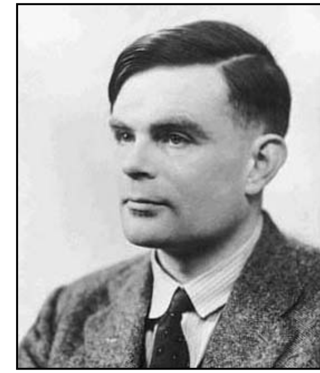
Turing Machines
Part 1

Adam Wyner
CS3518, Spring 2017
University of Aberdeen

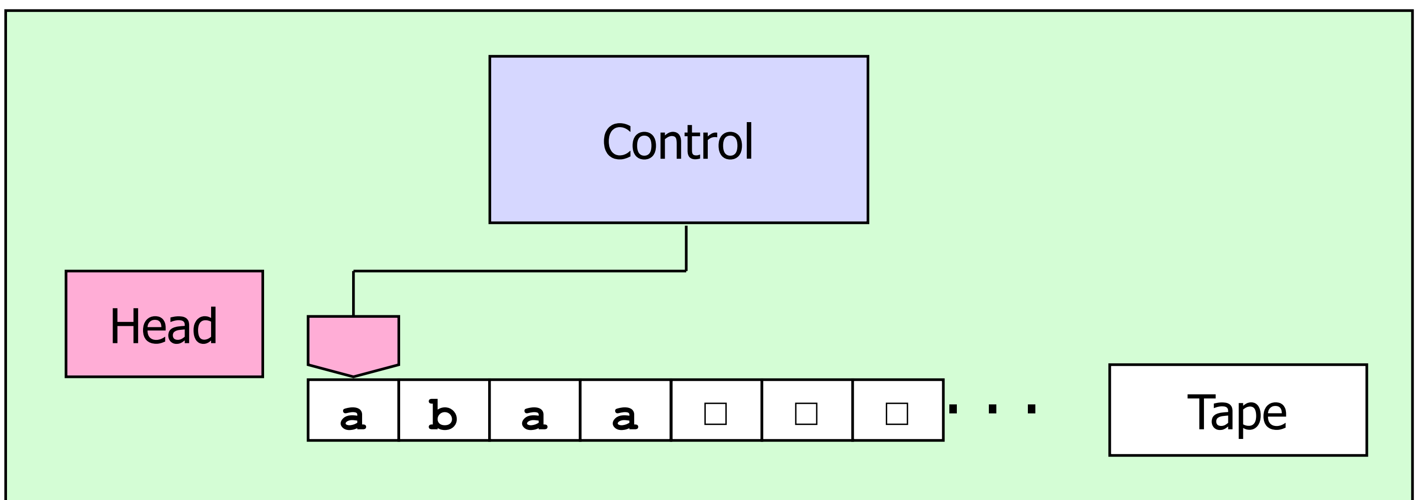
Turing Machines



- Abstract but accurate model of computers
- Proposed by Alan Turing in 1936
- Turing's motivation: find out whether there exist mathematical problems that cannot be solved algorithmically, e.g. Hilbert's decision problem
- Similar to a FSA but with an unlimited and unrestricted memory
- Able to do everything a real computer can do
- However: There are problems even a Turing Machine (TM) can't solve, which are beyond the theoretical limits of computation

Turing Machines: Features

- A tape is a countably infinite list of cells
- A tape head can
  - read and write symbols from/onto a cell on the tape
  - move forward and backward on the tape
- Schematically:



Turing Machines: Features (Cont'd)

- Initially the tape contains only the input string and is blank everywhere else
- To store information, the TM writes it onto the tape
- To read the info it has written, TM can move the head back over it
- TMs have the following behaviours:
  - They "compute" then stop in a "reject" state
  - They "compute" then stop in an "accept" state
  - They loop forever
- Compare to FSAs, which have no "reject" states and no looping forever.

