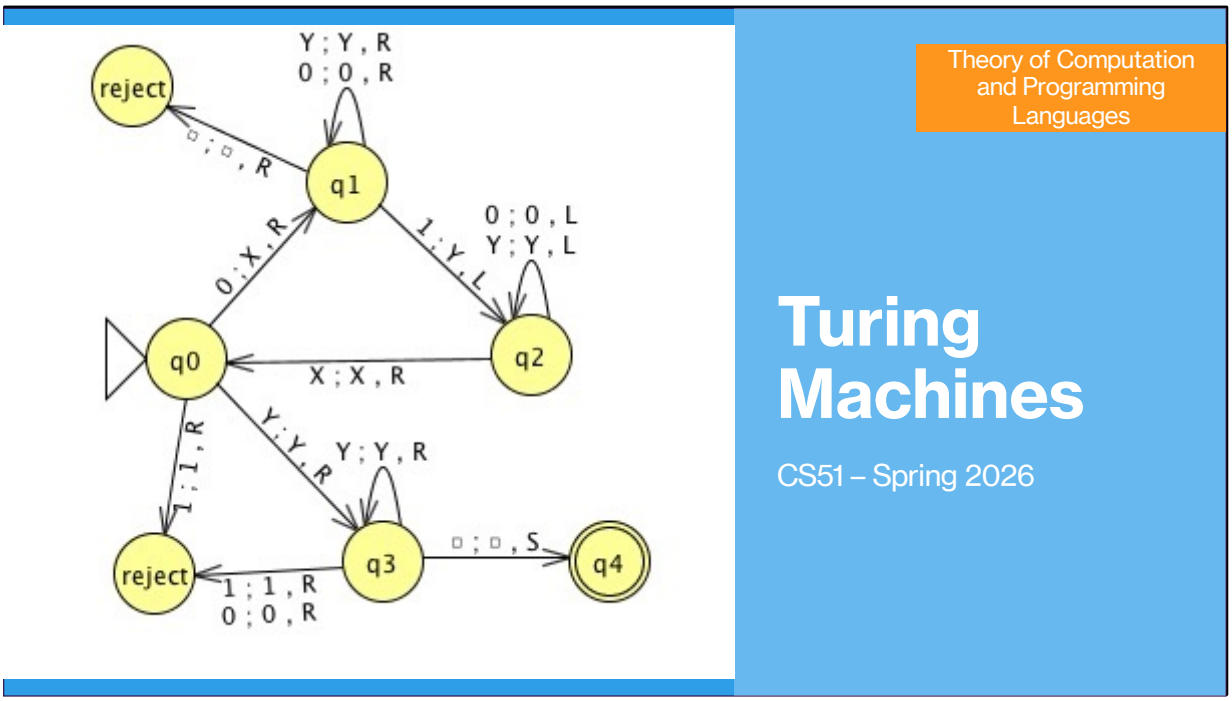


Welcome to the last lecture of this unit where we will talk about Turing machines.

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

2

Turing machines are similar to the DFAs and NFAs we have seen so far. They have states with transitions defined over an alphabet, they have a starting state and one or more final/accepting states, and there is an input tape where each cell has a symbol (that is a bit more broadly defined than just a character of our alphabet).

Turing Machines

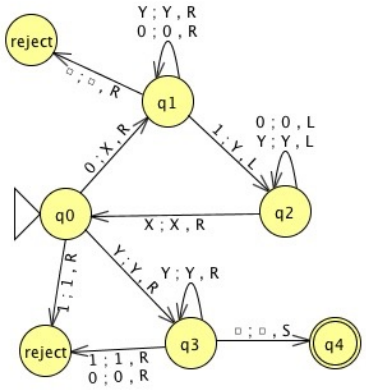
- Key differences:
  - They can **write** to the input tape as well as read
  - They have two alphabets:
    - **input alphabet**: just like DFAs and NFAs
    - **tape alphabet**: includes the input alphabet PLUS possibly additional characters
  - Can move head right *and* left on the input tape
  - 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

3

Nevertheless, Turing machines have some key differences. First, they cannot only read but also write to the input tape. They operate on the input alphabet just like DFAs and NFAs but the tape alphabet includes not only the input alphabet but possibly additional characters. Additionally, the tape head can move both right AND left on the input tape. And most amazingly the input tape is considered infinite and can extend both left and right beyond the string input. Finally, there are both accept and reject states which if the head enters it immediately accepts or rejects the string.

