Turing machines

Ashley Montanaro
ashley@cs.bris.ac.uk

Department of Computer Science, University of Bristol
Bristol, UK

21 March 2014

Introduction

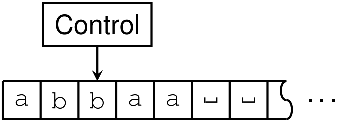
We have seen two models of computation: finite automata and pushdown automata. We now discuss a model which is much more powerful: the Turing machine.

Introduction

We have seen two models of computation: finite automata and pushdown automata. We now discuss a model which is much more powerful: the Turing machine.

A Turing machine is like a finite automaton, with three major differences:

- It can write to its tape;
- It can move both left and right;
- The tape is infinite in one direction.

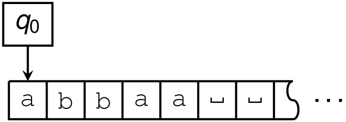


Initially, the input is provided on the left-hand end of the tape, and followed by an infinite sequence of blank spaces ("␣").

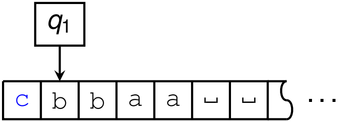
Ashley Montanaro
ashley@cs.bris.ac.uk

University of BRISTOL

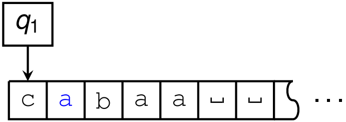
Example



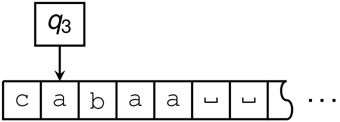
Example



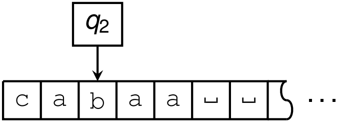
Example



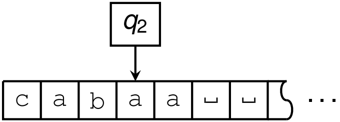
Example



Example

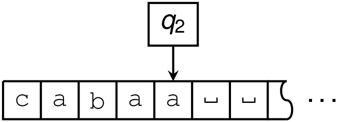


Example

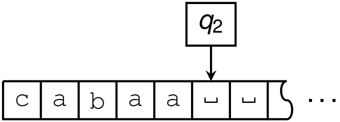


Ashley Montanaro
ashley@cs.bris.ac.uk

Example

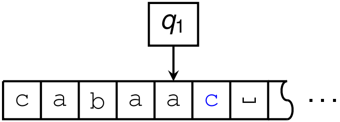


Example



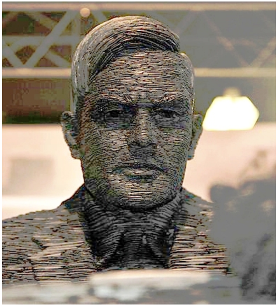
Ashley Montanaro
ashley@cs.bris.ac.uk

Example



Alan Turing (1912-1954)

- 1936: Invented the Turing machine and the concept of computability.
- 1939-1945: Worked at Bletchley Park on cracking the Enigma cryptosystem and others.
- 1946-1954: Work on practical computers, AI, mathematical biology, ...
- 1952: Convicted of indecency. Died of cyanide poisoning in 1954.
- 2014: Received a royal pardon.



Pic: Wikipedia/Bletchley Park

Turing machines

- Turing machines have two special states: an accept state and a reject state.

Turing machines

- Turing machines have two special states: an accept state and a reject state.
- If the machine enters the accept or reject state, it halts (stops).

Turing machines

- Turing machines have two special states: an accept state and a reject state.
- If the machine enters the accept or reject state, it halts (stops).
- If it doesn't ever enter either of these states, it never halts (i.e. it runs forever).