Turing Machines vs. Finite Automata

- TMs can both write on the tape and read from it
  - FSAs can only read (or only write if they are generators)
- TM's read/write head can move to the left and right
  - FSAs cannot "go back" on the input
- TM's tape is infinite
  - In FSAs the input and output is always finite
- TM's accept/reject states take immediate effect
  - There is no need to "consume" all the input

- TMs come in different flavours. Their differences are unimportant

Turing Machine: an informal Example

- TM M_1 to test if an input (on the tape) belongs to

$$B = \{w\#w \mid w \in \{0,1\}^* \}$$

M_1 checks if the contents of the tape consists of two identical strings of 0's & 1's separated by "#"

  - M_1 accepts inputs

    0110#0110, 000#000, 010101010#010101010

  - M_1 rejects inputs

    1111#0000, 000#0000, 0001#1000

Turing Machine: an informal Example

- M_1 should work for any size of input (general)
  - There are infinitely many, so we cannot use "cases"
- How would you program this?
  - All you have is a tape, and a head to read/write it

Turing Machine: an Example (Cont'd)

- Strategy for M_1:
  - "Zig-zag" to corresponding places at both sides of "**#**" and check if they match
  - Mark (cross off) those places you have checked
  - If it crosses off all symbols, then everything matches and M_1 goes into an accept state
  - If it discovers a mismatch then it enters a reject state

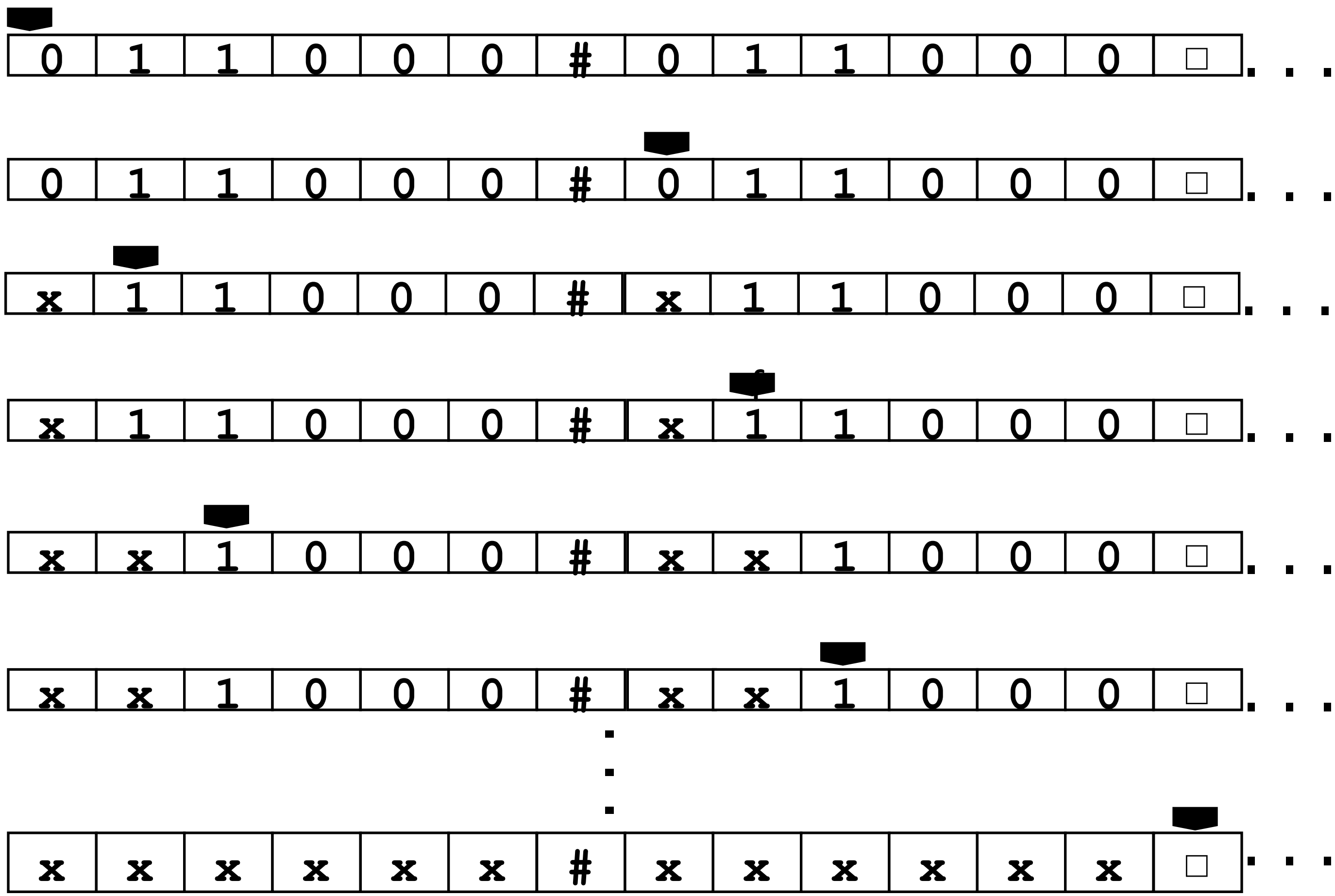
Turing Machine: an Example (Cont'd)

