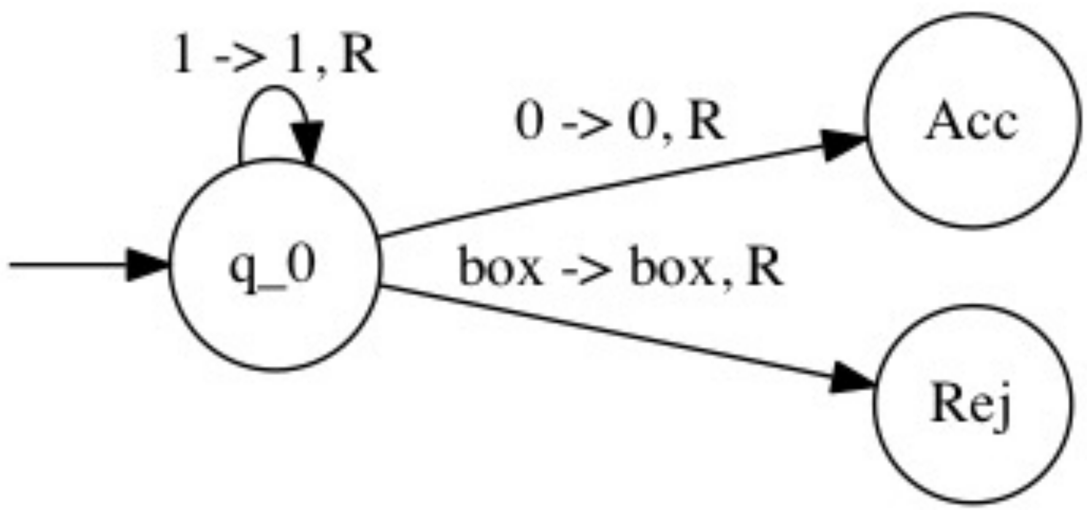
Turing Machines
Part 2

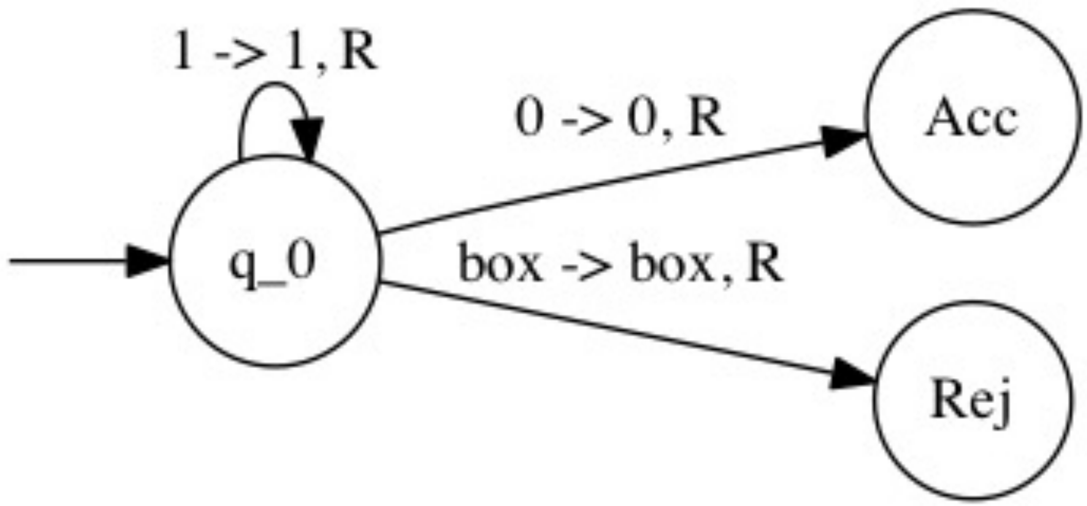
Adam Wyner
CS3518, Spring 2017
University of Aberdeen

First TM example

- Given: w is a bitstring
- TM that recognises L = {w: w contains at least one 0}