Turing machines

- Turing machines have two special states: an accept state and a reject state.
- If the machine enters the accept or reject state, it halts (stops).
- If it doesn't ever enter either of these states, it never halts (i.e. it runs forever).
- The language $L(M)$ recognised by a Turing machine M is the set

$$\{x \mid M \text{ halts in the accept state on input } x\}$$

Turing machines

- Turing machines have two special states: an accept state and a reject state.
- If the machine enters the accept or reject state, it halts (stops).
- If it doesn't ever enter either of these states, it never halts (i.e. it runs forever).
- The language $L(M)$ recognised by a Turing machine M is the set

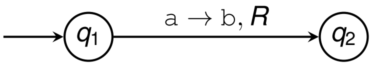
$$\{x \mid M \text{ halts in the accept state on input } x\}$$

- For some language $\mathcal{L}$, if there exists a Turing machine M such that $\mathcal{L} = L(M)$, we say that $\mathcal{L}$ is Turing-recognisable. (These languages are also sometimes called recursively enumerable.)

Describing Turing machines

We can describe a Turing machine by its state diagram.

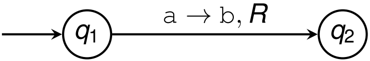
- As with DFAs and PDAs, we have a graph whose vertices are labelled by states of the machine, and whose edges are labelled by possible transitions.



Describing Turing machines

We can describe a Turing machine by its state diagram.

- As with DFAs and PDAs, we have a graph whose vertices are labelled by states of the machine, and whose edges are labelled by possible transitions.



- A label of the form

$$\mathtt{a} \rightarrow \mathtt{b}, R$$

  means that on reading tape symbol $\mathtt{a}$, the machine writes $\mathtt{b}$ to the tape and then moves right.

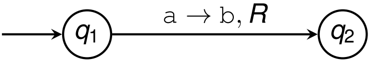
Ashley Montanaro
ashley@cs.bris.ac.uk

University of BRISTOL

Describing Turing machines

We can describe a Turing machine by its state diagram.

- As with DFAs and PDAs, we have a graph whose vertices are labelled by states of the machine, and whose edges are labelled by possible transitions.



- A label of the form

$$\mathtt{a} \to \mathtt{b}, R$$

  means that on reading tape symbol $\mathtt{a}$, the machine writes $\mathtt{b}$ to the tape and then moves right.

- Another example: a label

$$\mathtt{a}, \mathtt{b} \to L$$

  means that on reading either tape symbol $\mathtt{a}$ or $\mathtt{b}$, the machine doesn't write anything to the tape and then moves left.

Turing machines

Imagine we want to test membership in the language

$$\mathcal{L}_{EQ} = \{w\#w \mid w \in \{0,1\}^*\}.$$

$x \in \mathcal{L}_{EQ}$ if it is made up of two equal bit-strings, separated by a $\#$ symbol.

Turing machines

Imagine we want to test membership in the language

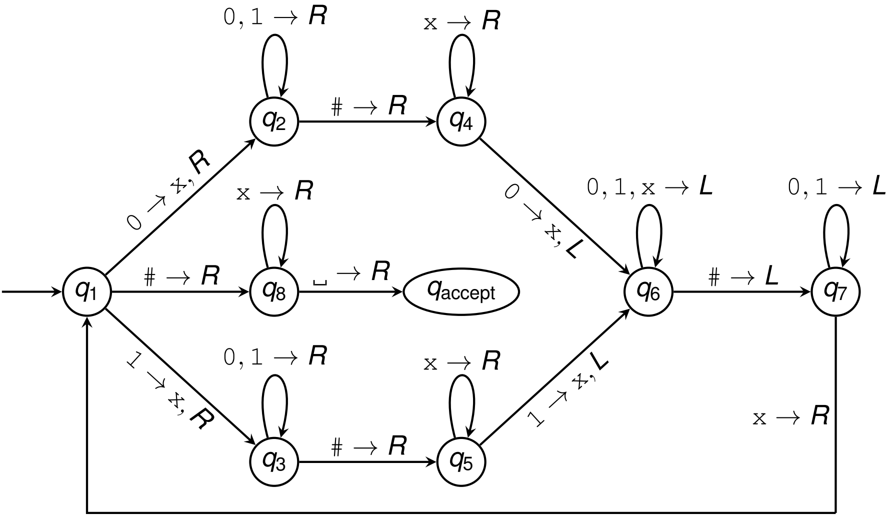
$$\mathcal{L}_{EQ} = \{w\#w \mid w \in \{\mathtt{0}, \mathtt{1}\}^*\}.$$

$x \in \mathcal{L}_{EQ}$ if it is made up of two equal bit-strings, separated by a $\#$ symbol.