- M_1 in effect says, on input string S
  1. Scan S to verify there is a single **#** symbol; if not, reject.
  2. Zig-zag across the tape to corresponding positions on either side of **#** to check if they contain the same symbol.
     - If they do not match, reject.
     - Cross off symbols as they are checked to keep track of things.
  3. When all symbols to the left of **#** have been crossed off, check for any remaining symbols to the right of **#**.
     - If any symbols remain, reject; otherwise accept.

Turing Machine: an Example (Cont'd)

- Strategy:
  1. start at the leftmost of the tape, char x_1
  2. go to the symbol s_1 leftmost relative to the #
  3. check if x_1 and s_1 match.
     a. if so, cross off x_1 and s_1 and continue
     b. otherwise reject.
  4. go to the leftmost symbol of the tape not crossed off
  5. go to the symbol leftmost relative to the # and not crossed off
  6. check match....
  7. iterate from 3 and considering previous conditions.
- The blank symbol right of # is ☐

Turing Machine: an Example (Cont'd)



Accept

Turing Machine: an Example (Cont’d)

0	1	1	0	0	0	#	0	1	1	0	0	0	□	. . .

0	1	1	0	0	0	#	0	1	1	0	0	0	□	. . .

x	1	1	0	0	0	#	x	1	1	0	0	0	□	. . .

x	1	1	0	0	0	#	x	0	1	0	0	0	□	. . .

Reject

Formal Definition of Turing Machines

- Previous slides give a flavour of TMs, but not their details.
- We can describe TMs formally, similarly to what you did for FAS
- We shall not always use formal descriptions for TMs because these would tend to be quite long
- But ultimately, our informal descriptions should be "translatable" into formal ones, so it is crucial to understand these formal descriptions

Formal definition of Turing Machines

- The imperative model of “algorithmhood” is formulated in terms of actions:
  - A TM is always in one of a specified number of states (these include the accept and reject states)
  - The action of the TM, at a given moment, depends on its state and on what the TM is reading at that moment
  - A TM always performs three types of actions: (1) replace the symbol that it reads by another (or the same) symbol, (2) move the head Left or Right, and (3) enter a new state (or the same state again).

Formal Definition of TMs (Cont'd)

- The heart of a TM is a function δ mapping from
  - A state *s* of the machine, and
  - The symbol *a* on the tape where the head is

  to
  - A new state *t* of the machine, and
  - A symbol *b* to be written on the tape (over *a*), and
  - A movement L (left) or R (right) of the head

Formal Definition of TMs (Cont'd)

$$\delta(s,a) = (t,b,\mathrm{L})$$

- When the machine is in state *s* and the head is over a cell containing a symbol *a*, then the machine writes the symbol *b* (replacing *a*), goes to state *t* and moves the head to the Left.

Formal Definition of TMs (Cont' d)

A Turing Machine is a 7-tuple

$$(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$$

where Q, Σ, Γ are all finite sets and

1. Q is the set of states
2. Σ is the input alphabet not containing the special blank symbol "□"
3. Γ is the tape alphabet, where $\Gamma \subseteq \Sigma \cup \{□\}$
4. $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function
5. $q_0 \in Q$ is the start state
6. $q_{acc} \in Q$ is the accept state
7. $q_{rej} \in Q$ is the reject state, where $q_{rej} \neq q_{acc}$

How Turing Machines "Compute"