Example

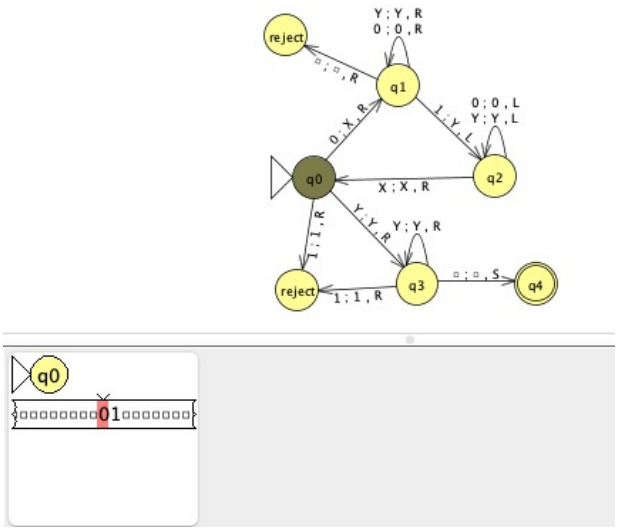
- Transitions are labeled with three things:
  1) the input character to read
  2) the output character to write
  3) which way to move the head on the tape (L, R or S)



4

Here's an example of what a Turing machine looks like. States are just like DFAs/NFAs. But transitions you see are labeled a bit more complicated. Each transition has a tuple of three elements: the input character to read from the tape, an output character to write in its place, and which way to move the head: left (L), right (R), or stay where it is, as is don't move the head (S). The empty character is denoted with that empty square. Let's see what this Turing machine does.

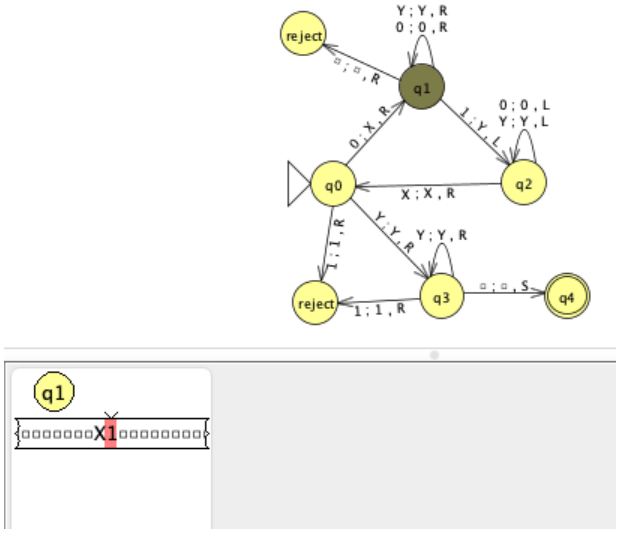
Example 1: Input 01



5

Say we are given the input string and start at our starting state q0. The transition says that when we read 0, we overwrite with an X and move one character to the right and transition to q1.

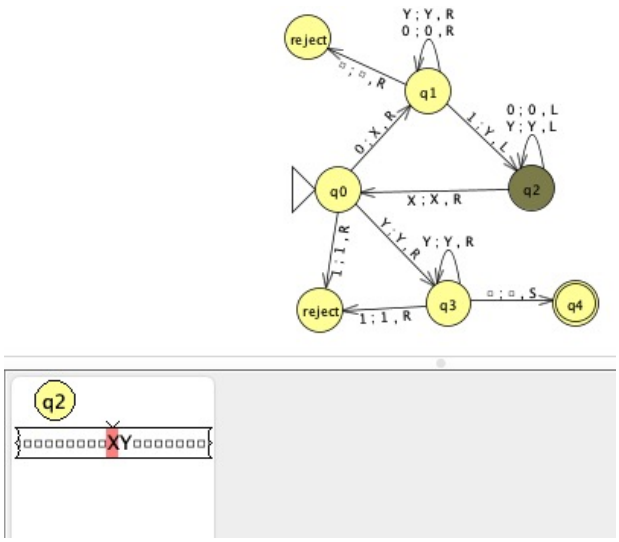
Example 1: Input 01



6

So our string is now X1 and the head is on the 1. We are also on q1 and about to read 1. When we read 1, we overwrite it with Y and move the head to the left. We also move to q2

