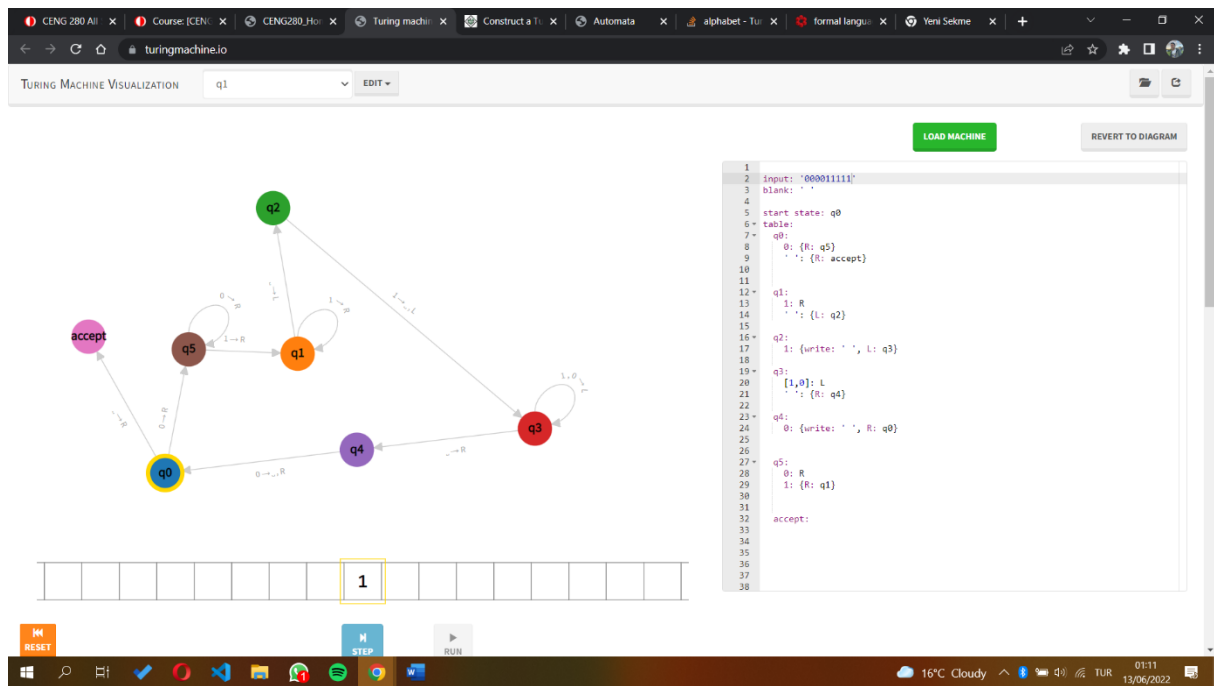
Initial state:0000111,q0



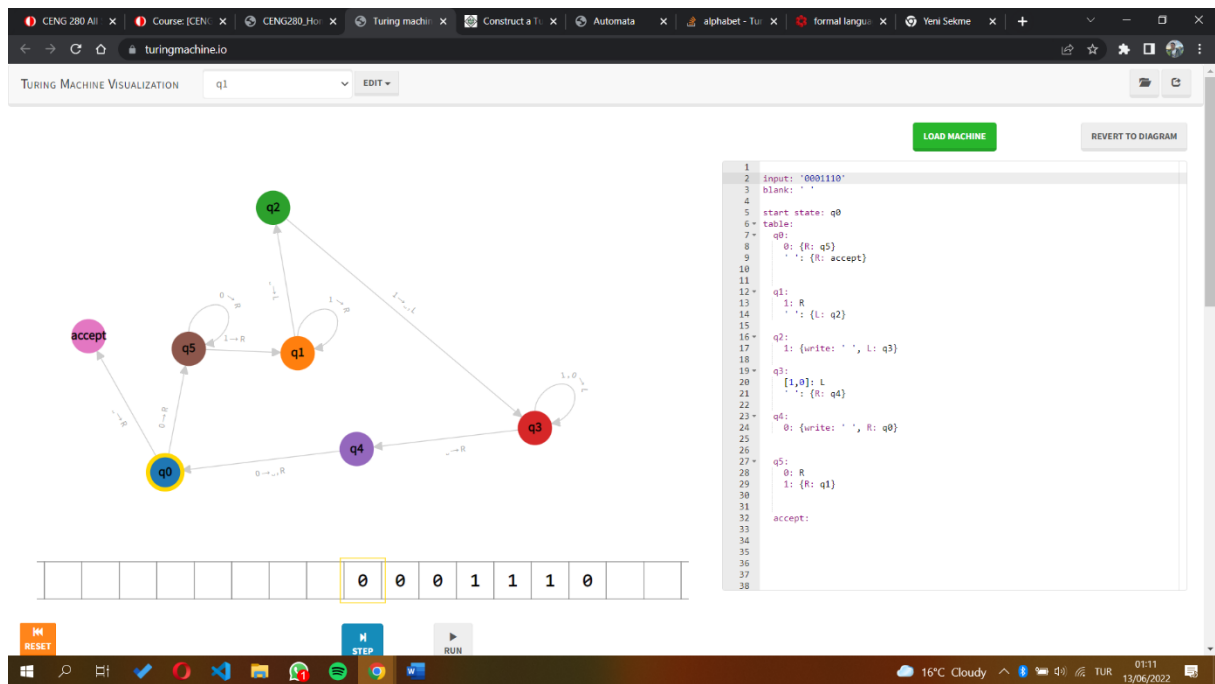
End state: 0,q5 rejected



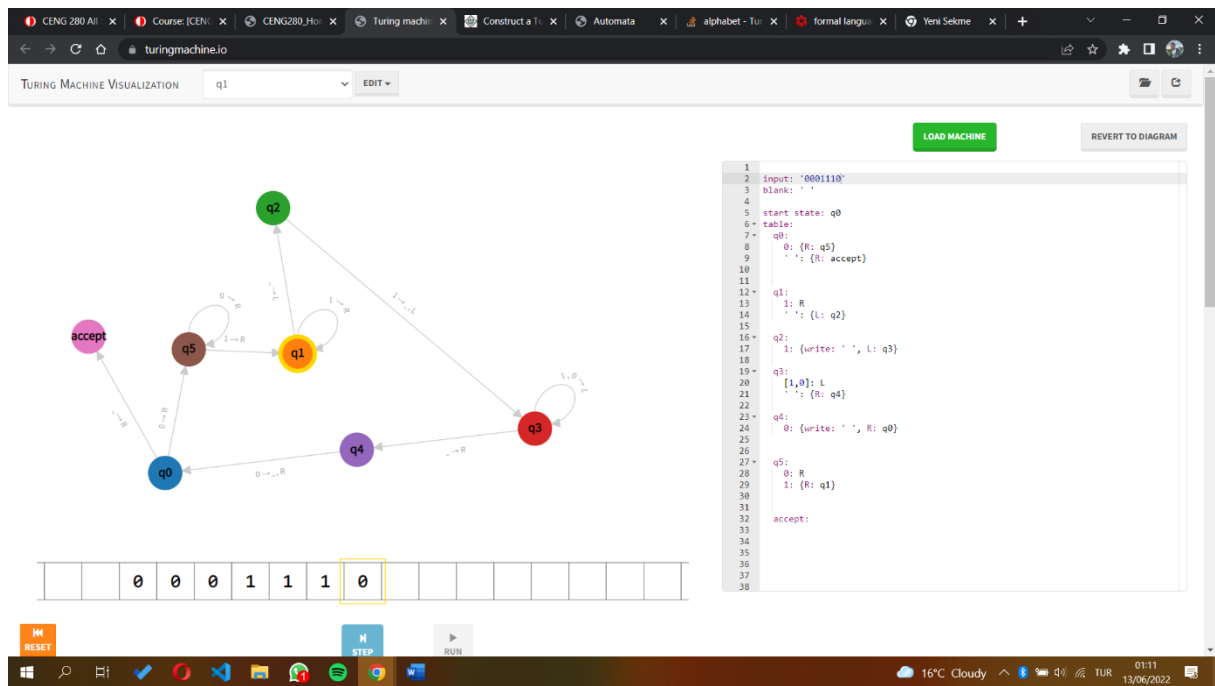
Initial state: 000011111,q0



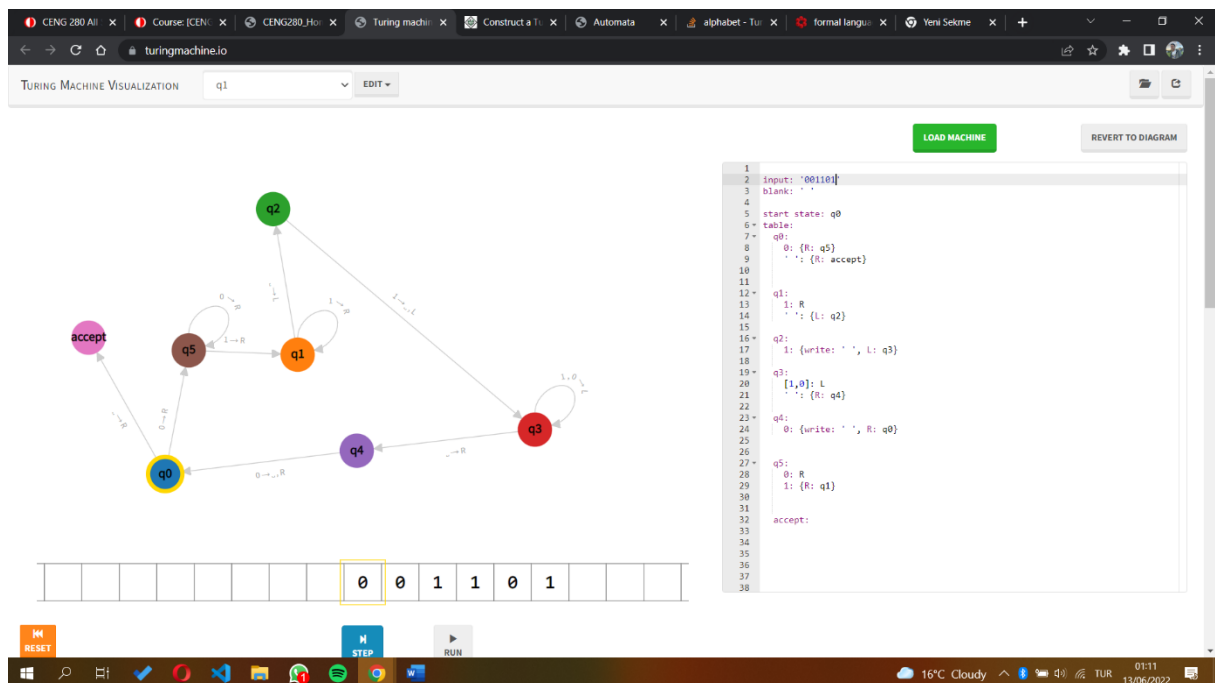
End state: 1,q0 rejected



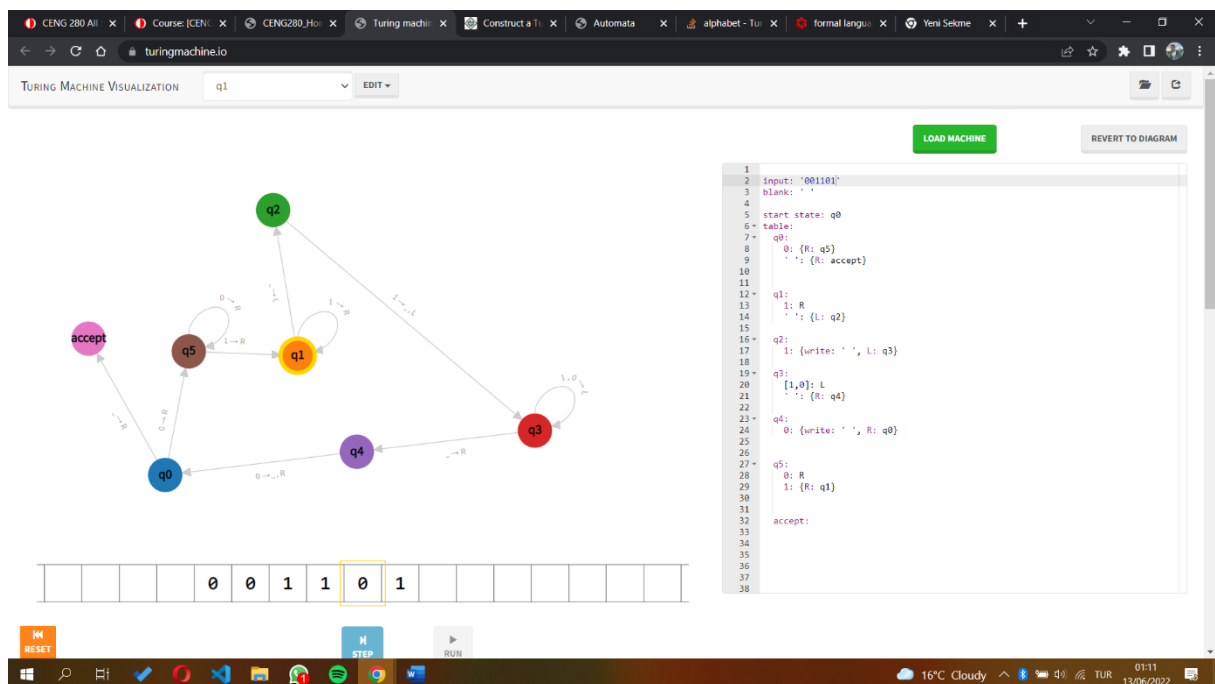