First TM example



Formally

$\delta(q_0, 1)=(q_0, 1, R)$

$\delta(q_0, 0)=(Acc, 0, R)$

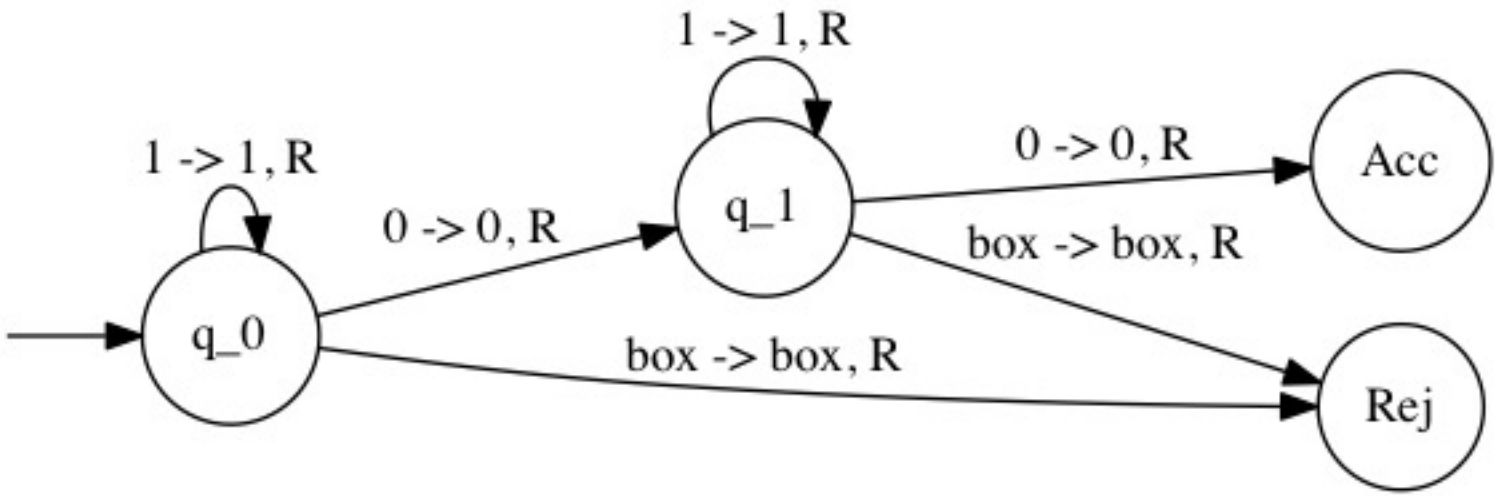
$\delta(q_0, \square)=(Rej, \square, R)$

Observe ..

- The "transition function" in this graph is not a **total** function. However,
  - The missing arrows (from the Acc and Rej states) can easily be added
  - How you do this does not affect the language recognised by the TM