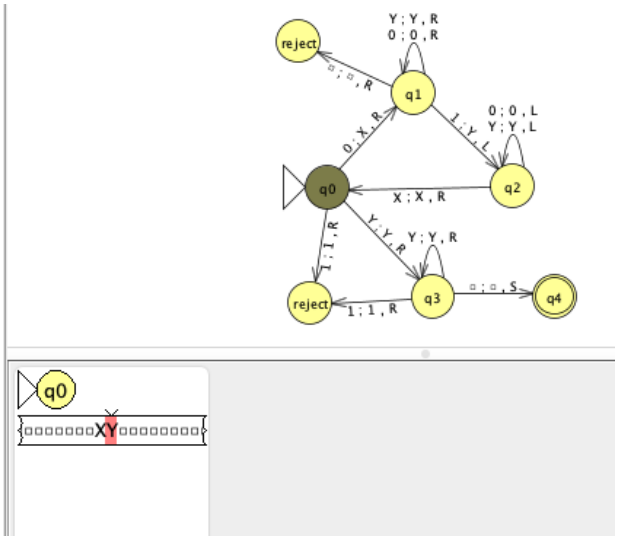
Example 1: Input 01



7

Which leaves the string as XY and moves the head to X. We are at q2 and read X. We see that we leave X as is and move the head to the right while we transition to state q0.

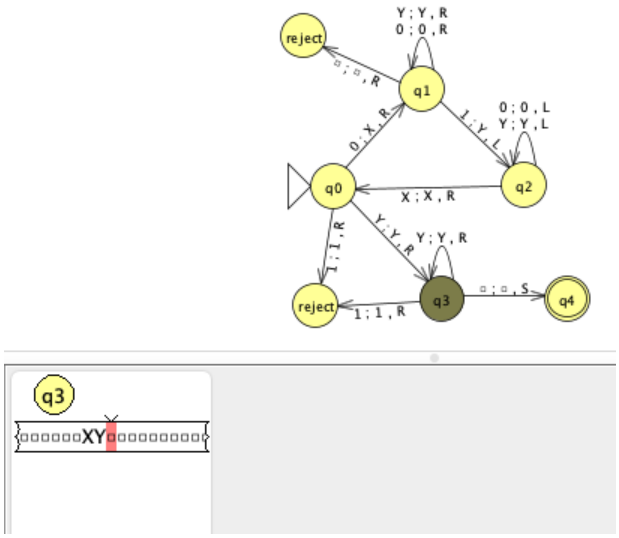
Example 1: Input 01



8

The string remains XY and the head is on Y. We are back at state q0 and since we read Y we leave it as is, move the head one character to the right and transition to state q3.

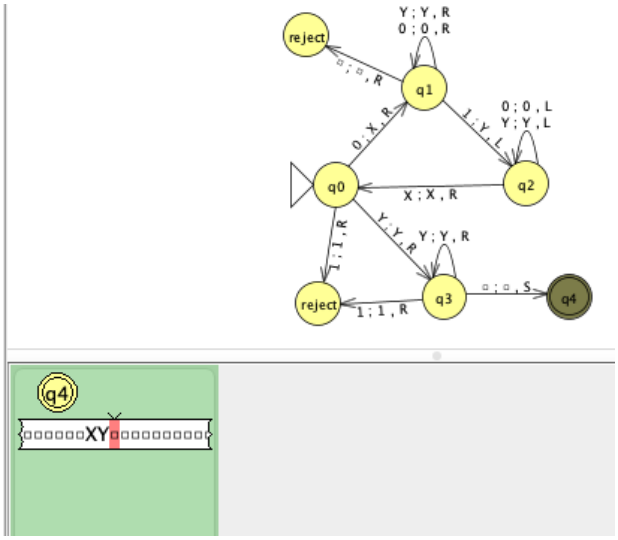
Example 1: Input 01



9

We are one character right of the string XY and at state q3. Since we read the empty character, we leave it as is, stay where we are on the tape and transition to q4.

