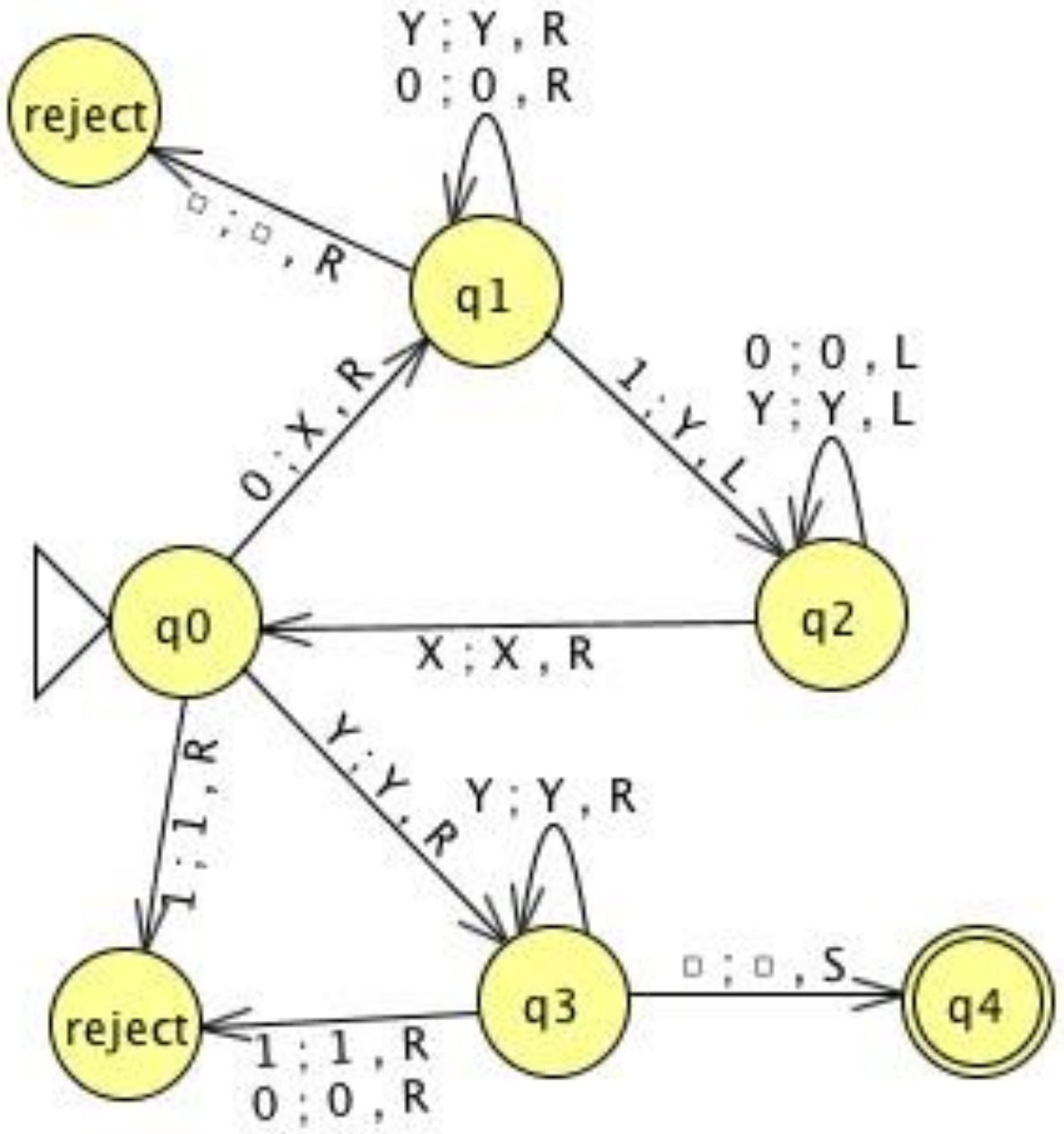


Theory of Computation and Programming Languages

Turing Machines

CS51 – Fall 2025

Turing Machines

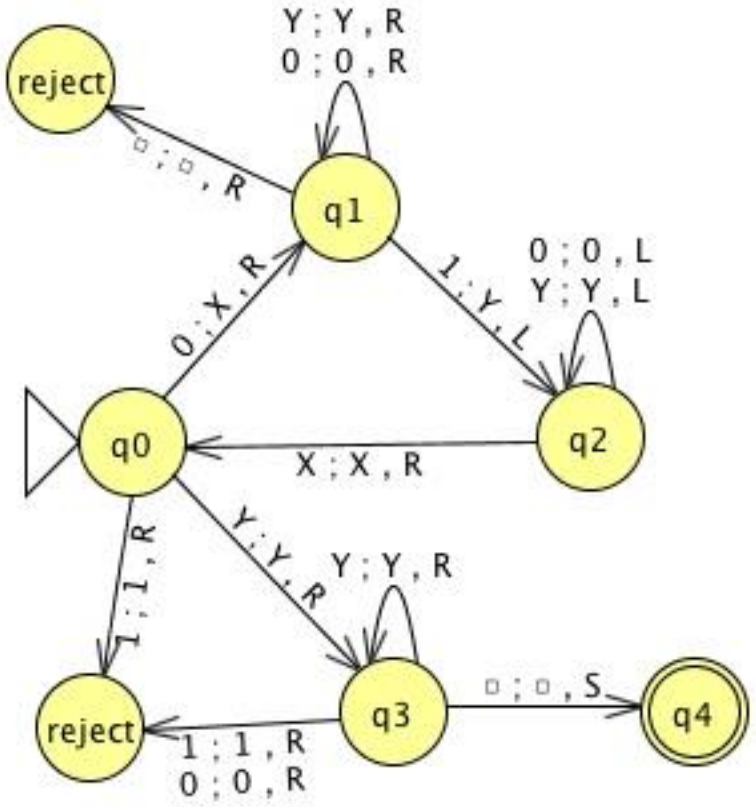
- Similar to DFAs/NFAs
  - they have states with transitions defined over the alphabet
  - they have a starting state and one or more final/accepting states
  - there is an input tape where each cell has a symbol

Turing Machines

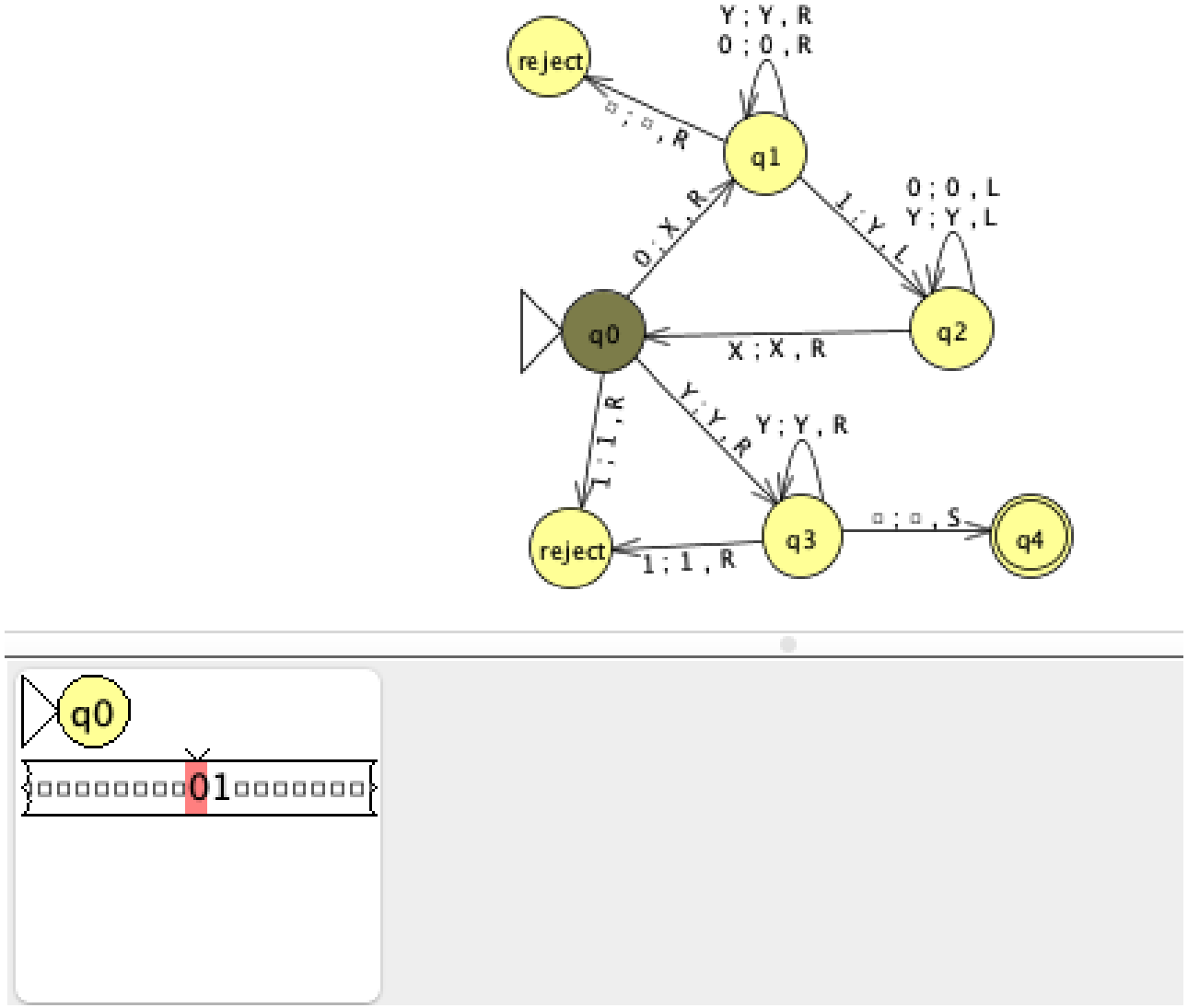
- Key differences:
  - They can **write** to the input tape as well as read
  - They have two alphabets:
    - **input alphabet**: just like DFAs and NFAs
    - **tape alphabet**: includes the input alphabet PLUS possibly additional characters
  - Can move head right *and* left on the input tape
  - The input tape is infinite!
    - it has "blank" characters everywhere beyond the string input (in both directions)
  - Has both **accept** *and* **reject** states
    - if it ever enters either it *immediately* accepts/rejects