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$

- M receives its input $w = w_1 w_2 \ldots w_n \in \Sigma^*$
  - w is on the leftmost n cells of the tape.
  - The rest of the tape is blank (infinite list of "□")
- The head starts on the leftmost cell of the tape
  - Σ does not contain "□", so the first blank on the tape marks the end of the input
- M follows the "moves" encoded in δ
  - If M tries to move its head to the left of the left-hand end of the tape, the head stays in the same place
  - The computation continues until it enters the accept or reject state, at which point M halts
  - If neither occurs, M goes on forever…

Formalisation of TMs

- Formalisation is needed to say precisely how TMs behave.
- TMs use “sudden death” (and “sudden life”)
  - FSA accepts a string iff the string is the label of a successful path.
  - TM accepts a string if, while processing it, an accept state is reached; TM rejects a string if, while processing it, a reject state is reached
- What exactly does it mean to “move to the right”?
  - What if the head is already at the rightmost edge?

Configurations of TMs

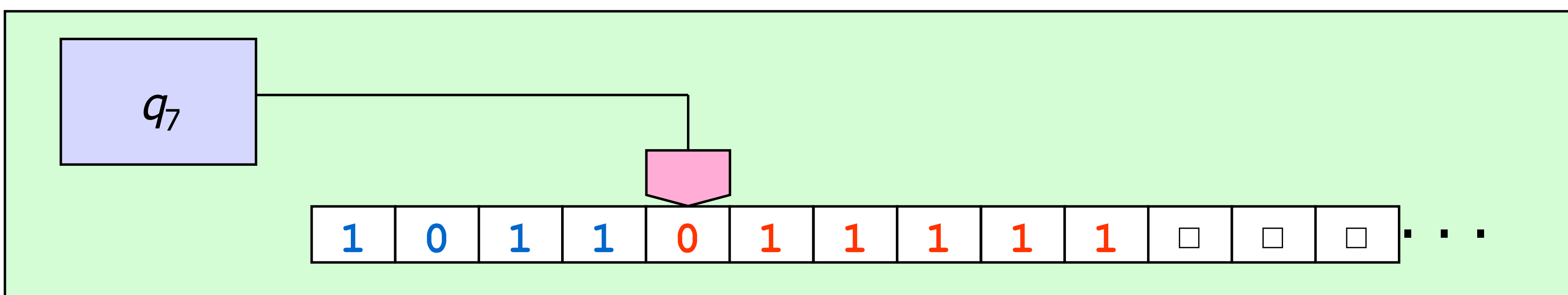
- As a TM computes, changes occur in its:
  - State
  - Tape contents
  - Head location

These three items are a configuration of the TM

- Configurations are represented in a special way:
  - When the TM is in state q, and
  - The contents of the tape is two strings uv, and
  - The head is on the leftmost position of string v
  - Then we represent this configuration as uqv

Configurations of TMs

- Configurations are represented in a special way:
  - When the TM is in state *q7*, and
  - The contents of the tape is two strings u = 1011 and v = 011111, and
  - The head is on the leftmost position of string *v*
  - Then we represent this configuration as *uqv*
- Example: $1011q_7011111$
- Same thing as the picture:



Formalising TMs Computations

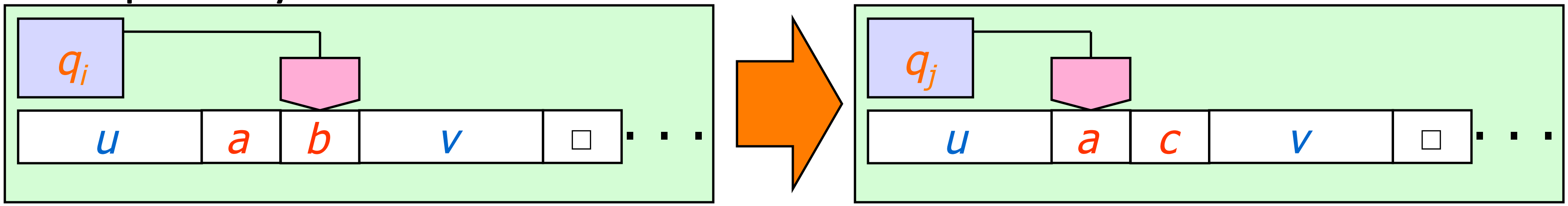
Assume

- a, b, c in Γ (3 characters from the tape)
- u and v in Γ^* (2 strings from the tape)
- states q_i and q_j