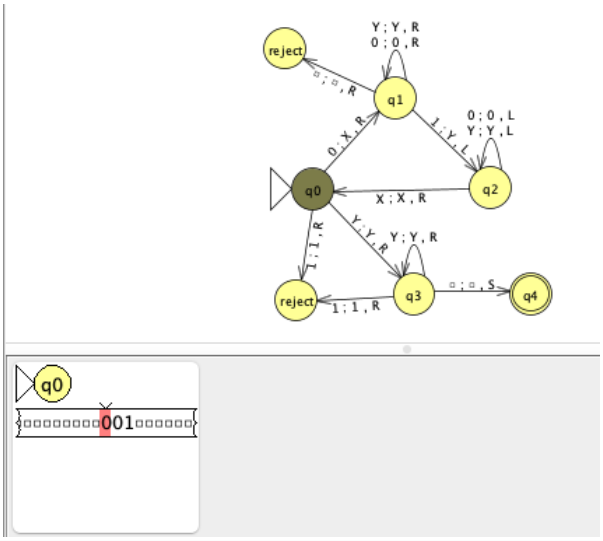
Example 1: Input 01



10

q4 is an accept state and we accept the string 01. Hmmm what does this Turing machine actually do?

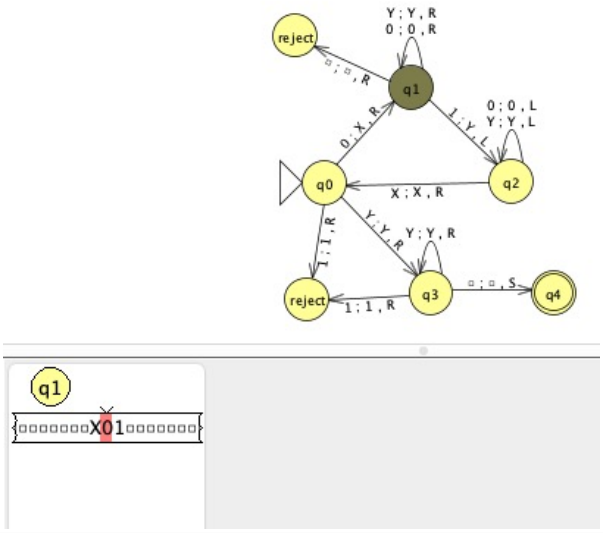
Example 1: Input 001



11

What if we started with the input 001? Again we are at q0 and transition to q1 by subbing 0 with X and moving the head to the right.

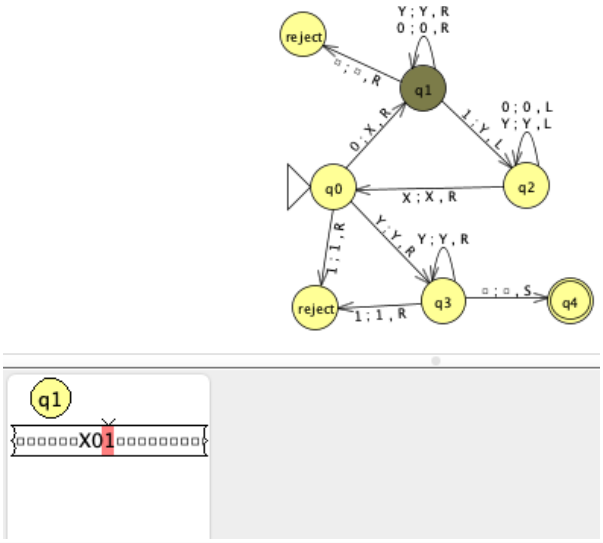
Example 1: Input 001



12

Our string is X01; the head is on 0 and we are state q1. We read 0 which means we stay in q1 without changing 0 and move the head to the right.