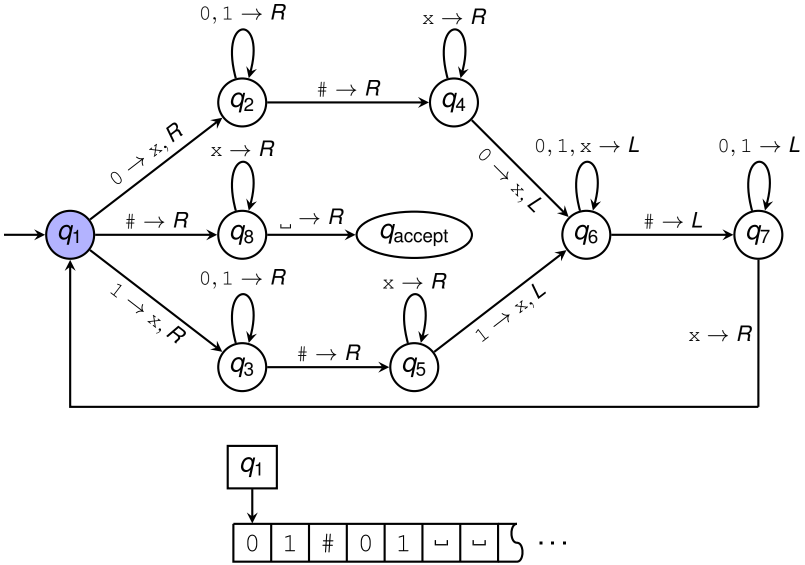
Idea for recognising this language

1. Our algorithm will scan forwards and backwards, testing each corresponding pair of bits either side of the $\#$ for equality in turn.
2. We can overwrite each bit with an $\mathtt{x}$ symbol after checking it so we don't check the same bits twice.

State diagram (Sipser, Figure 3.10)