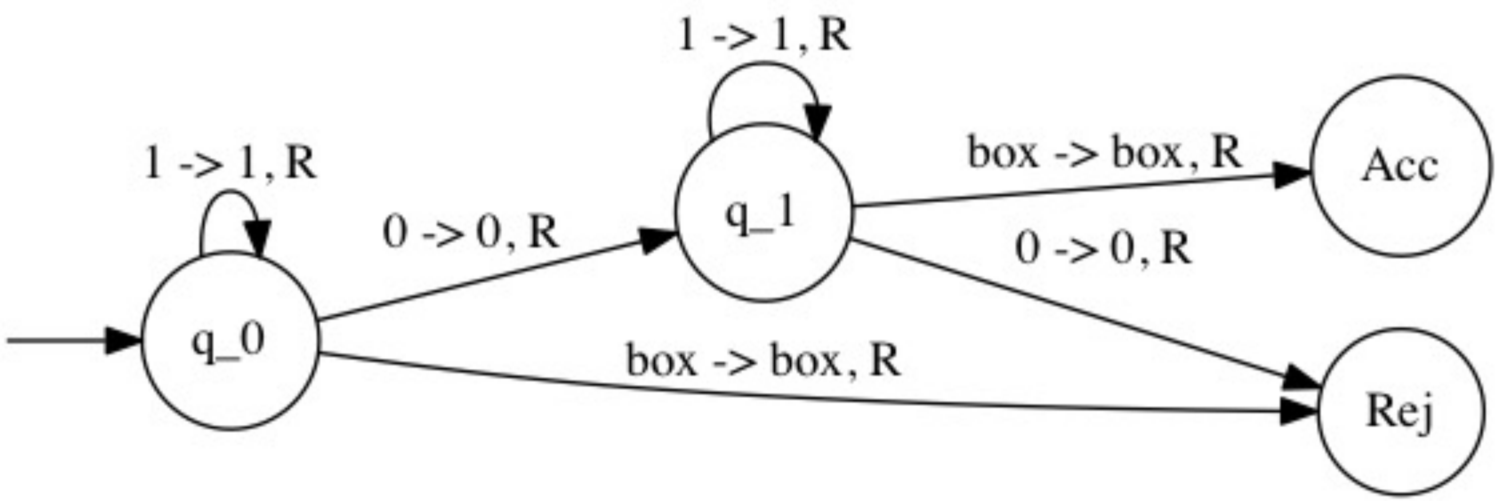
Second TM example

- Given: w is a bitstring
- TM that recognises L = {w: w contains at least two 0s}



Third example

- Given: w is a bitstring
- TM that recognises L={w: w contains exactly one 0}



High-level description

- These TMs were very simple and only solve problems that FSAs were able to solve already.
- More sophisticated TMs get complicated
  - A lengthy document
  - We often settle for a higher-level description
  - Easier to understand than transition rules or diagrams
- Every higher-level description is just a shorthand for its formal specification

0s power of 2

- M_2 recognises all strings of 0s whose length is a power of 2, that is, the language

$$A = \{ \mathbf{0}^{2^n} \mid n \geq 0\}$$

= {0,00,0000,00000000,0000000000000000, ...}

0s power of 2

= {0,00,0000,00000000,0000000000000000, ...}

M_2 = On input string w:

1. Sweep left to right across the tape, crossing off every other **0**.
    a. If tape contains a single **0**, accept.
    b. If tape contains more than a single **0** and the number of **0**s was odd, reject.
2. Return the head to the left-hand of the tape
3. Go to step 1.

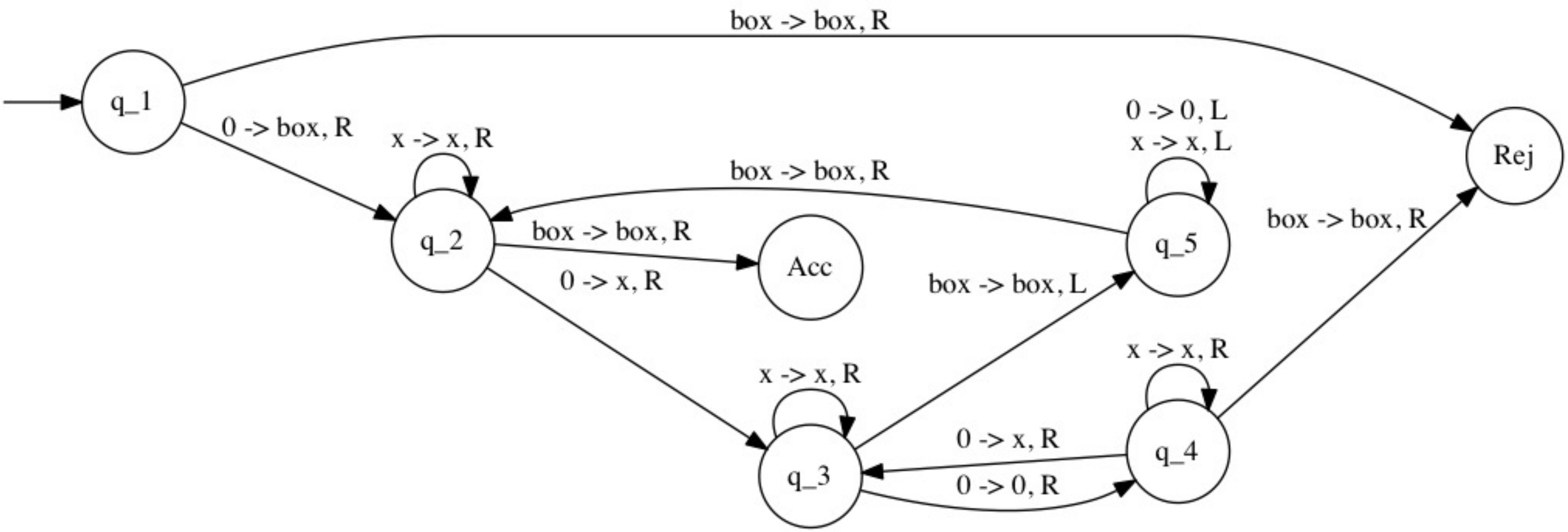
0s power of 2

Formally $M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{acc}, q_{rej})$

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_{acc}, q_{rej}\}$
- $\Sigma = \{\mathbf{0}\}$
- $\Gamma = \{\mathbf{0}, \mathbf{x}, \square\}$ (x is a character we use to ignore characters in subsequent processing)
- δ is given as a state diagram (next slide)
- The start, accept and reject states are q_1, q_{acc}, q_{rej}

- Recall the left-hand rule – if on the left edge of the input tape and trying to move left, just overwrite the character, and remain in place.