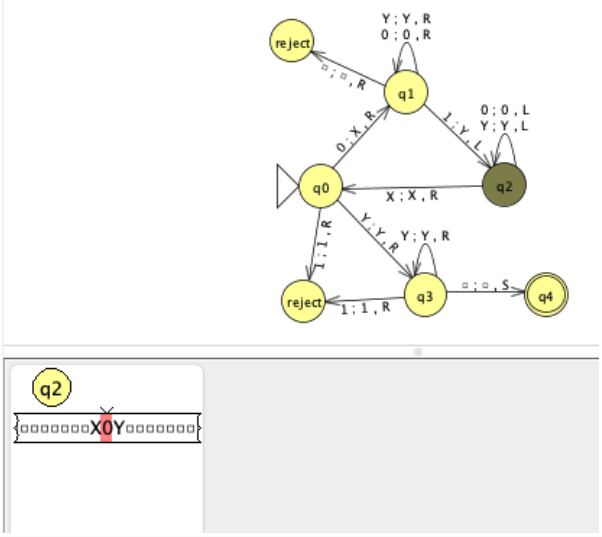
Example 1: Input 001



13

The head is at 1. When we read 1 at q1, we sub it with Y, move the head one character to the left and transition to q2.

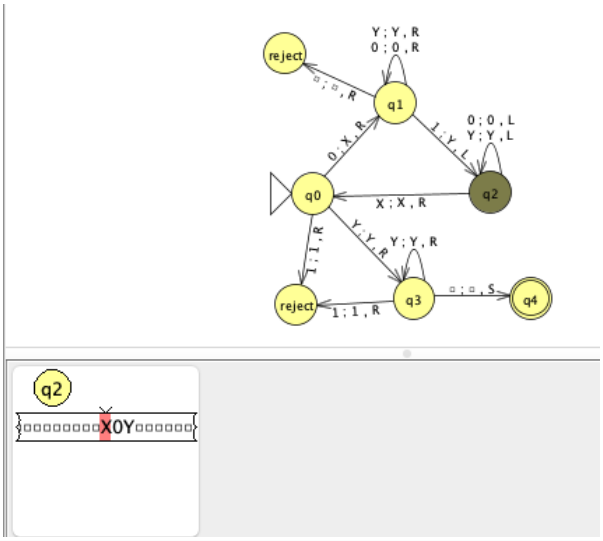
Example 1: Input 001



14

We are at q2 and the head is at 0 of the string X0Y. We stay in q2 leaving the string unaffected and moving the head to the left.

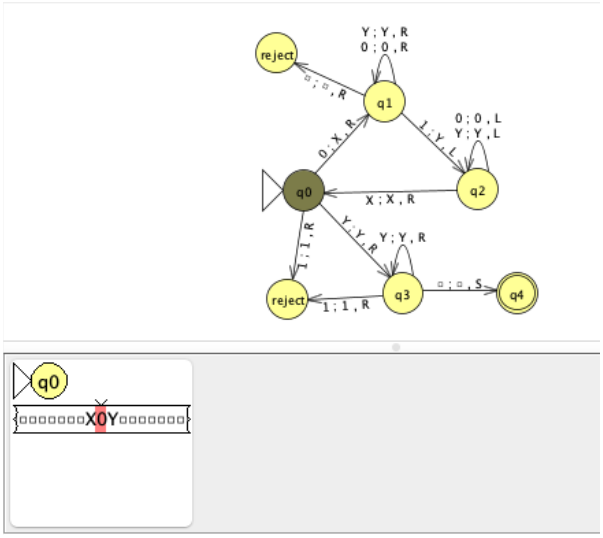
Example 1: Input 001



15

We are at state q2 and the head is at character X of the string X0Y. We transition to state q0, leaving X on the tape and moving the head to the right.