Initial state: 0001110 , q0



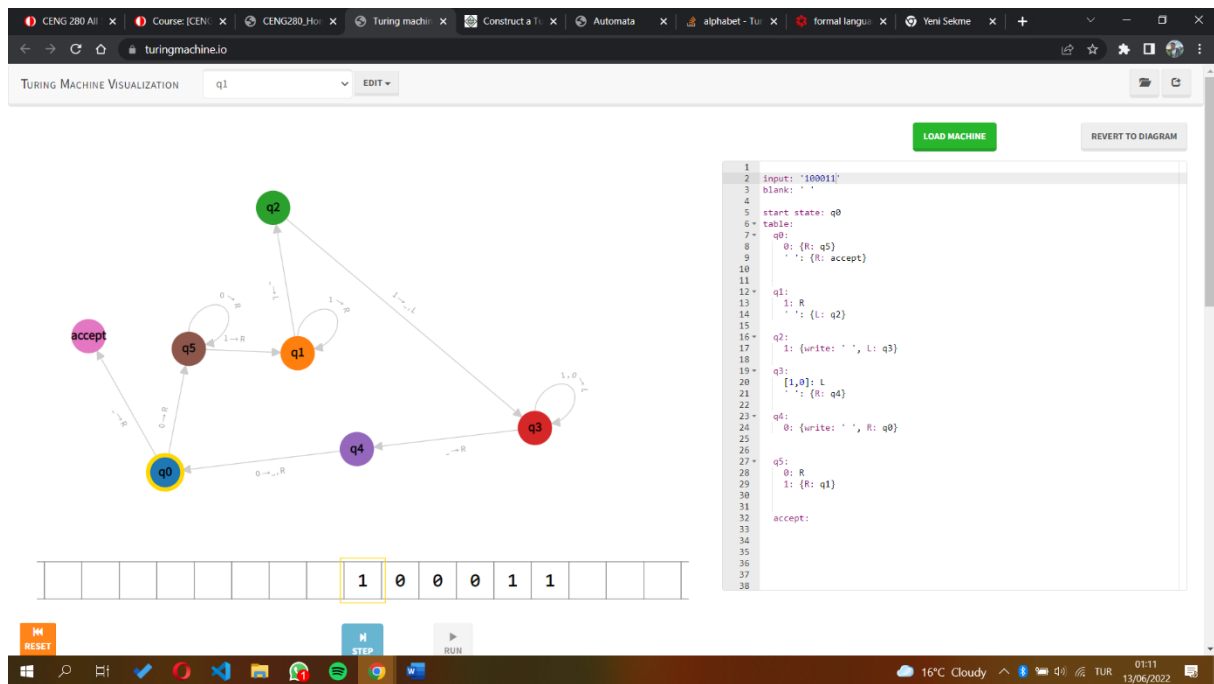
End state: 0001110, q1 rejected



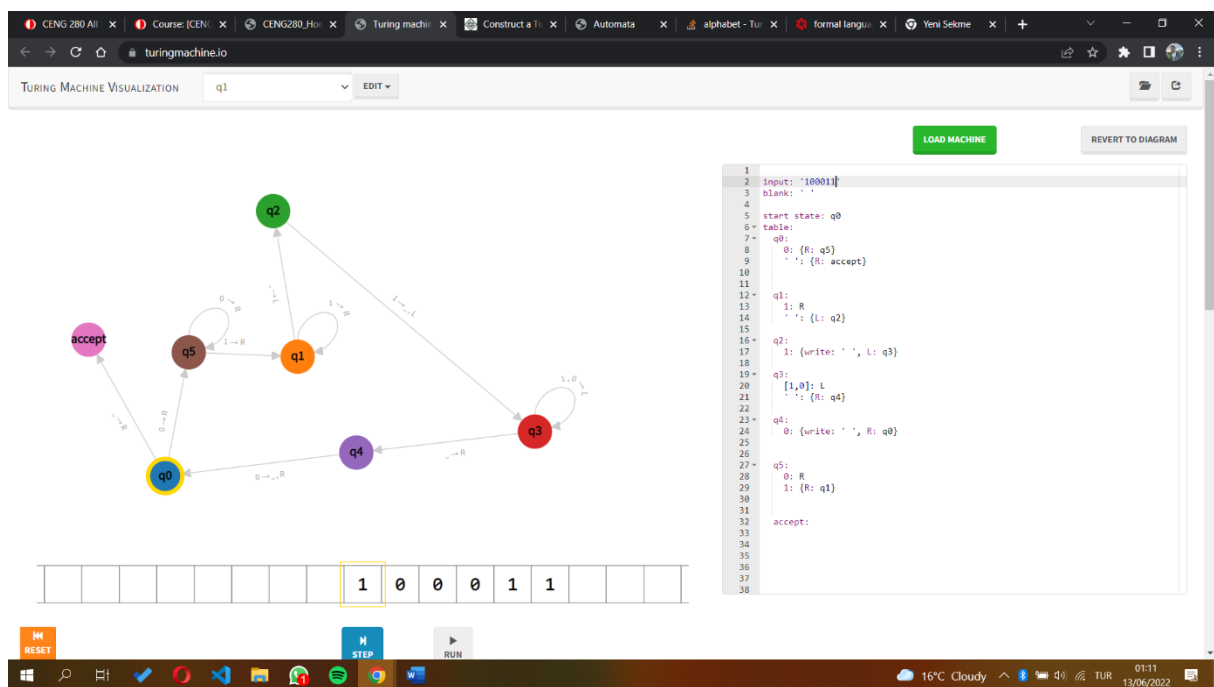
Initial state:001101, q0



End state: 001101,q1 rejected



Initial state: 100011, q0



End state: 100011, q0 rejected

Description of the states:

**Start State q0:**

If symbol 0 move right go to the q5 state

If symbol blank move right go to accept state

**State q1:**

If symbol q1 move right stay in the q1 state

If symbol blank move left go to the q2 state

**State q2:**

If symbol 1 replace it by blank move left go to the q3 state

**State q3:**

If symbol 1 or 0 move left stay in the q3 state

If symbol blank move right go to the q4 state

**State q4:**

If symbol 0 replace it by blank move right go to the q0 state

**State q5:**

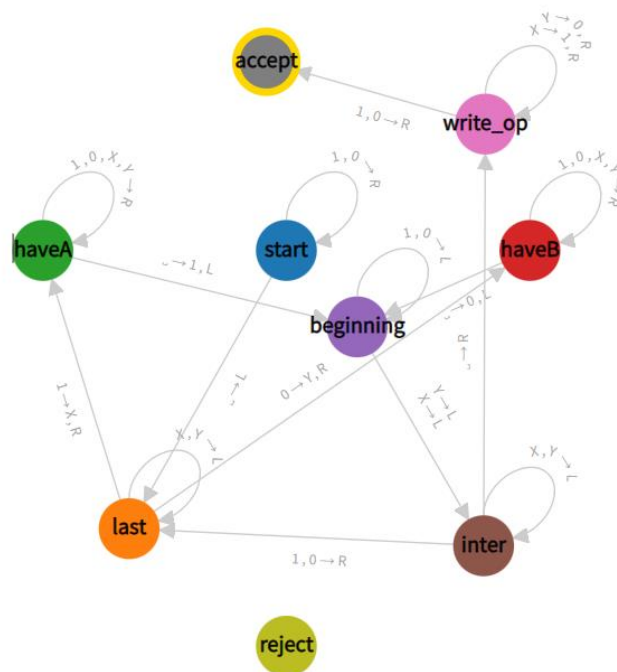
If symbol 0 move right stay in the q5 state

If symbol 1 move right go to the state q1

Accept state:

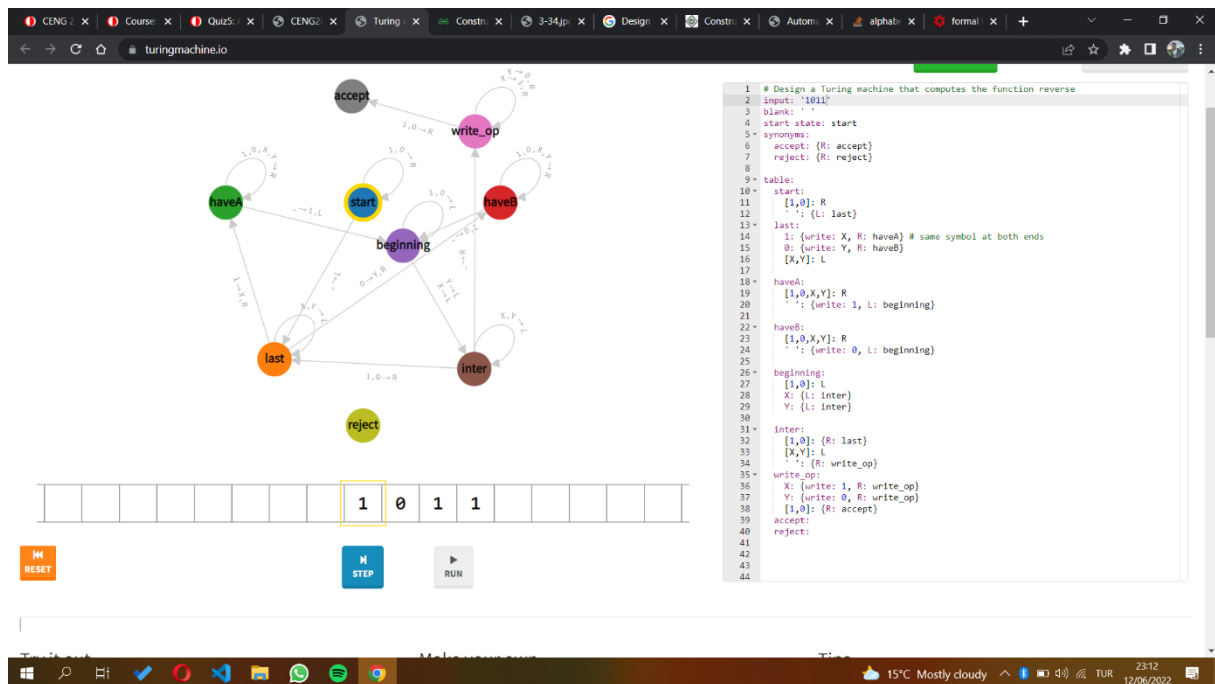
Accept the string

**Q2)**

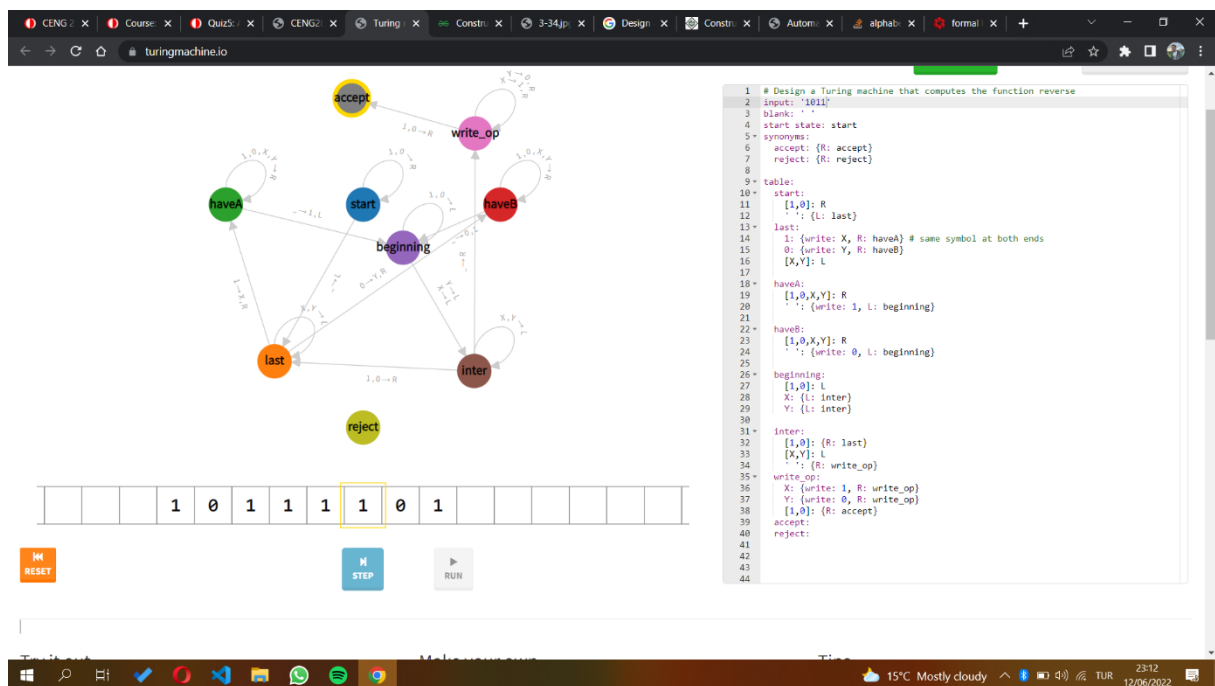




## Figure of Machine



Initial state 1011



End state: 1011101

The screenshot shows the Turing Machine interface with the initial state '1110'. The tape contains the sequence 1 1 1 0. The state transition diagram on the left shows the machine's logic, including states like 'start', 'beginning', 'last', 'inter', 'write\_op', 'haveA', 'haveB', 'accept', and 'reject'. The code on the right defines the machine's configuration, including the input '1110', the start state, and the transition rules for each state.

```

1 # Design a Turing machine that computes the function reverse
2 input: '1110'
3 blank: ' '
4 start state: start
5 synonyms:
6   accept: {R: accept}
7   reject: {R: reject}
8
9 table:
10 start:
11   [1,0]: R
12   ': {L: last}
13 last:
14   1: {write: X, R: haveA} # same symbol at both ends
15   0: {write: Y, R: haveB}
16   [X,Y]: L
17
18 haveA:
19   [1,0,X,Y]: R
20   ': {write: 1, L: beginning}
21
22 haveB:
23   [1,0,X,Y]: R
24   ': {write: 0, L: beginning}
25
26 beginning:
27   [1,0]: L
28   X: {L: inter}
29   Y: {L: inter}
30
31 inter:
32   [1,0]: {R: last}
33   [X,Y]: L
34   ': {R: write_op}
35
36 write_op:
37   X: {write: 1, R: write_op}
38   Y: {write: 0, R: write_op}
39   [1,0]: {R: accept}
40
41 accept:
42
43 reject:
44