0s power of 2


box -> box, R
q_1
0 -> box, R
x -> x, R
q_2
box -> box, R
box -> box, R
Acc
0 -> x, R
0 -> 0, L
x -> x, L
q_5
box -> box, L
Rej
box -> box, R
x -> x, R
q_3
x -> x, R
q_4
0 -> x, R
0 -> 0, R

Sample Turing Machine (Cont' d)

Sample run of M_2 on input **0000**:

$q_1\mathtt{0000}$
$\square q_2\mathtt{000}$
$\square\mathtt{x}q_3\mathtt{00}$
$\square\mathtt{x0}q_4\mathtt{0}$
$\square\mathtt{x0x}q_3\square$
$\square\mathtt{x0}q_5\mathtt{x}\square$
$\square\mathtt{x}q_5\mathtt{0x}\square$

$\square q_5\mathtt{x0x}\square$
$q_5\square\mathtt{x0x}\square$
$\square q_2\mathtt{x0x}\square$
$\square\mathtt{x}q_2\mathtt{0x}\square$
$\square\mathtt{xx}q_3\mathtt{x}\square$
$\square\mathtt{xxx}q_3\square$
$\square\mathtt{xx}q_5\mathtt{x}\square$

$\square\mathtt{x}q_5\mathtt{xx}\square$
$\square q_5\mathtt{xxx}\square$
$q_5\square\mathtt{xxx}\square$
$\square q_2\mathtt{xxx}\square$
$\square\mathtt{x}q_2\mathtt{xx}\square$
$\square\mathtt{xx}q_2\mathtt{x}\square$
$\square\mathtt{xxx}q_2\square$
$\square\mathtt{xxx}\square q_{acc}$

Turing-Recognisable Languages

- A language is Turing Recognisable if some TM recognises it. This is what we called recursively enumerable before
- So if a TM recognises L, this means that all and only the elements of L are accepted by the TM. These strings result in the Accept state
- But what happens with the strings that are not in L?

Turing-Recognisable Languages

- But what happens with the strings that are not in L?
  - these might end in the Reject state ...
  - or the TM might never reach Accept or Reject on the inputs

Turing-Recognisable Languages (Cont'd)

- A TM fails to accept an input either by
  - Entering q_{rej} and rejecting the input
  - Looping forever
- It is not easy to distinguish a machine that is looping from one that is just taking a long time!
  - Loops can be complex, not just $C_i \rightarrow C_j \rightarrow C_i \rightarrow C_j \rightarrow \ldots$

Turing-Recognisable Languages (Cont'd)

- We prefer TMs that halt on all inputs
  - They are called deciders
  - A decider that recognises a language is said to decide that language
  - A decider answers every question of the form "Does the TM accept this string?" in finite time
  - Non-deciders keep you guessing on some strings

Turing-Decidable Languages

- A language is (Turing-)decidable if some TM decides it
  – Every decidable language is Turing-recognisable
  – Some Turing-recognisable languages are not decidable (though you haven't seen any examples yet)

- We return to this topic later. It is another way of saying there are problems without solutions.

Variants of Turing Machines

- There are many alternative definitions of TMs
  - They are called variants of the Turing machine
- They all have the same power as the original TM
- Some of these deviate from the TMs we have seen so far in minor ways

Variants of Turing Machines

- Some TMs put strings in between two infinite sequences of blanks. This avoids the need that sometimes exists to mark the start of the string
- Some TMs halt only when they reach a state from which no transitions are possible. Compare our TM: “sudden death” after reaching q_{acc} , q_{rej}
- Let’s briefly look at some more interesting variants

Multitape Turing Machines

- A TM with several tapes instead of just one
- Each tape has its own head for reading/writing
  - Initially, input appears on tape 1; other tapes are blank
- Transition function caters for k different tapes:

$$\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{\mathrm{L}, \mathrm{R}\}^k$$