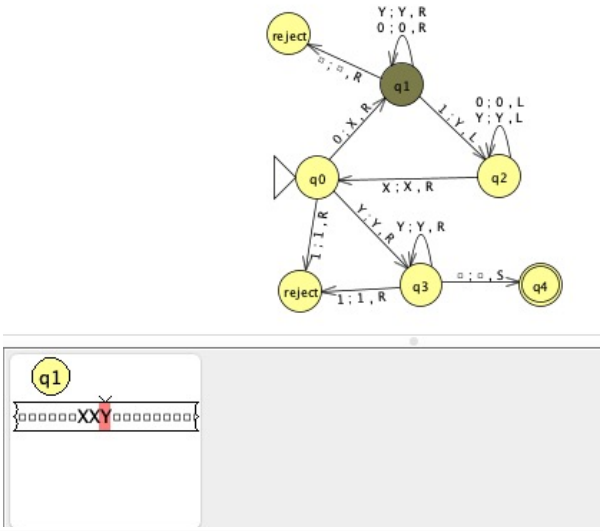
Example 1: Input 001



16

We are at q0, the head is on character 0 of string X0Y. We replace 0 with X and move the head to the right before transitioning to q1.

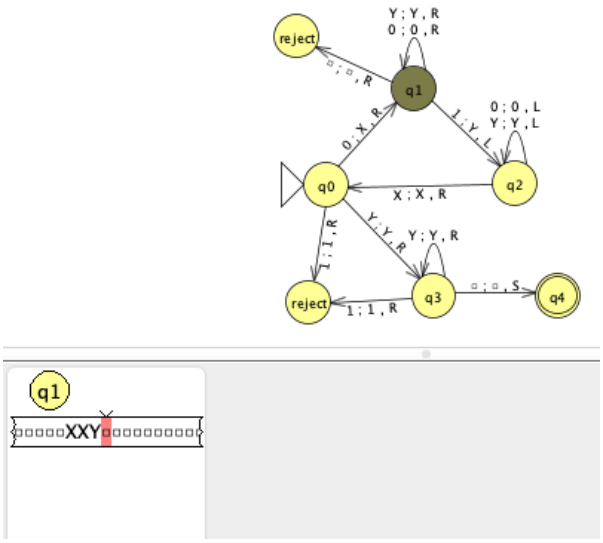
Example 1: Input 001



17

We are at state q1 and the head is at character Y of string XXY. We read Y, leave it as is, move head to the right and stay in q1.

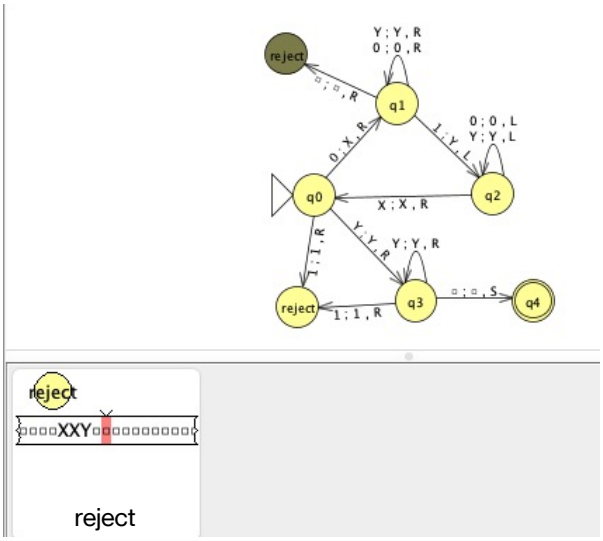
Example 1: Input 001



18

We are in q1, head on the empty character to the right of string XXY. We leave the empty character as is, move the head to the right and transition to the reject state.

Example 1: Input 001

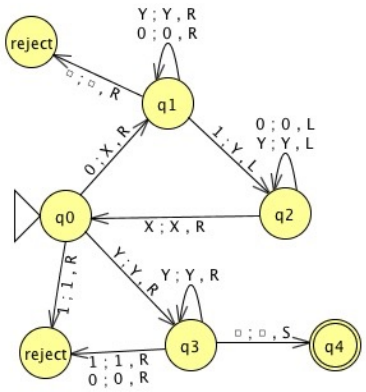


19

We immediately reject the string 001.