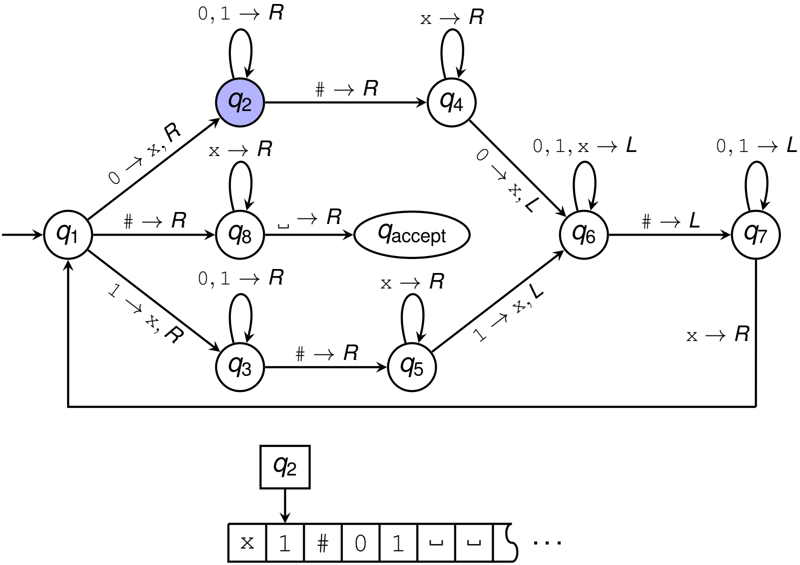


Example: testing whether `01#01` $\in \mathcal{L}_{EQ}$

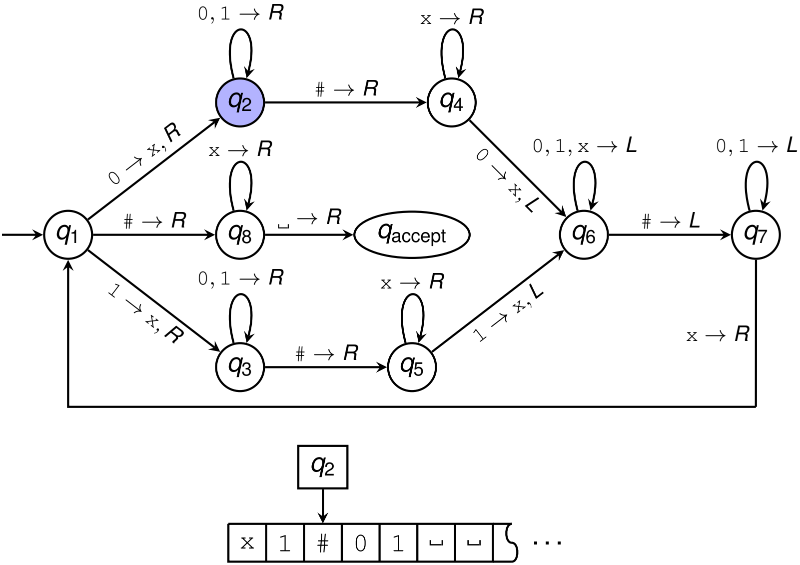


Ashley Montanaro
ashley@cs.bris.ac.uk

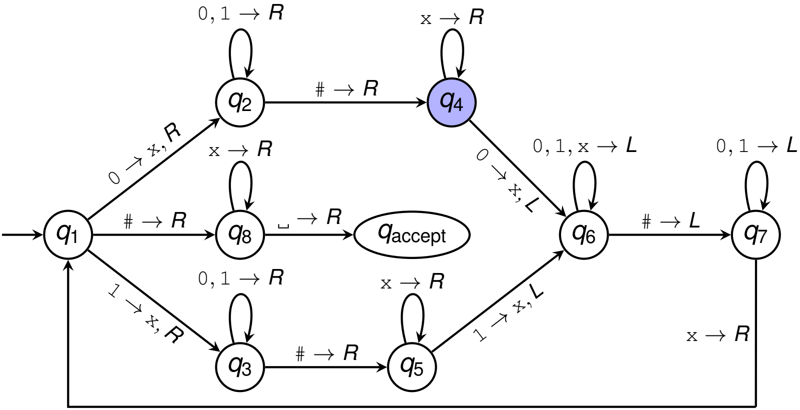
Example: testing whether $\texttt{01\#01} \in \mathcal{L}_{EQ}$

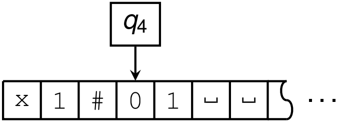


Example: testing whether $\texttt{01\#01} \in \mathcal{L}_{EQ}$



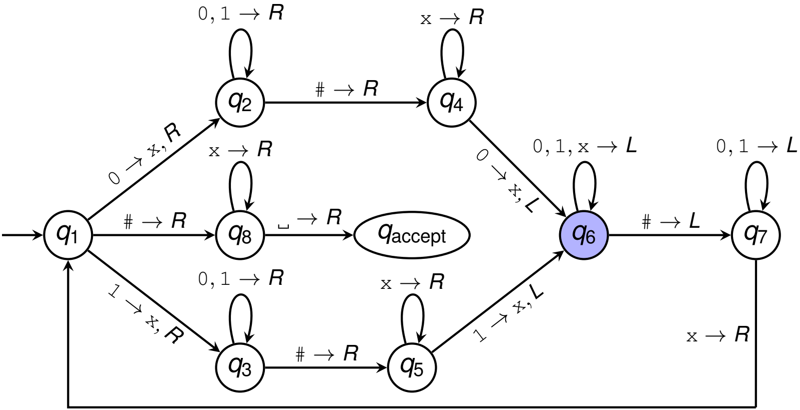
Example: testing whether $\texttt{01\#01} \in \mathcal{L}_{EQ}$