```

Initial state:1110

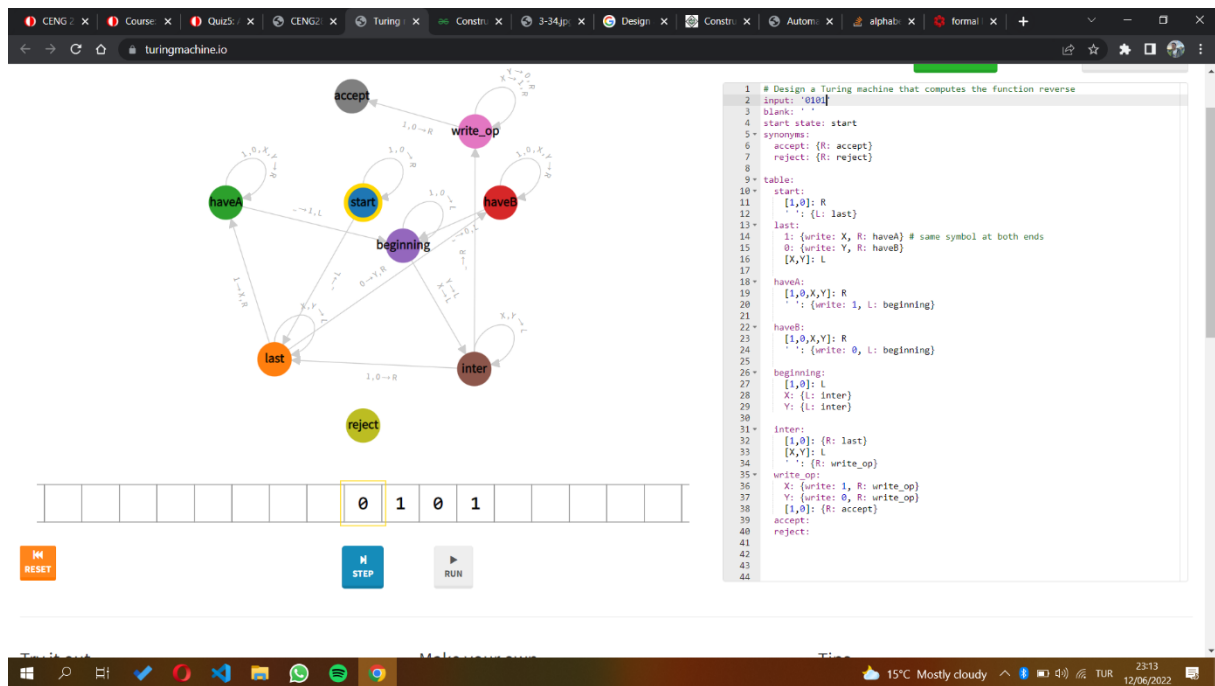
The screenshot shows the Turing Machine interface with the end state '11100111'. The tape contains the sequence 1 1 1 0 0 1 1 1. The state transition diagram on the left shows the machine's logic, including states like 'start', 'beginning', 'last', 'inter', 'write\_op', 'haveA', 'haveB', 'accept', and 'reject'. The code on the right defines the machine's configuration, including the input '1110', the start state, and the transition rules for each state.

```

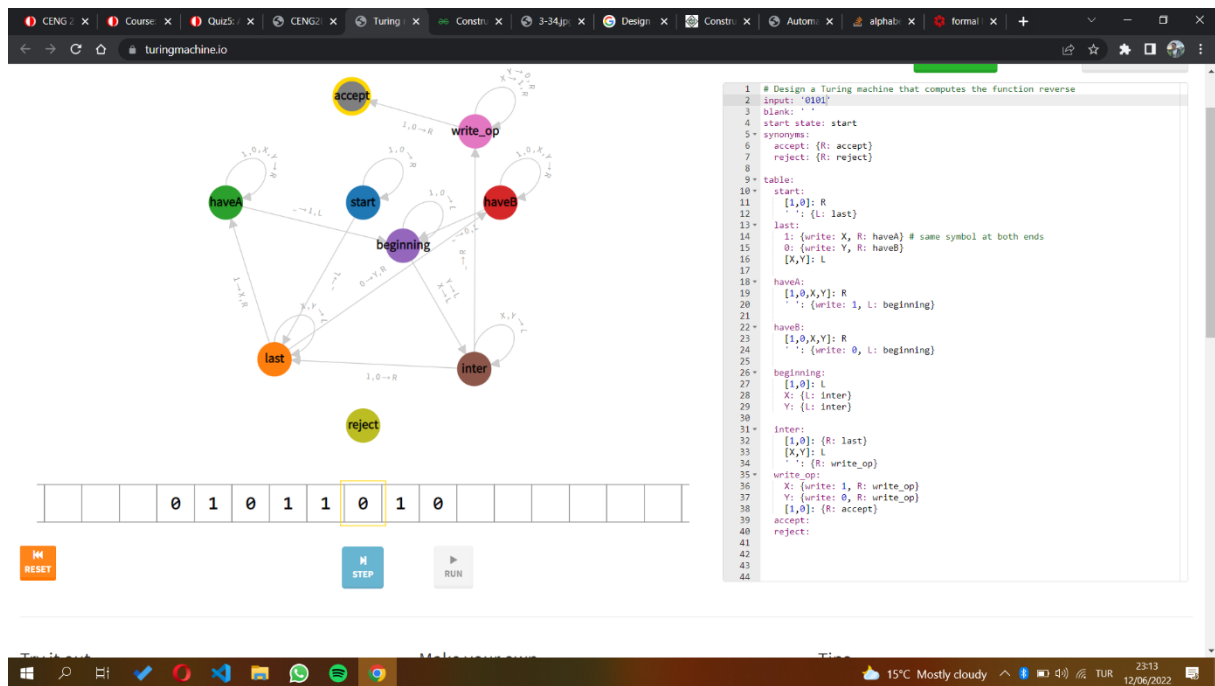
1 # Design a Turing machine that computes the function reverse
2 input: '1110'
3 blank: ' '
4 start state: start
5 synonyms:
6   accept: {R: accept}
7   reject: {R: reject}
8
9 table:
10 start:
11   [1,0]: R
12   ': {L: last}
13 last:
14   1: {write: X, R: haveA} # same symbol at both ends
15   0: {write: Y, R: haveB}
16   [X,Y]: L
17
18 haveA:
19   [1,0,X,Y]: R
20   ': {write: 1, L: beginning}
21
22 haveB:
23   [1,0,X,Y]: R
24   ': {write: 0, L: beginning}
25
26 beginning:
27   [1,0]: L
28   X: {L: inter}
29   Y: {L: inter}
30
31 inter:
32   [1,0]: {R: last}
33   [X,Y]: L
34   ': {R: write_op}
35
36 write_op:
37   X: {write: 1, R: write_op}
38   Y: {write: 0, R: write_op}
39   [1,0]: {R: accept}
40
41 accept:
42
43 reject:
44