Basic idea for example 1

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



20

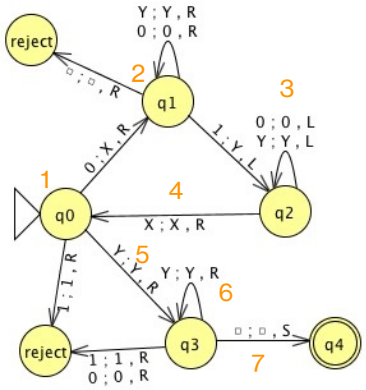
The basic idea behind this Turing machine is that it describes the language of strings that starts with a certain number of 0s and then those are followed by the same number of 1s

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!



21

How does it accomplish this? To confirm that it starts with 0s it immediately rejects strings if they start with a 1. Whenever it reads a 0, it puts in its place an X and moves the head to the right. If it encounters a 0 or Y it moves to the right until it finds a 1. It replaces the 1 with the Y and moves the head to the left. It keeps reading Ys and 0s going left until it finds an X and moves right. It keeps repeating this loop again and again. If it gets to a point where the first character is a Y it reads all Ys moving to the right. If it reaches a blank, then it accepts.

JFLAP and Turing Machines

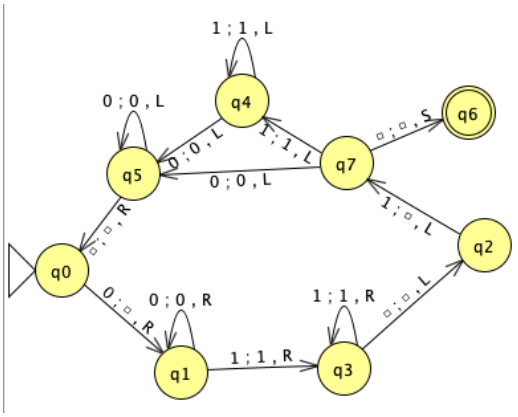
- JFLAP doesn't have specific reject states for Turing machines.
- if it ever is in a state and there is no transition to take, it rejects.
- To make it clear (and to be consistent with other formulations) we will make one or more reject states without any outgoing transitions.
- if we want to reject,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

22

Talking about how to simulate TMs in JFLAP, keep in mind that JFLAP does not have an explicit reject state. It rejects the string if it is in a state without transition.

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

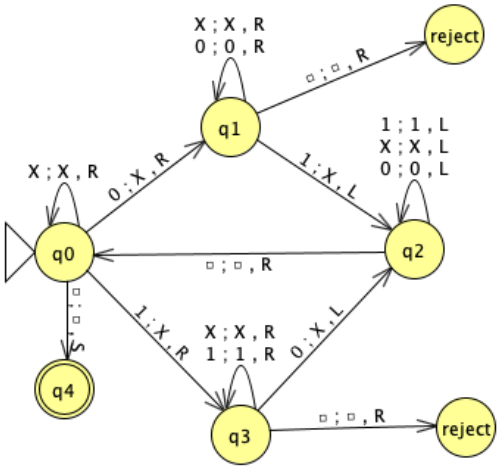


23

Here's another example of a Turing machine that describes the same language but instead of using Xs and Ys covers the end 0s and 1s with blanks.

Example 3

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

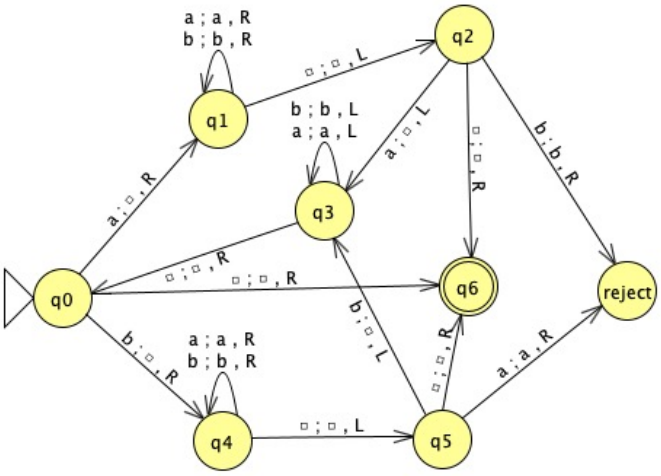


24

This Turing machine describes the language with an equal number of 0s and 1s

Example 4

- Describes palindromes



25

This one describes palindromes of a and bs (read right to left it's the same string)