Example

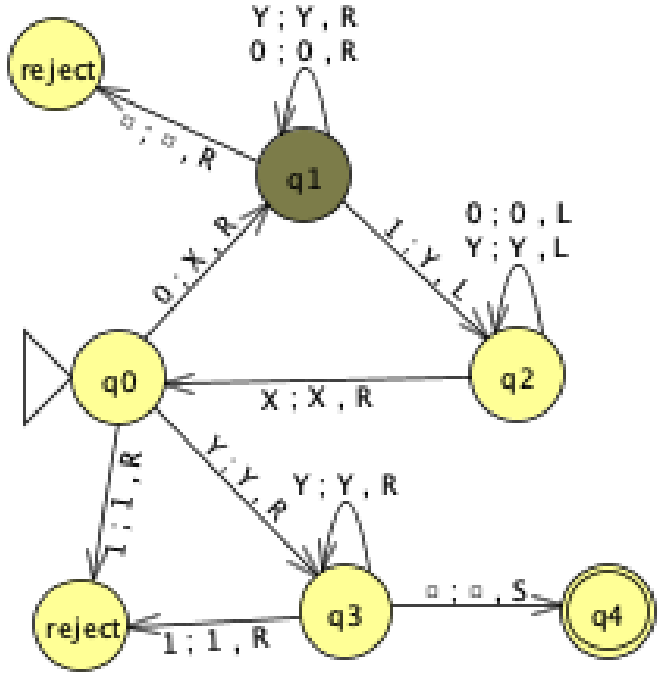
- Transitions are labeled with three things:
  1) the input character to read
  2) the output character to write
  3) which way to move the head on the tape (L, R or S)