```

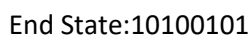
End state: 11100111



Initial state: 0101



End State: 01011010



```

1 # Design a Turing machine that computes the function reverse
2 input: '00111'
3 blank: ' '
4 start state: start
5 synonyms:
6   accept: {R: accept}
7   reject: {R: reject}
8
9 table:
10  start:
11    [1,0]: R
12    ': {L: last}
13  last:
14    1: {write: X, R: haveA} # same symbol at both ends
15    0: {write: Y, R: haveB}
16    [X,Y]: L
17  haveA:
18    [1,0,X,Y]: R
19    ': {write: 1, L: beginning}
20  haveB:
21    [1,0,X,Y]: R
22    ': {write: 0, L: beginning}
23  beginning:
24    [1,0]: L
25    X: {L: inter}
26    Y: {L: inter}
27  inter:
28    [1,0]: {R: last}
29    [X,Y]: L
30    ': {R: write_op}
31  write_op:
32    X: {write: 1, R: write_op}
33    Y: {write: 0, R: write_op}
34    [1,0]: {R: accept}
35  accept:
36    reject:
  
```

Initial State: 00111

Initial State: 00111

```

1 # Design a Turing machine that computes the function reverse
2 input: '00111'
3 blank: ' '
4 start state: start
5 synonyms:
6   accept: {R: accept}
7   reject: {R: reject}
8
9 table:
10  start:
11    [1,0]: R
12    ': {L: last}
13  last:
14    1: {write: X, R: haveA} # same symbol at both ends
15    0: {write: Y, R: haveB}
16    [X,Y]: L
17  haveA:
18    [1,0,X,Y]: R
19    ': {write: 1, L: beginning}
20  haveB:
21    [1,0,X,Y]: R
22    ': {write: 0, L: beginning}
23  beginning:
24    [1,0]: L
25    X: {L: inter}
26    Y: {L: inter}
27  inter:
28    [1,0]: {R: last}
29    [X,Y]: L
30    ': {R: write_op}
31  write_op:
32    X: {write: 1, R: write_op}
33    Y: {write: 0, R: write_op}
34    [1,0]: {R: accept}
35  accept:
36    reject:
  
```

End State: 001111100

End State: 001111100

Initial State: 1010001

```

1 # Design a Turing machine that computes the function reverse
2 input: '1010001'
3 blank: ' '
4 start state: start
5 synonyms:
6   accept: {R: accept}
7   reject: {R: reject}
8
9 table:
10  start:
11    [1,0]: R
12    ' ': {L: last}
13  last:
14    1: {write: X, R: haveA} # same symbol at both ends
15    0: {write: Y, R: haveB}
16    [X,Y]: L
17  haveA:
18    [1,0,X,Y]: R
19    ' ': {write: 1, L: beginning}
20  haveB:
21    [1,0,X,Y]: R
22    ' ': {write: 0, L: beginning}
23  beginning:
24    [1,0]: L
25    X: {L: inter}
26    Y: {L: inter}
27  inter:
28    [1,0]: {R: last}
29    [X,Y]: L
30    ' ': {R: write_op}
31  write_op:
32    X: {write: 1, R: write_op}
33    Y: {write: 0, R: write_op}
34    [1,0]: {R: accept}
35  accept:
36    reject:
  