Producing output

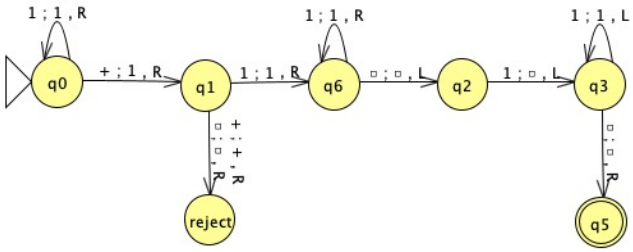
- For this class, we'll just focus on accepting or rejecting strings.
- However, since Turing machines can write, they can also provide output.

26

In CS51, we will just assume that TMs accept or reject strings but in practice they could 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



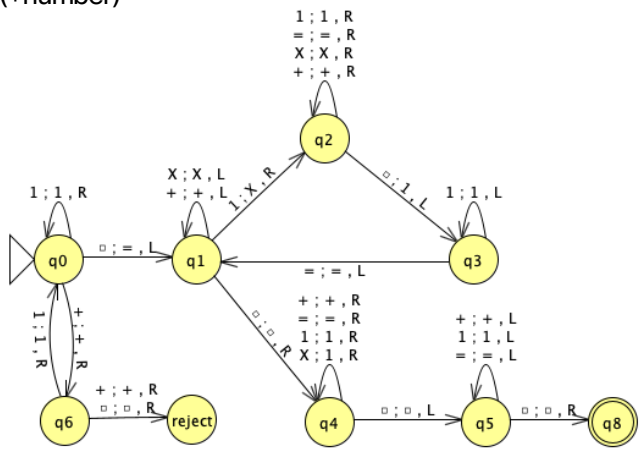
- In addition to accepting strings of this form, it also calculates the result!
- E.g. 11+111 should give us 11111 by putting it on the tape

27

For example, this TM accepts strings of the form number+number where numbers are represented in unary (all 1s). This not only accepts strings of this form but also calculates the result on the tape!

Example 6

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



28

This machine accepts strings of the form number(+number)*

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

29

Remember at the beginning of the semester we talked about the Church-Turing thesis which is named for the mathematicians Alonzo Church and Alan Turing, both pioneers in the theory of computation. The Church-Turing Thesis states that 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. isn't this amazing!?

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.

30

This brings us also to a very important problem in theoretical computer science, the Halting problem. Could we write a Turing machine that simulates the running of another Turing machine? It would take as input two things: some representation of th eTuring machine and some input to run on the TM. This would be a universal Turing machine. Alan 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.

31

How do we go about proving that we cannot write this Turing Machine? We would do a proof by contradiction (more in cs54), 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 proof by contradiction

- Assume there is a Turing Machine *H* that decides for any program *P* and input *x*, whether *P* halts on *x*:
  H(P, x) = YES if *P* halts on input *x*, and
  H(P, x) = NO if *P* does not halt on input *x*, that is it loops forever.
- We will construct a new machine *D* that takes *P* as a program and feeds it to *H* both as a program and input and then does the opposite of what *H* predicts, that is,
  D(P) = does not halt (loops forever), if *H(P, P)* = YES
  D(P) = halt, if *H(P,P)* = NO.
- What does *D* do when it runs with itself as input, that is *D(D)*?
  - **Case 1:** Suppose *D(D)* halts:
    - Then *H(D, D)* must have returned NO (since *D* halts only when *H* says NO).
    - But *H(D, D)* = NO means *H* predicted that *D* loops forever on input *D* . This is a contradiction!
  - **Case 2:** Suppose *D(D)* does not halt (loops forever).
    - Then *H(D, D)* must have returned YES (since *D* loops forever only when *H* says YES).
    - But *H(D, D)* = YES means *H* predicted that *D* halts on input *D.* This is a contradiction!

32

This would be the proof by contradiction for the halting problem. Since both cases lead to a contradiction, there can't exist such a Turing Machine H and the Halting Problem is undecidable.

JFLAP examples:

- Turing Machine examples

33

These are the examples we saw today!