Example 1: Input 01

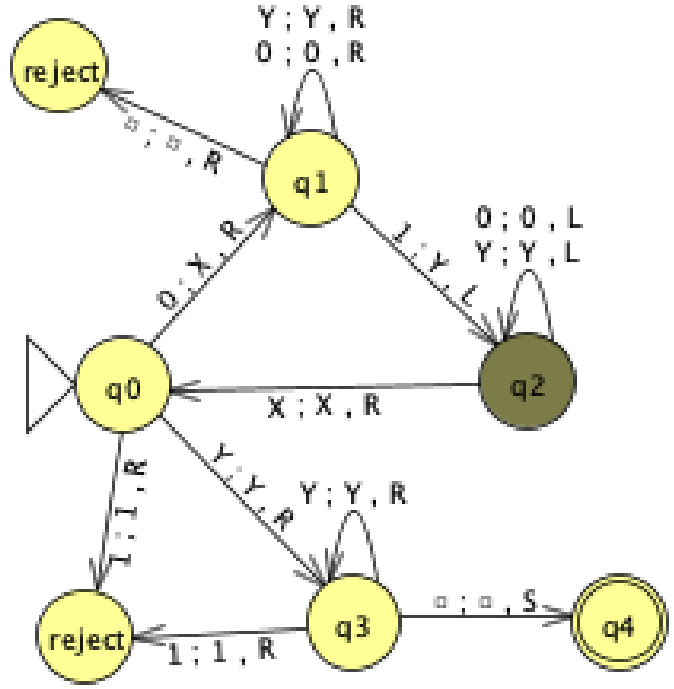


Example 1: Input 01



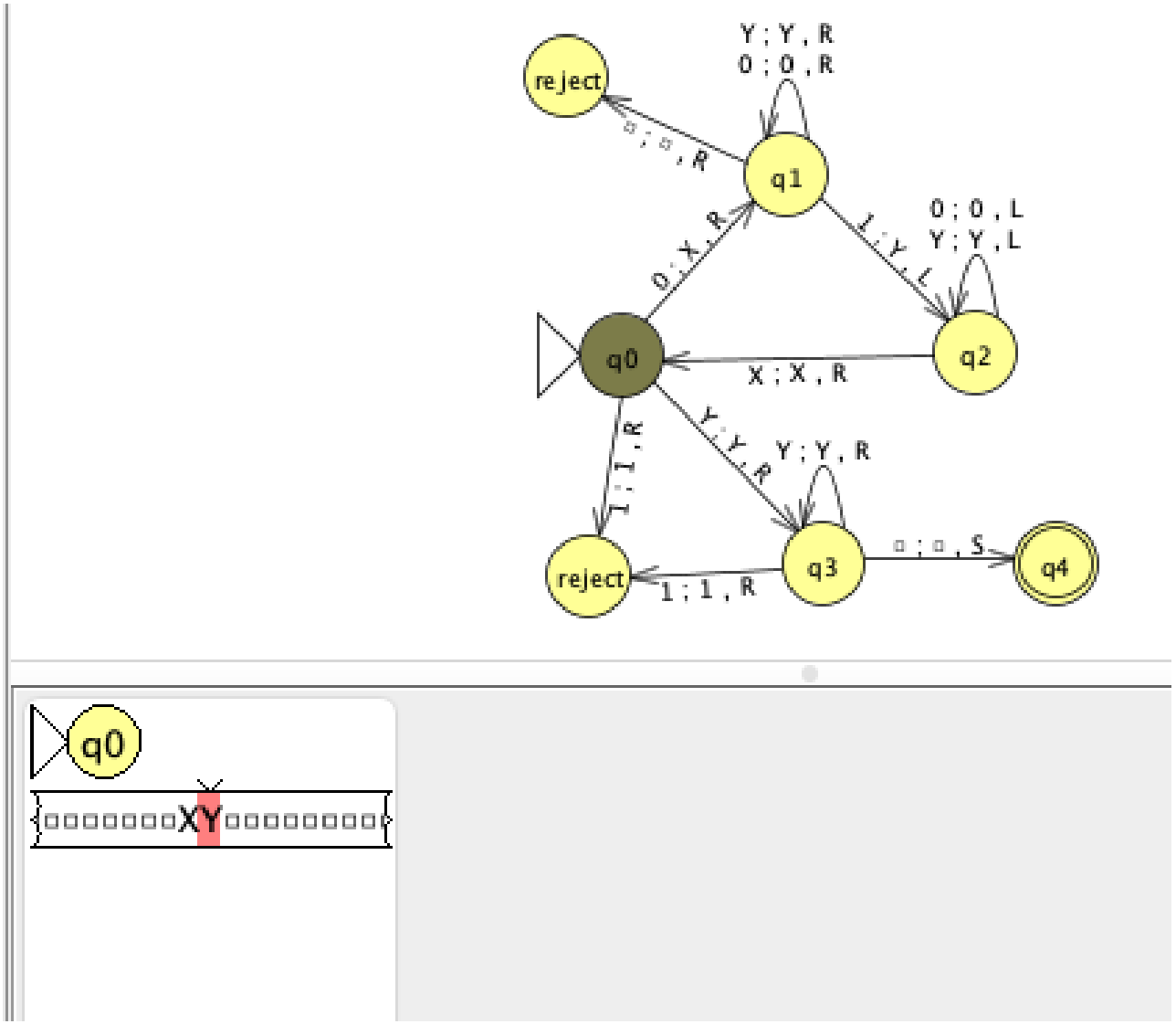


Example 1: Input 01

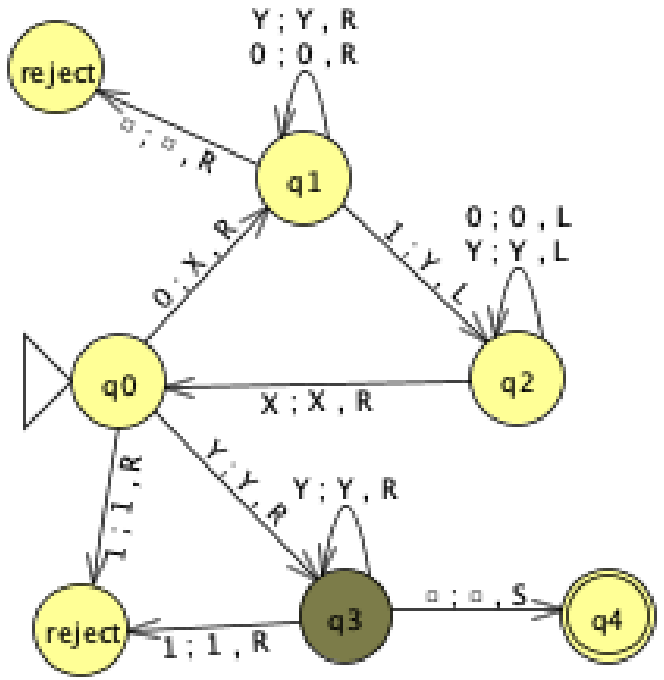




Example 1: Input 01

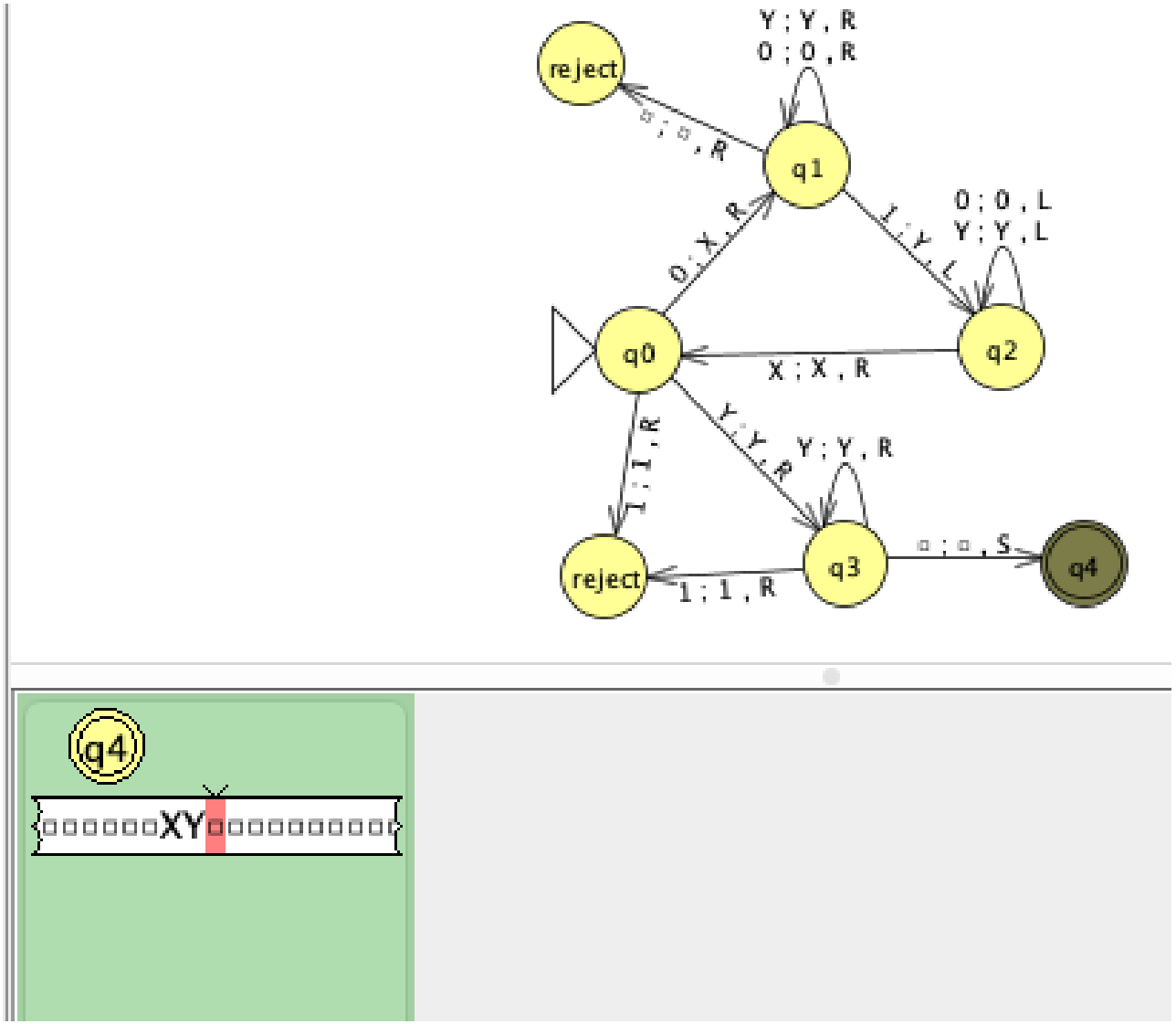


Example 1: Input 01

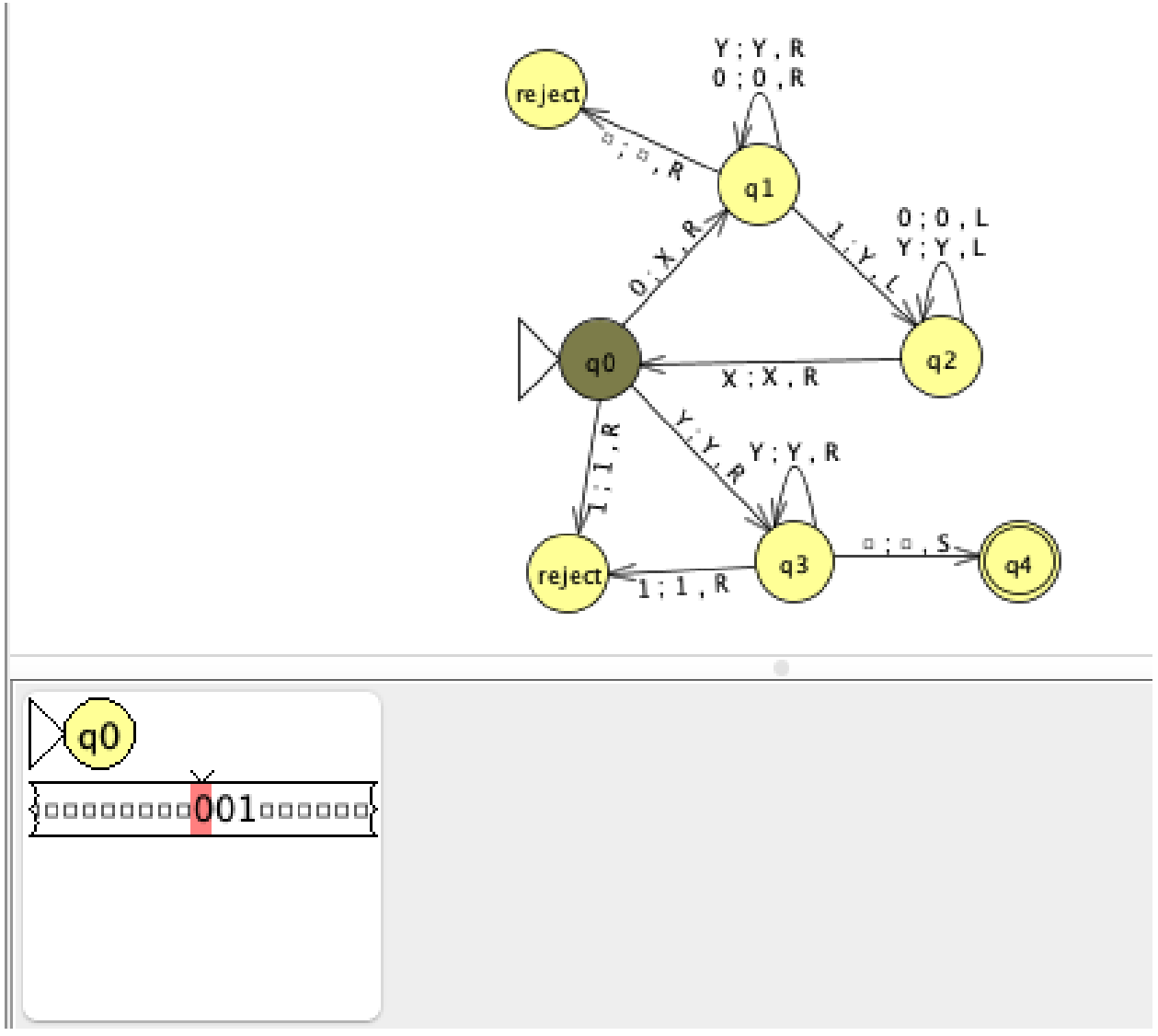




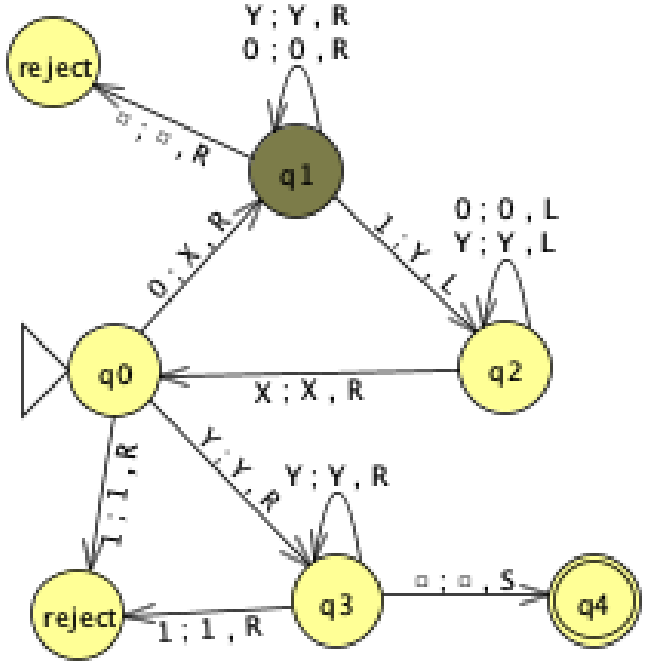
Example 1: Input 01