```

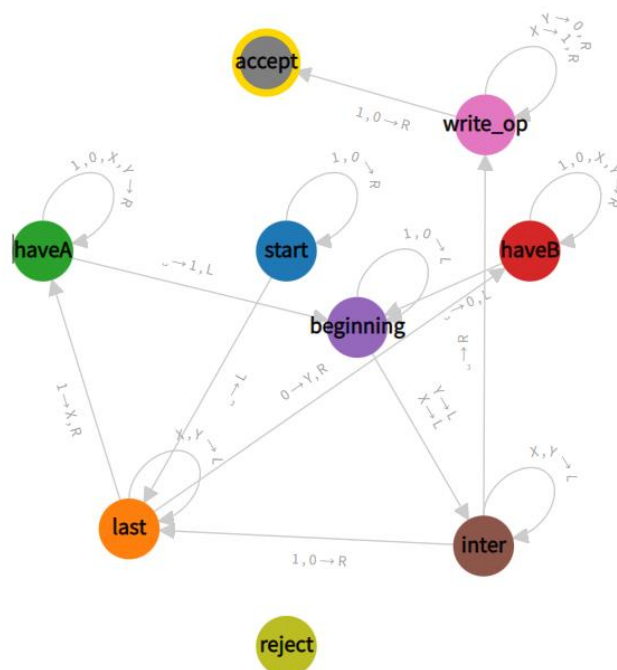
Initial State: 1010001

End State: 10100011000101

```

1 # Design a Turing machine that computes the function reverse
2 input: '1010001'
3 blank: ' '
4 start state: start
5 synonyms:
6   accept: {R: accept}
7   reject: {R: reject}
8
9 table:
10  start:
11    [1,0]: R
12    ' ': {L: last}
13  last:
14    1: {write: X, R: haveA} # same symbol at both ends
15    0: {write: Y, R: haveB}
16    [X,Y]: L
17  haveA:
18    [1,0,X,Y]: R
19    ' ': {write: 1, L: beginning}
20  haveB:
21    [1,0,X,Y]: R
22    ' ': {write: 0, L: beginning}
23  beginning:
24    [1,0]: L
25    X: {L: inter}
26    Y: {L: inter}
27  inter:
28    [1,0]: {R: last}
29    [X,Y]: L
30    ' ': {R: write_op}
31  write_op:
32    X: {write: 1, R: write_op}
33    Y: {write: 0, R: write_op}
34    [1,0]: {R: accept}
35  accept:
36    reject:
  
```

End State: 10100011000101



### Description of states:

#### **Start State:**

If symbol 1 or 0 move right, stay in the start state

If symbol blank move left, go to state last

#### **Last State:**

If symbol is 1, replace it by X and move right go to the haveA state

If symbol is 0, replace it by Y and move right go to the haveB state

If symbol X or Y move to the left stay in the last state

#### **haveA state:**

If symbol 1,0,X or Y move right, stay in the haveA state

If symbol blank replace it by 1 move left and go to the beginning state

#### **haveB state:**

If symbol 1,0,X or Y move right, stay in the haveB state

If symbol blank replace it by 0 move left and go to the beginning state

#### **Beginning state:**

If symbol 1 or 0 move left stay in the beginning state

If symbol is X or Y move left go to the inter state

#### Inter state:

If symbol 1 or 0 move right go tot the last state

If symbol X or Y, move left stay in the inter state

If symbol blank move right go to the write\_op state

#### Write\_op state:

If symbol X replace it by 1 move right stay in the write\_op state

If symbol Y replace it by 0 move right stay in the write\_op state

If symbol 1 or 0 move right, go to the Accept state

## Q3)

Turing machine with two dimensional tape, have one finite control, one read-write head and one two dimensional tape. It has top and left ends, and they goes like right and down. The machine has subparts as rows of small squares.

v	v	v	v	....
h	1	2	6	...
h	3	5	8	..
h	4	9	13	...
...	...	...	..	..

This machine is a pentuple  $M=(K,\Sigma,\delta,s,H)$ ,

$\delta$  function is from  $K \times \Sigma$  to  $K \times (\Sigma \cup \{ \rightarrow, \downarrow, \leftarrow, \uparrow \})$ , and  $\delta(q_1, \triangleright) = (q_2, \rightarrow)$  ,  $\delta(q_1, \Delta) = (q_2, \uparrow)$  for all  $q_1$

Configuration is :  $K \times N \times N \times N \times T$  where S is set of functions form  $N \times N$  to  $\Sigma$

Configuration is represented by current state, current head position, list of all non-blank squares on the tape.

$(q_1, a_1, b_1, z_1) \vdash_M (q_2, a_2, b_2, z_2)$  This holds

if  $\delta(q_1, z_1(a_1, b_1)) = (q_2, \#)$  and one of the below

$a_1 = a_2, b_1 + 1 = b_2, z_1 = z_2$ , and  $\# = \rightarrow$

$a_1 = a_2, b_1 - 1 = b_2, z_1 = z_2$ , and  $\# = \leftarrow$

$a_1 + 1 = a_2, b_1 = b_2, z_1 = z_2$ , and  $\# = \uparrow$

$a_1 - 1 = a_2, b_1 = b_2, z_1 = z_2$ , and  $\# = \downarrow$



$a_1=a_2, b_1=b_2, z_2(a_1,b_1)=\#, z_2(a,b)=z_1(a,b)$  for all other pairs  $(a,b)$ , and  $\# \notin \{\rightarrow, \downarrow, \leftarrow, \uparrow\}$

given a string  $w$  let's take  $z_w \in T$  be a function which  $z(i+1,1) = w(i)$  for  $0 < i \leq |w|$ ,  $z(0,b) = \triangleright$

for  $b \in \mathbb{N}$ ,  $z(a,0) = \Delta$  for all  $a > 0$ , and  $z(a,b) = \text{" "}$ , in different situation. If machine has two halting states  $y$  and  $n$  such that for any string  $w$

$(s,1,1,z_w) \vdash_M^* (y,i,j,z')$  or  $(s,1,1,z_w) \vdash_M^* (n,i,j,z')$  For such a machine deciding a language is the set of strings for which halts in the  $y$  state.

Standard Turing machines can simulate every move of a Turing machine with two dimensional tape. Hence they are at least as powerful as Turing machines with a two dimensional tape.

Since we have two dimensional tape it requires a quadratic time to find the coordinates that we do our operations and we can do our operations in constant time let's say. If we operate  $t$  times we will have  $O(t^3)$ . So we can say that can be simulated by a standard Turing machine in time that is polynomial in  $t$  and  $n$ .