We can now say that (for all u and v)

$$u\,a\,q_i\,b\,v \text{ yields } u\,q_j\,a\,c\,v \text{ if } \delta(q_i, b) = (q_j, c, \mathrm{L})$$

Graphically:

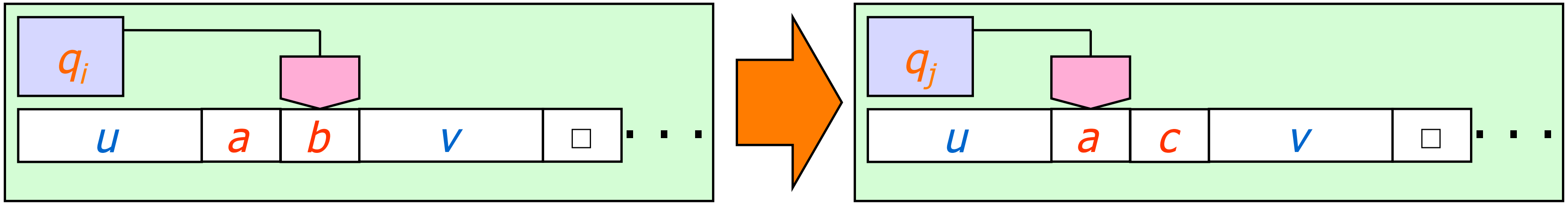


Formalising TMs Computations

We can now say that (for all u and v)

$$u\,a\,q_i\,b\,v \text{ yields } u\,q_j\,a\,c\,v \text{ if } \delta(q_i, b) = (q_j, c, \mathrm{L})$$

When the machine is in state q_i and the head is over a cell containing a symbol b, then the machine writes the symbol c (replacing b), goes to state q_j and moves the head to the Left.

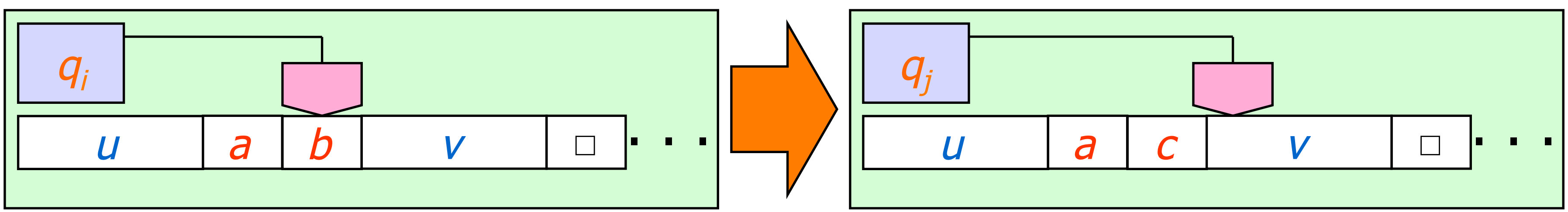


Formalising TMs Computations (Cont'd)

A similar definition for rightward moves

$$ua\, q_i\, bv \text{ yields } uacq_j\, v \text{ if } \delta(q_i, b) = (q_j, c, \mathrm{R})$$

Graphically:

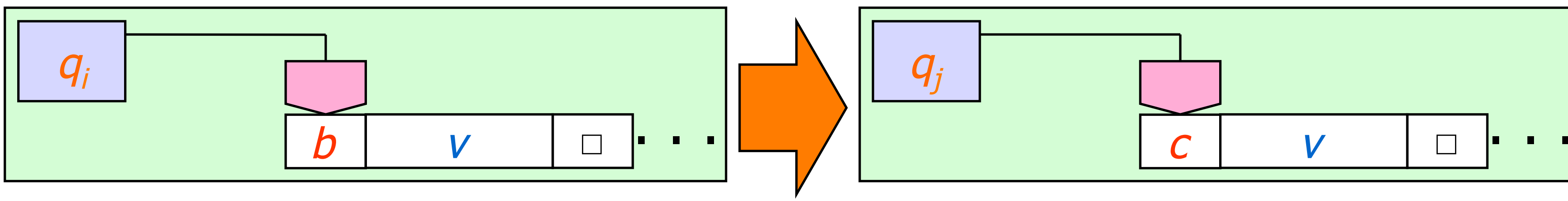


Formalising TMs Computations (Cont'd)