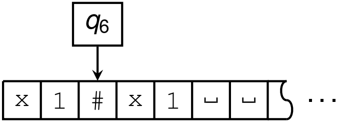




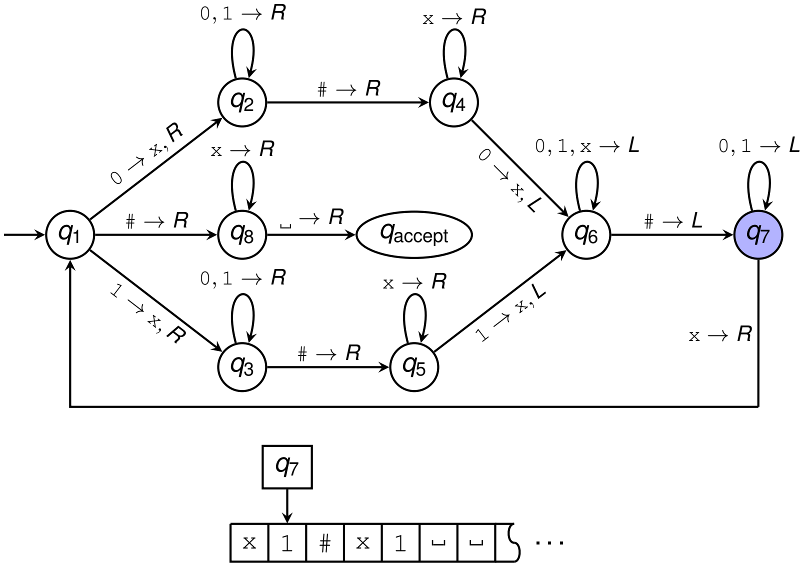
Ashley Montanaro
ashley@cs.bris.ac.uk

Example: testing whether $\texttt{01\#01} \in \mathcal{L}_{EQ}$



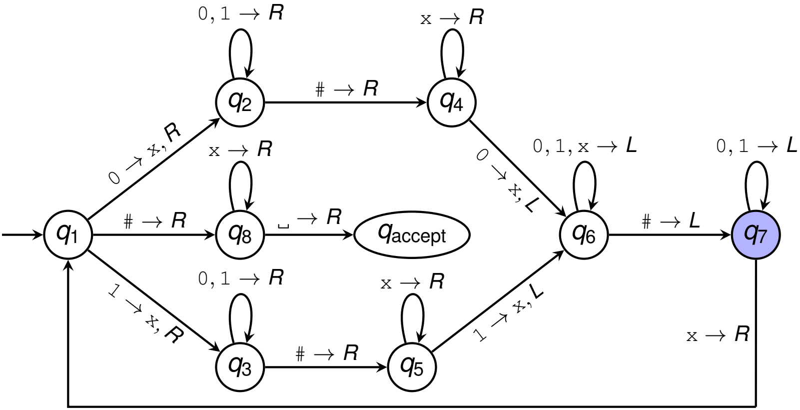


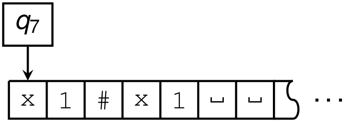
Example: testing whether $\texttt{01\#01} \in \mathcal{L}_{EQ}$



Ashley Montanaro
ashley@cs.bris.ac.uk

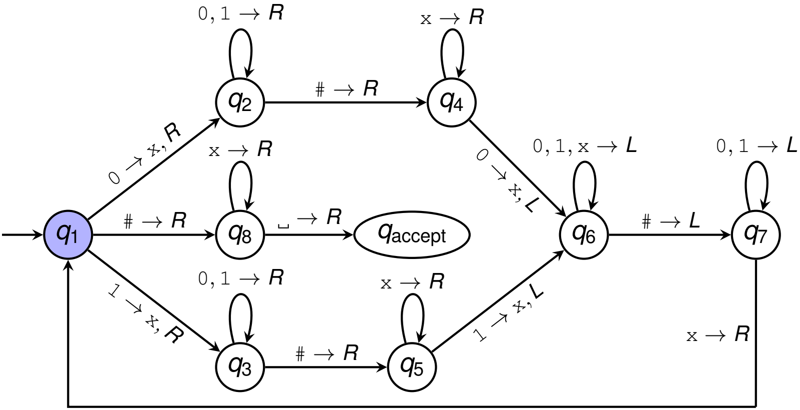
Example: testing whether $\texttt{01\#01} \in \mathcal{L}_{EQ}$