- The expression

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, \mathrm{L}, \mathrm{R}, \ldots, \mathrm{L})$$

Means

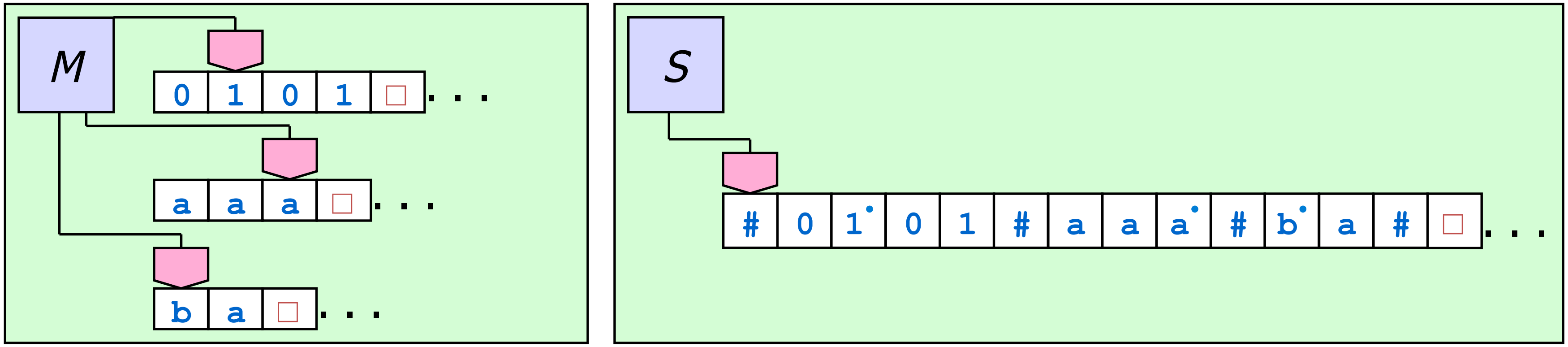
  - If in q_i and heads 1 through k read a_1 through a_k,
  - Then go to q_j, write b_1 through b_k and move heads to the left or right, as specified

Multitape Turing Machines (Cont'd)

Theorem:

Every multitape Turing machine has an equivalent single tape Turing machine.

Proof: show how to convert a multitape TM *M* to an equivalent single tape TM *S*. (Sketch only!)



Nondeterministic Turing Machines

- This time, at a given point in the computation, there may be various possibilities to proceed
- Transition function maps to powerset (options):

$$\delta : Q \times \Gamma \rightarrow 2^{(Q \times \Gamma \times \{\mathrm{L},\mathrm{R}\})}$$

- The expression

$$\delta(q_i, a) = \{(q_j, b_1, \mathrm{L}), \ldots, (q_j, b_2, \mathrm{R})\}$$

Means

  - If in q_i and head is on a
  - Then go to any one of the options (some finite number!)
- If there is a sequence of choices that lead to the accept state, the machine accepts the input

Nondeterministic TMs (Cont'd)

Theorem:

For every nondeterministic Turing machine there exists an equivalent deterministic Turing machine

Equivalent means:

-- accepts the same strings

-- rejects the same strings

(No proof offered here.)

Other types of TMs

- These lectures focus on TMs for accepting strings (recognizing languages)
- Other TMs were built primarily for manipulating strings. Examples:
  - TMs that count the number of symbols in a string (by producing a string that contains the right number of 1s)
  - TMs that adds up two bit strings (by producing a string that contains the right number of 1s)
  - See your Practical
- Such TMs do not need Accept/Reject states.
- Their behaviour can be simulated in the earlier (simpler) TMs: accept if the tape contains inputs + the correct output.

TMs are cumbersome to specify in detail

- TMs serve as a precise model for algorithms
- We shall often make do with informal descriptions
- We need to be comfortable enough with TMs to believe they capture all algorithms

Notation for TMs

- Input to a TM is always a string
- If we want to provide something else such as a polynomial, matrix, list of students, etc. then we need to
  - Represent these as strings (somehow), and
  - Program the TM to decode/act on the representation
- Notation:
  - Encoding of O as a string is $\langle O \rangle$
  - Encoding of $O_1, O_2, \ldots, O_k$ as a string is $\langle O_1, O_2, \ldots, O_k \rangle$