- Special cases when head at beginning of tape
- For the left-hand end (q_i on the left), moving left:

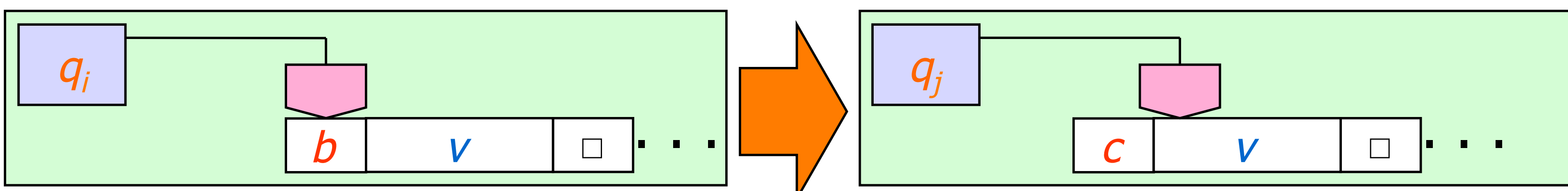
$$q_i\, bv \text{ yields } q_j\, c\, v \quad \text{if } \delta(q_i, b) = (q_j, c, \text{L})$$

We prevent the head from "falling off" the left-hand end of the tape:



- For the left-hand end, moving right:

$$q_i\, bv \text{ yields } cq_j\, v \quad \text{if } \delta(q_i, b) = (q_j, c, \text{R})$$



Formalising TMs Computations (Cont'd)

- For the right-hand "end" (not really the end…)
  - infinite sequence of blanks follows the part of the tape represented in the configuration
- We thus handle the case above as any other rightward move

Formalising TMs Computations (Cont'd)

- The start configuration of M on input w is q_0w
  - M in start state q_0 with head at leftmost position of tape
- An accepting configuration has state q_{acc}
- A rejecting configuration has state q_{rej}
- Rejecting and accepting configurations are halting configurations
  - They do not yield further configurations
  - No matter what else is in the configuration!

- Note: δ is a function, and there is only one start state, so the TM (as defined here) is deterministic

Formalising TMs Computations (Cont'd)

- An accepting configuration has state q_{acc}
- A rejecting configuration has state q_{rej}

- Notice here that "why" something is an accepting or rejecting configuration is not specified here. Just what the "program" does is given by the transition function:
  - δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$, where q_{acc}, $q_{rej} \in Q$

Formalising TMs Computations (Cont'd)

- A TM M accepts input w if there is a sequence of configurations $C_1, C_2, \ldots, C_k$ where
  1. C_1 is the start configuration of M on input w
  2. Each C_i yields C_{i+1}, and
  3. C_k is an accepting configuration

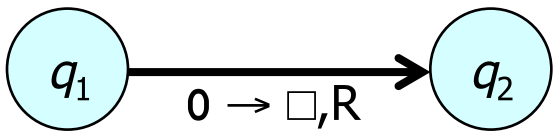
Analogous to FSAs,

- the set of strings that M accepts is the language of M, denoted $L(M)$
- we also say that a TM "accepts" or "recognizes" a language.

Sample Turing Machine (Cont'd)

Some shorthands to simplify notation:

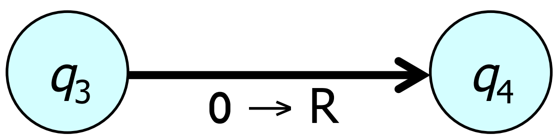
1.



$$\delta(q_1,0) = (q_2,\square,R)$$

"when in state q_1 with the head reading **0**, it goes to state q_2, writes $\square$ and moves the head to the right"

2.



$$\delta(q_3,0) = (q_4,0,R)$$

"machine moves its head to the right when reading 0 in the state q_3, but nothing is written onto the tape"

Transitions from accept/reject states can be omitted (because of sudden death).