Example 2: Input 001

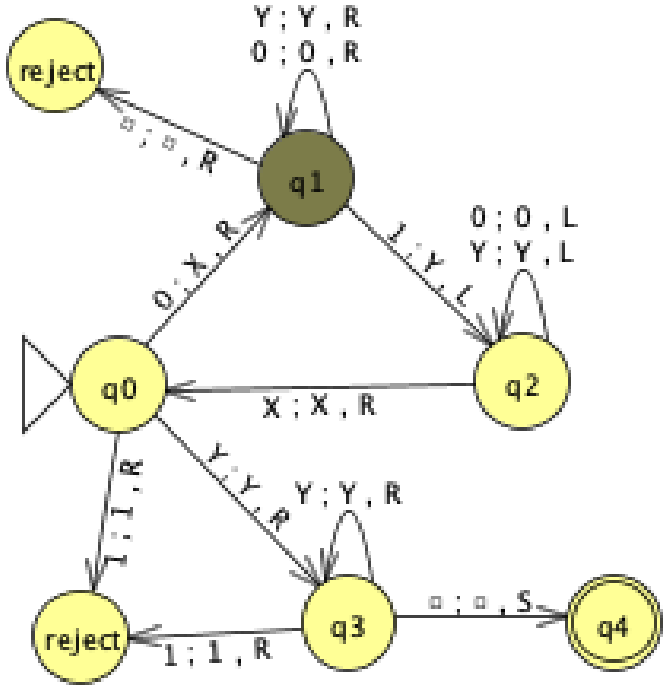


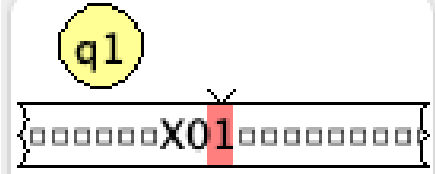
Example 2: Input 001



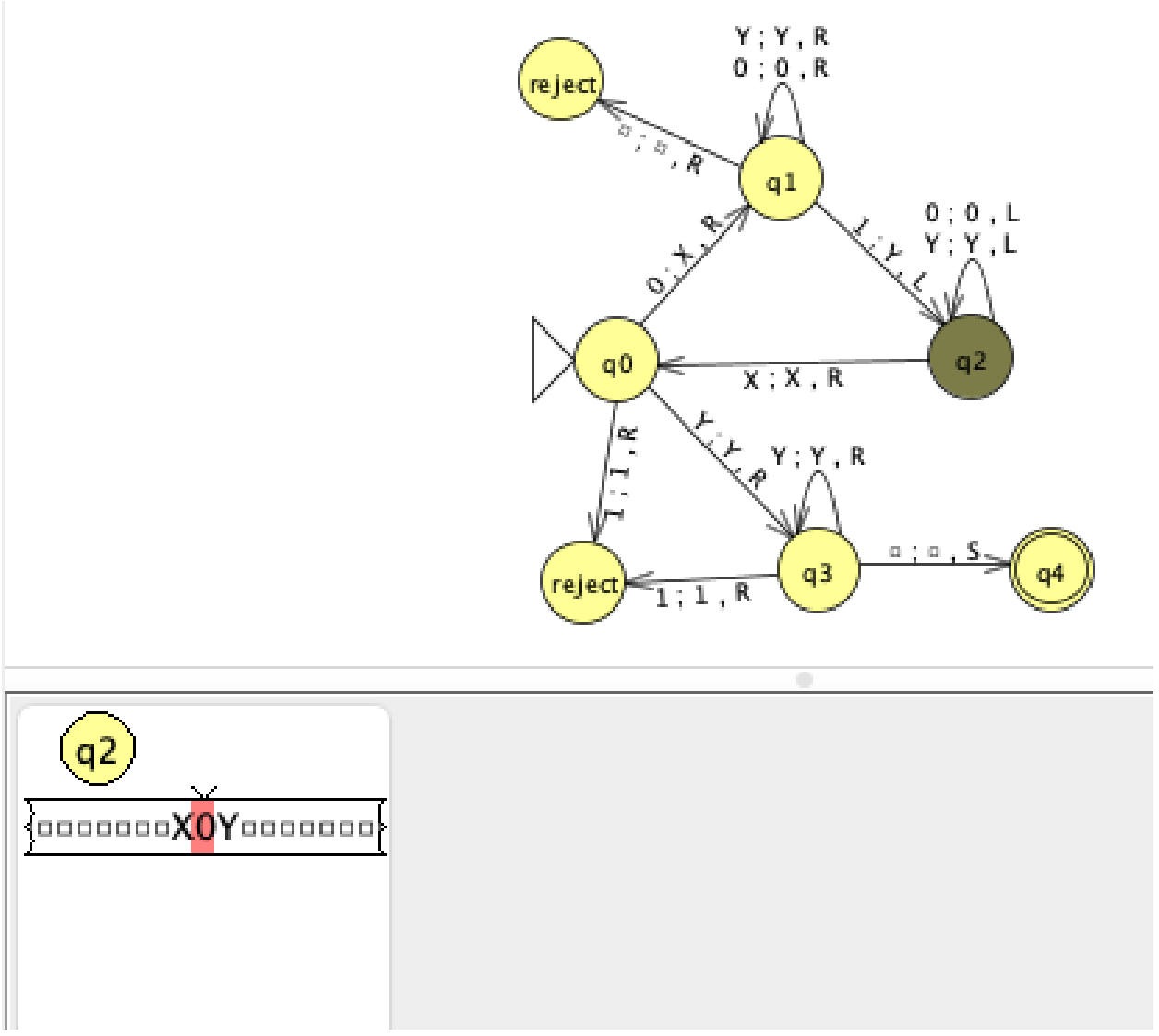


Example 2: Input 001

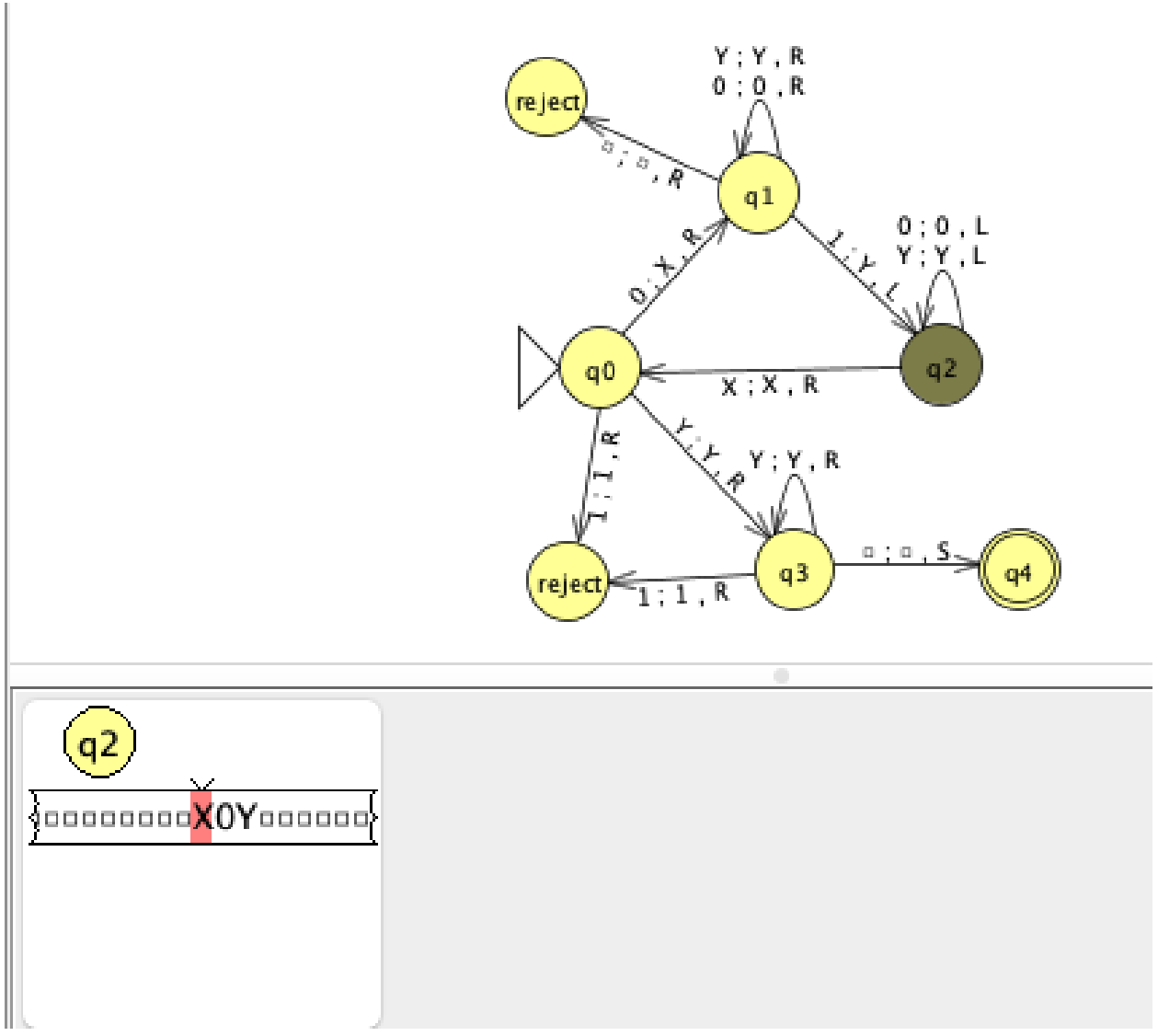




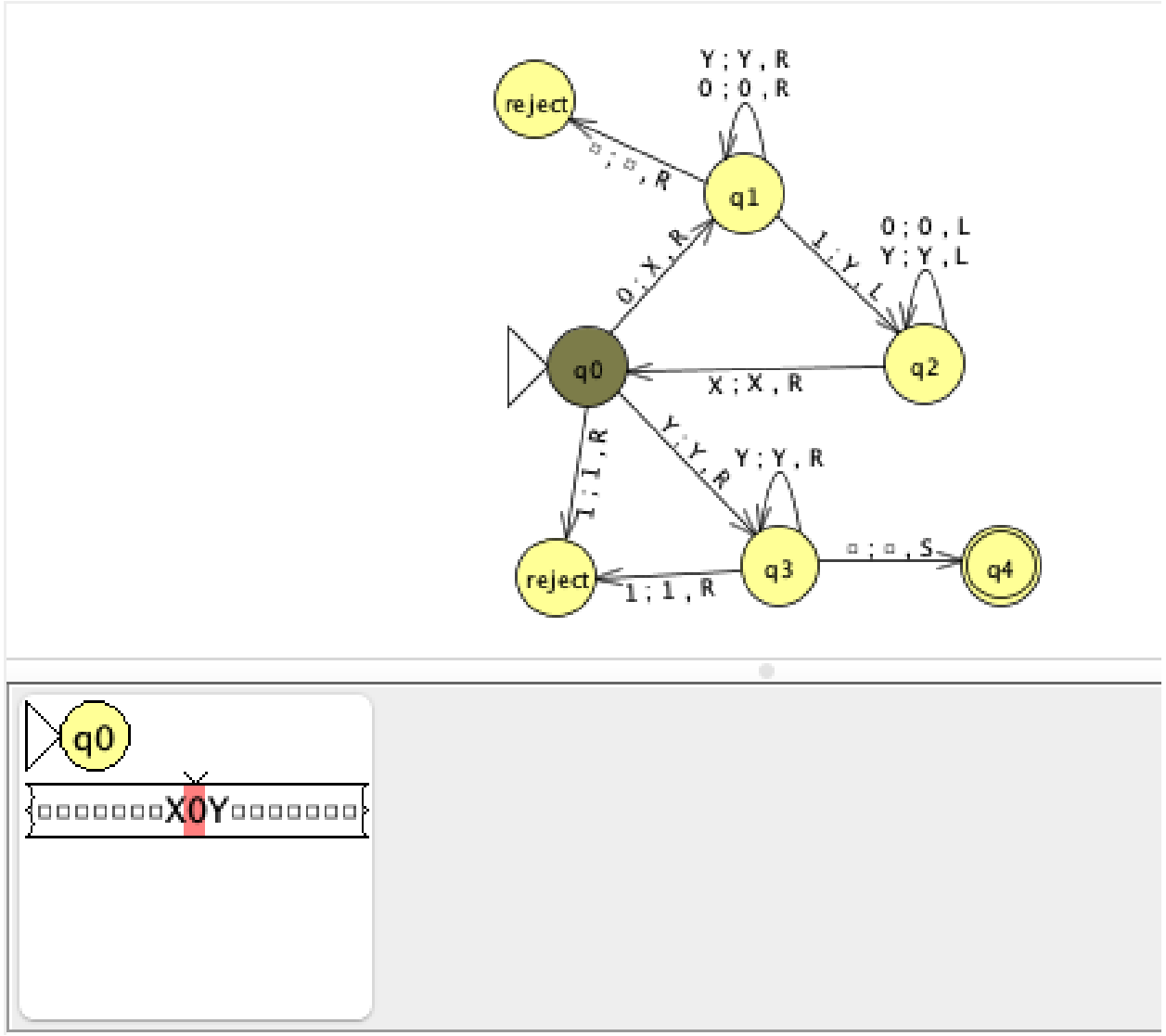
Example 2: Input 001



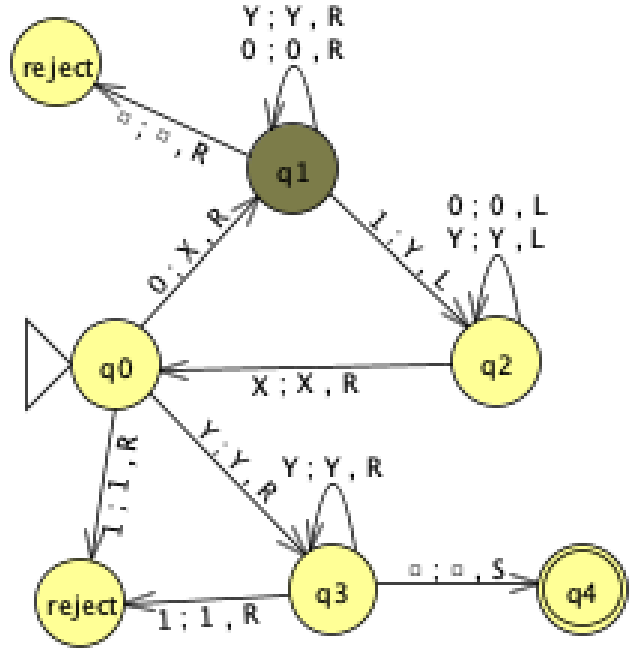
Example 2: Input 001



Example 2: Input 001