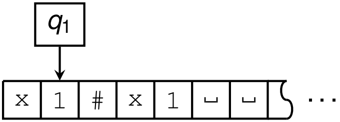




Ashley Montanaro
ashley@cs.bris.ac.uk

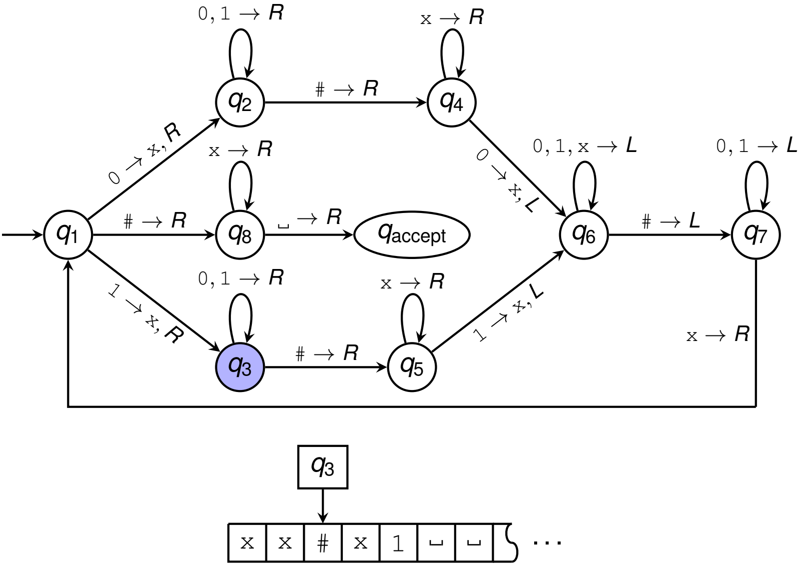
Example: testing whether $\texttt{01\#01} \in \mathcal{L}_{EQ}$