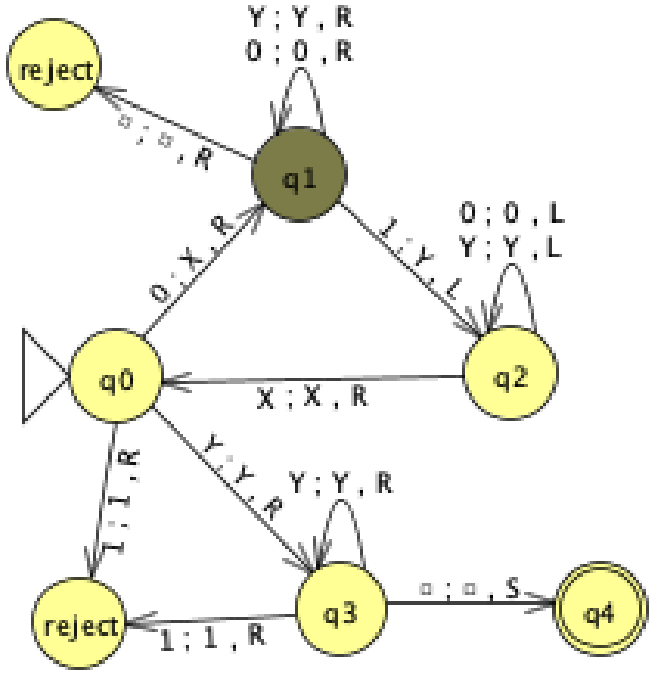


Example 2: Input 001



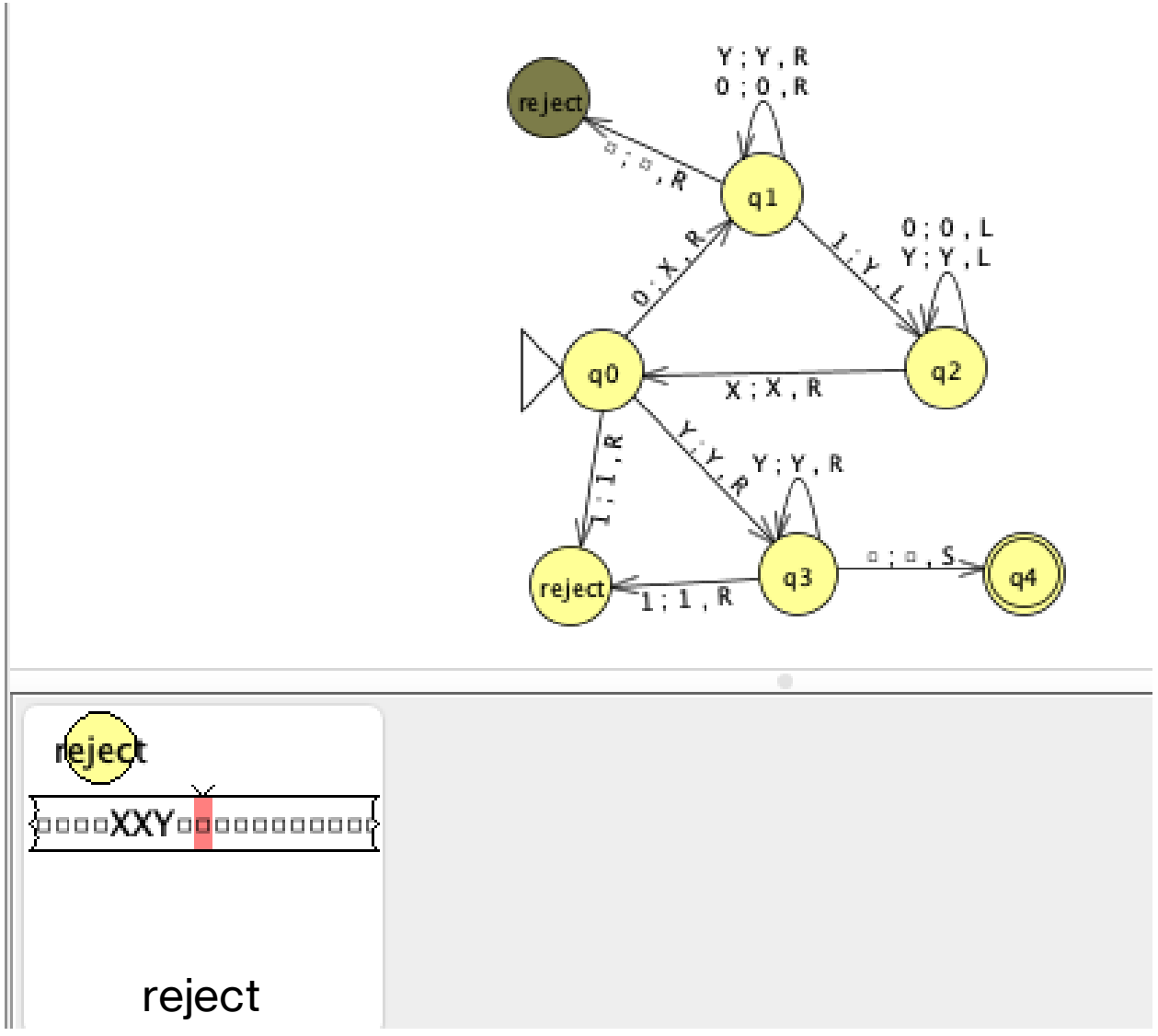


Example 2: Input 001



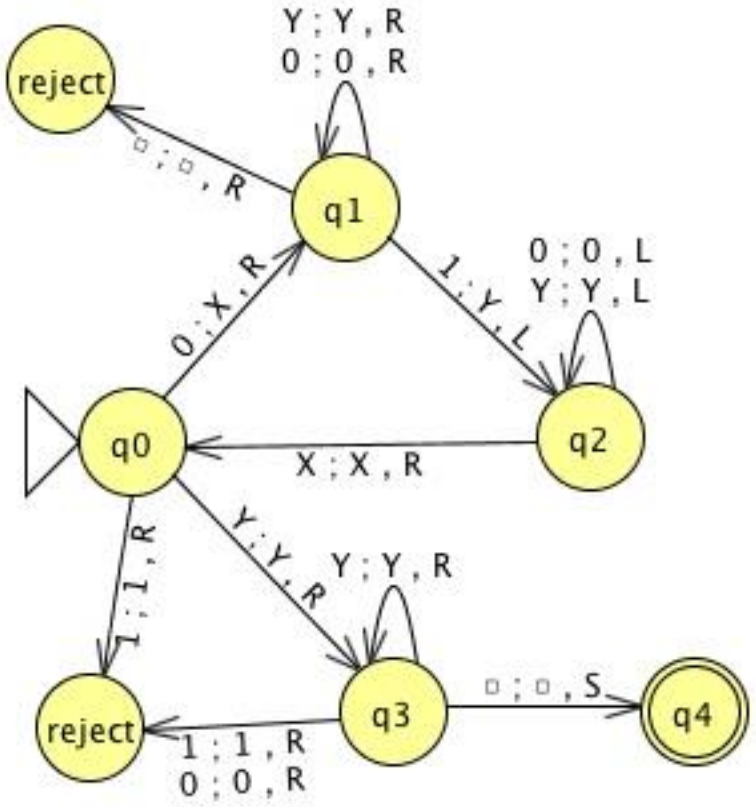


Example 2: Input 001



Basic idea for example 1

- Describes the language of strings with a number of 0s followed by the same number of 1s

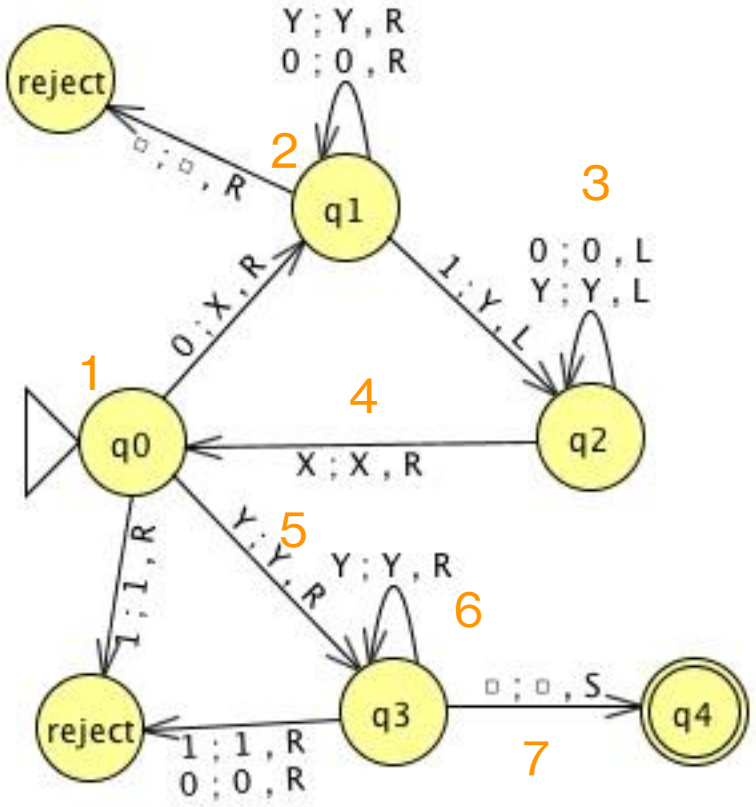


How does it do it?

0. read a 1: reject
1. read a 0: put an X down and move right
2. keep reading 0s and Ys to the right until you find a 1
3. put a Y on that 1 and move left
4. keep reading Ys and 0s going left until you find an X when you move right

- Repeat 1-4:

5. If you get to a point where the first character is a Y:
6. Read all the Ys and move right
7. Make sure that you get a blank (and not a 1 or a 0). If so, you accept!

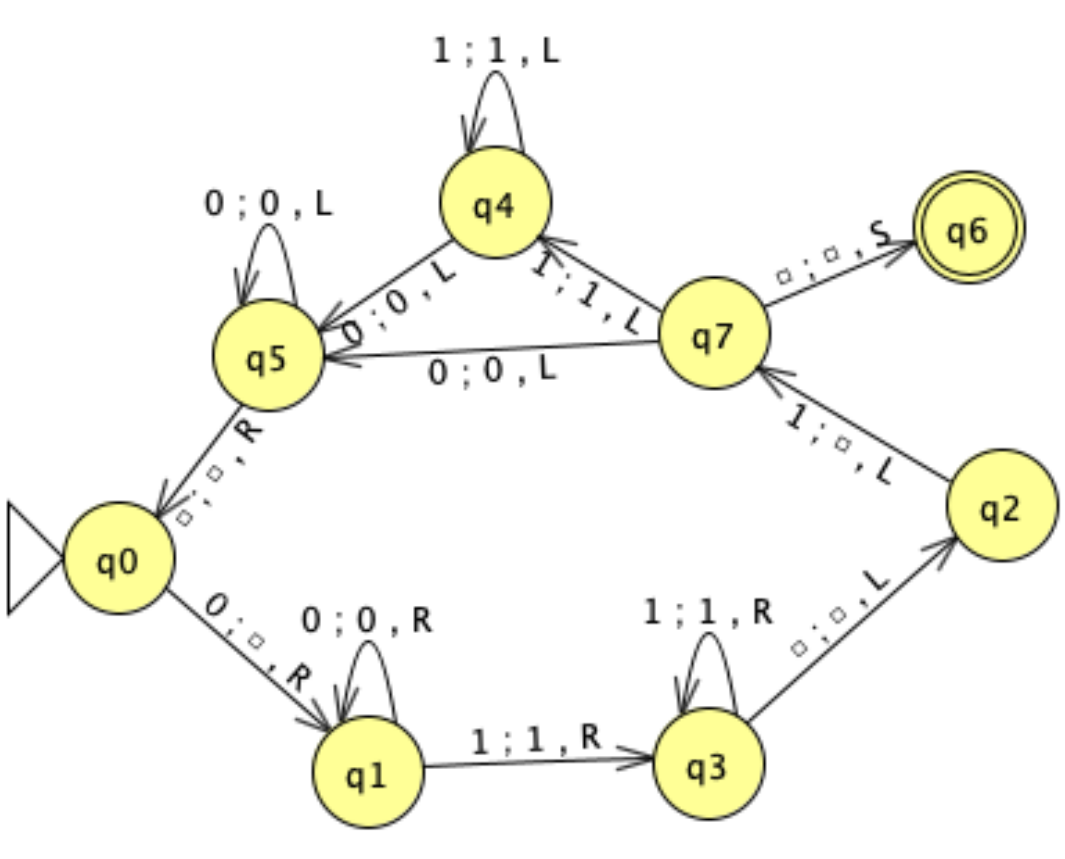


JFLAP and Turing Machines

- JFLAP doesn't have specific reject states for Turing machines.
- if it ever is in a state and there is no transition to take, it rejects.
- To make it clear (and to be consistent with other formulations) we will make one or more reject states without any outgoing transitions.
- if we want to reject, we will move to one of these states.

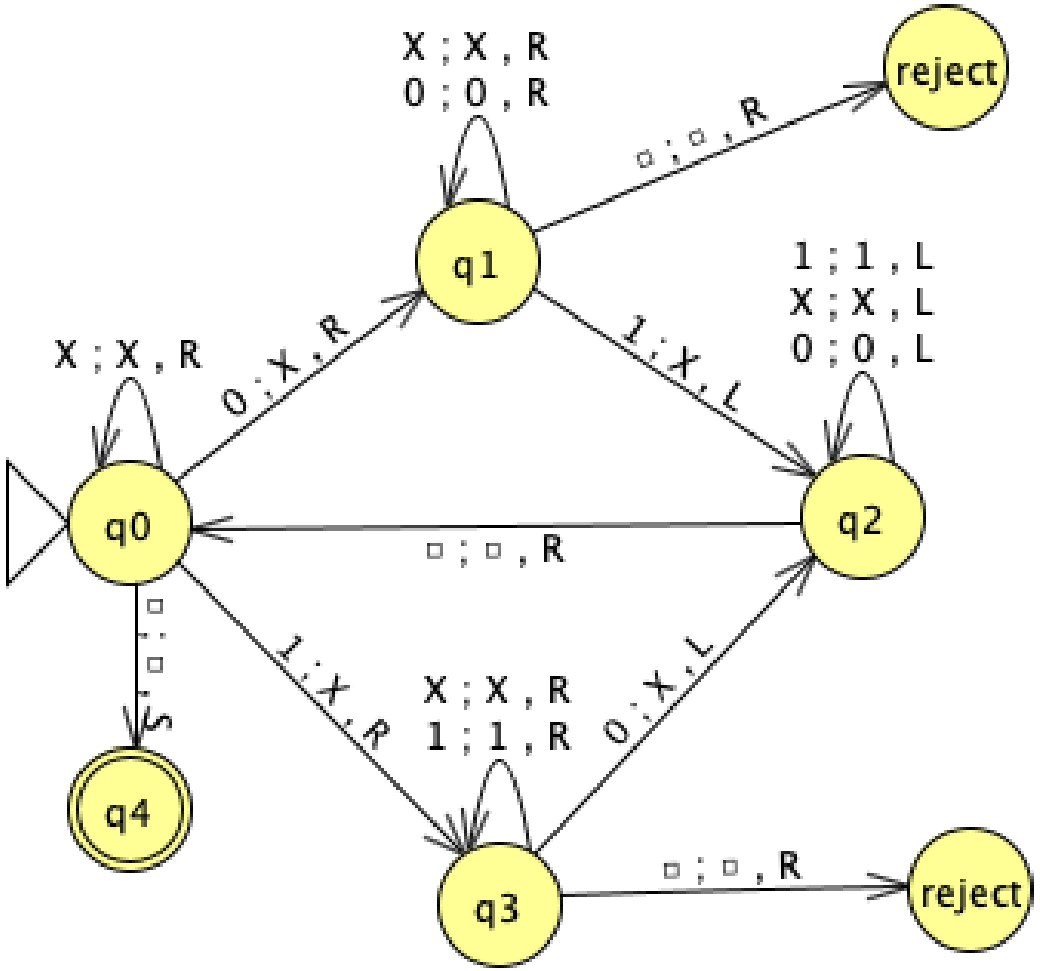
Example 2

- Describes the same language with example 1.
- Instead of using X and Y, we just cover up the end 0's and 1's with blanks
  - Cover up the first 0 with a blank
  - Search for the 1s, moving to the right when encountering a 0.
  - Go to the end of the 1s
  - Cover the last 1 with a blank
- Repeat



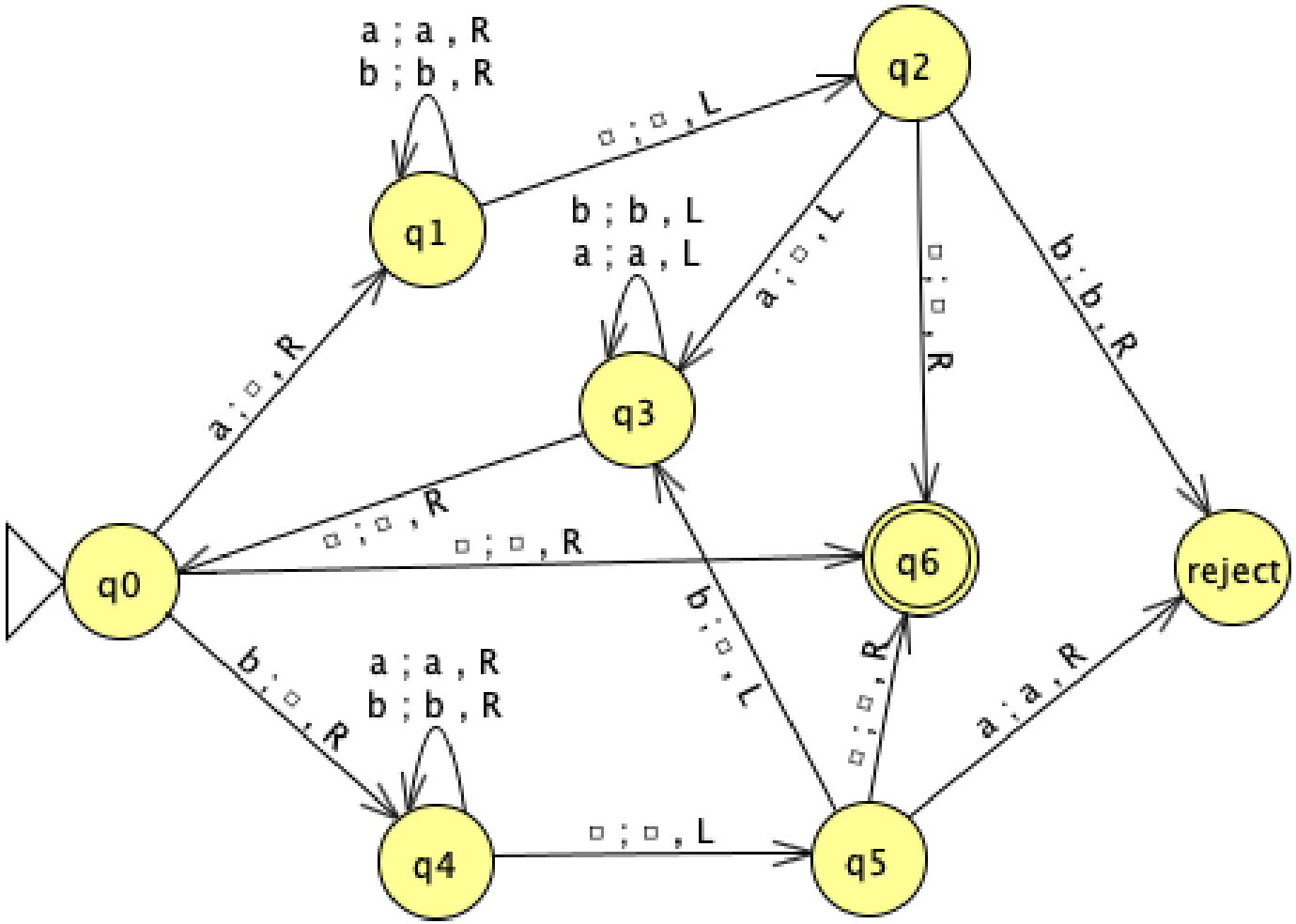
Example 3

- Describes the language with an equal number of 0s and 1s



Example 4

- Describes palindromes

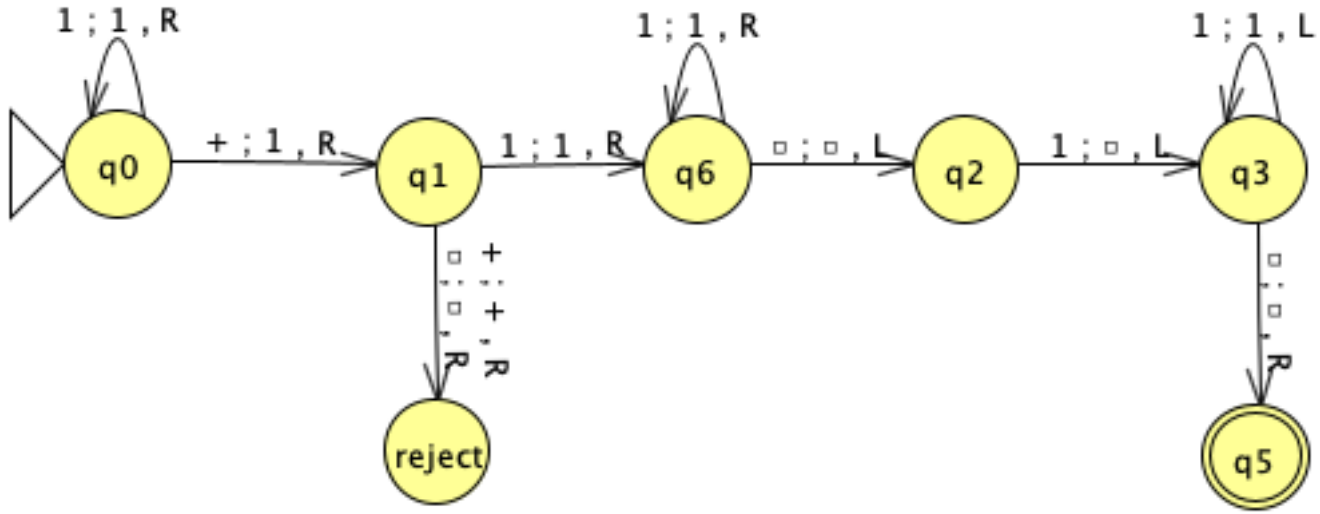


Producing output

- For this class, we'll just focus on accepting or rejecting strings.
- However, since Turing machines can write, they can also provide output.

Example 5

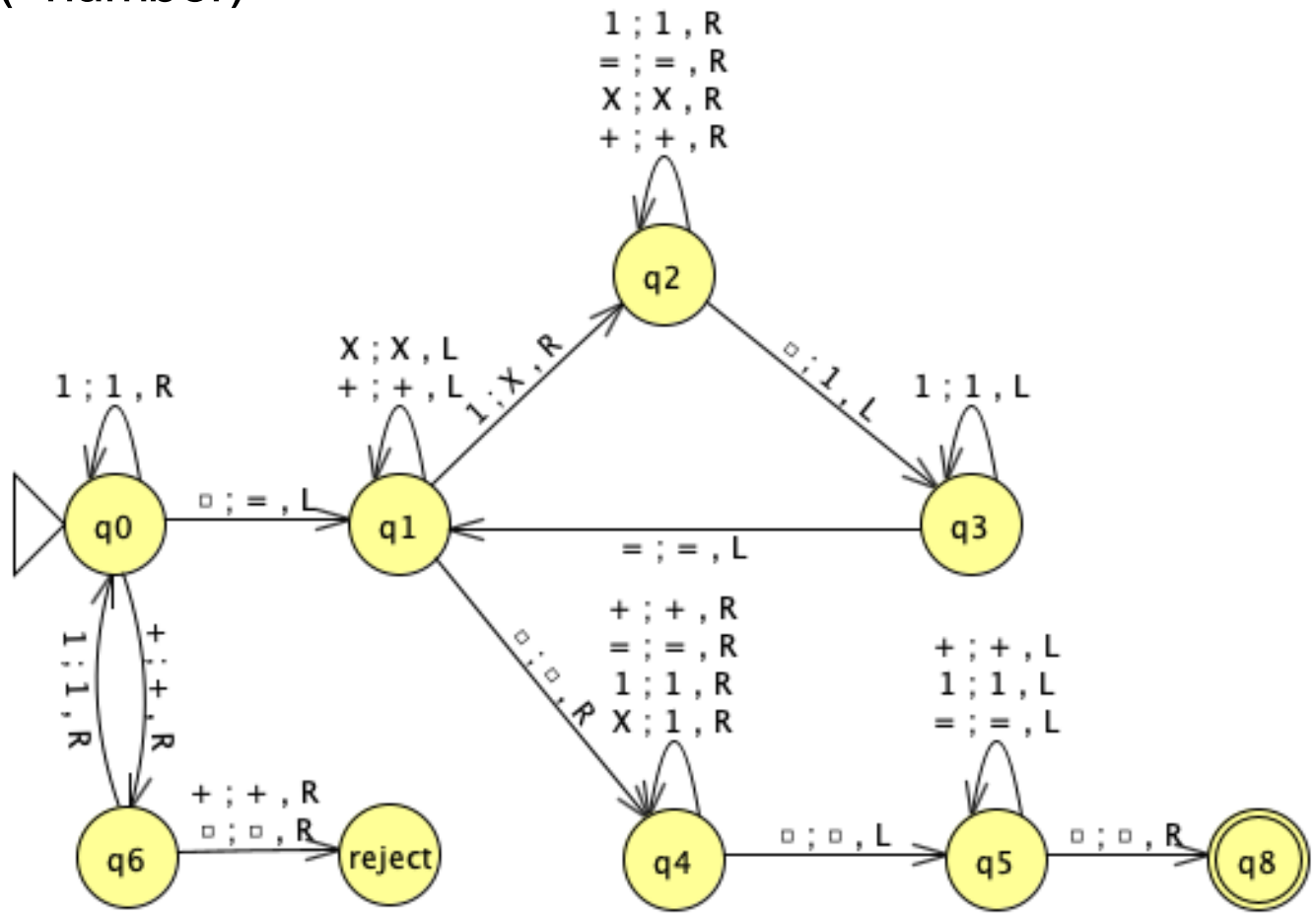
- Accepts strings of the form: number+number
- Numbers are represented in **unary**
- Unary numbers:
- 1 = 1
- 2 = 11
- 3 = 111
- 4 = 1111
- 5 = 11111



- In addition to accepting strings of this form, it also calculates the result!
- E.g. 11+111 should give us 11111 by putting it on the tape

Example 6

- Accepts strings of the form: number(+number)*
- Numbers are represented in unary



Church-Turing thesis

- Any computable process can be carried out on a Turing machine.
- It is widely acknowledged today that **all general-purpose computers can be reduced to the idea of a Turing Machine**.
- If a computer is as powerful as a Turing machine, it's **Turing complete**. Your laptop, phone, microwave, thermostat, are all Turing complete.



Alonzo Church



Alan Turing

Halting Problem

- Could we write a Turing machine that simulates the running of another Turing machine?
- Take as input two things:
  1. some representation of the Turing machine
  2. some input to run on the Turing machine input (i.e. the input Turing machine from 1)
- Would then "run" the Turing machine it was simulating on the input
- This is called a "**universal Turing machine**".
- Turing formulated the **Halting Problem** and showed that not everything is computable by a Turing machine (and thus by any computer...)
- It turns out that there is no algorithm to decide whether a program halts on a given input value.

Halting Problem

- To prove that we **cannot** write this Turing Machine we'll do a proof by contradiction.
- We assume that we can, i.e. that we could construct the Turing Machine H that can decide whether a program halts on a given input value.
- We will show that this results in a contradiction, meaning it couldn't be the case that such a Turing Machine exists.

Halting Problem

- Let's construct the following Turing machine D(M) = Run H(M, [M]) where [M] is the string representation of M
  - if it accepts (i.e. M(M) will finish), then loop infinitely
  - if it rejects (i.e. M(M) will loop forever), then stop
- What happens when we run D(D)? Run H(D, [D]):
  - If it accepts, that implies that D(D) finishes.
    - However, if H(D, [D]) accepts, then we know by the definition of D that D(D) loops infinitely
  - if it rejects, that implies that D(D) never finishes
    - however, if H(D, [D]) rejects, then we know that D(D) halts
- We've reached a contradiction!
  - If H says D finishes, then it loops infinitely
  - If H says D loops, then it halts

JFLAP examples:

- Turing Machine examples