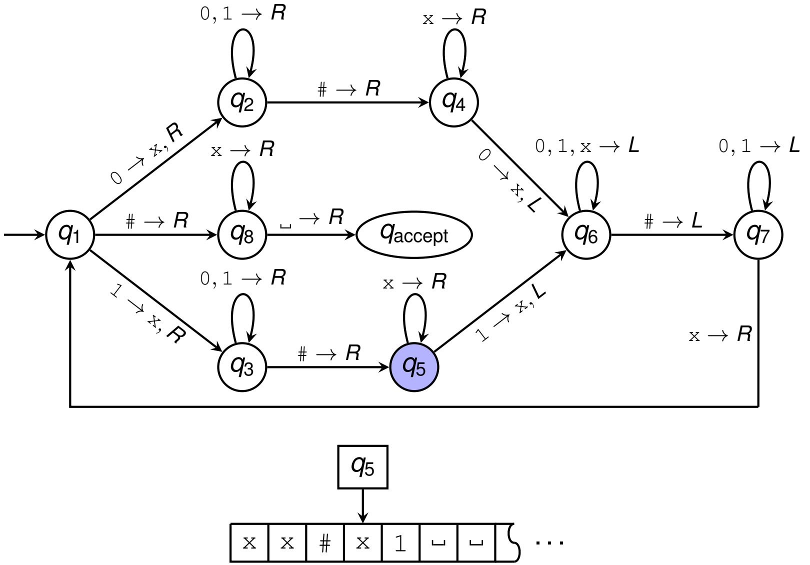
Ashley Montanaro
ashley@cs.bris.ac.uk

Example: testing whether 01#01 $\in \mathcal{L}_{EQ}$



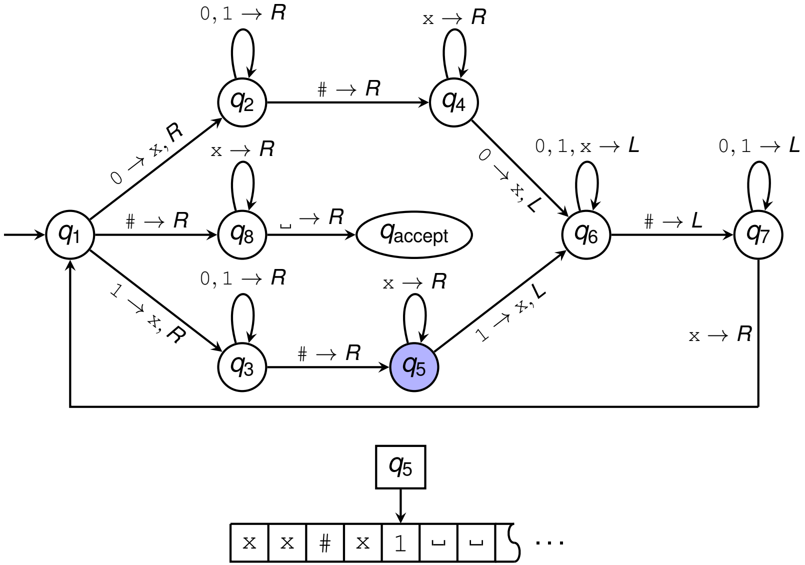
Ashley Montanaro
ashley@cs.bris.ac.uk

Example: testing whether $\texttt{01\#01} \in \mathcal{L}_{EQ}$

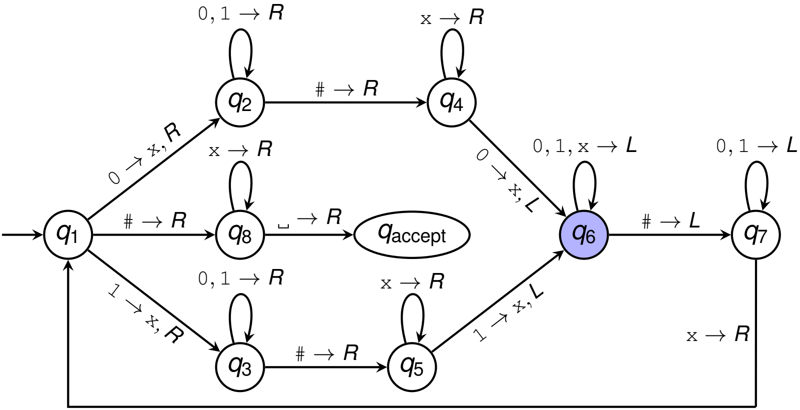


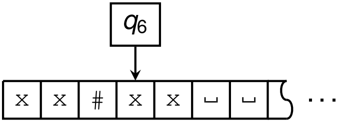
Ashley Montanaro
ashley@cs.bris.ac.uk

Example: testing whether $\texttt{01\#01} \in \mathcal{L}_{EQ}$

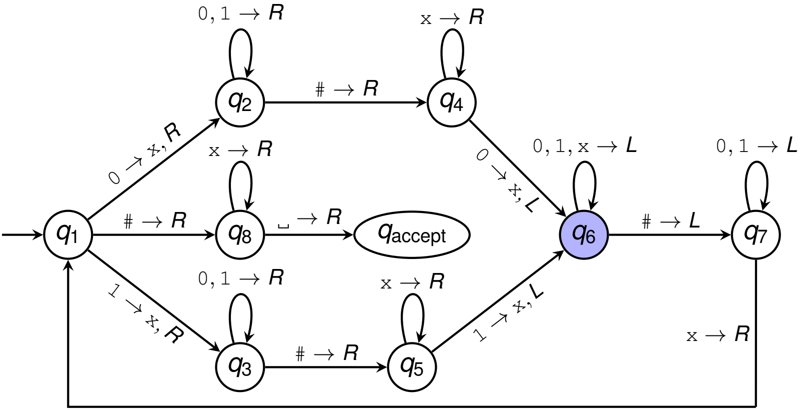


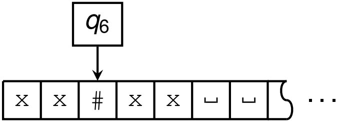
Example: testing whether 01#01 $\in \mathcal{L}_{EQ}$





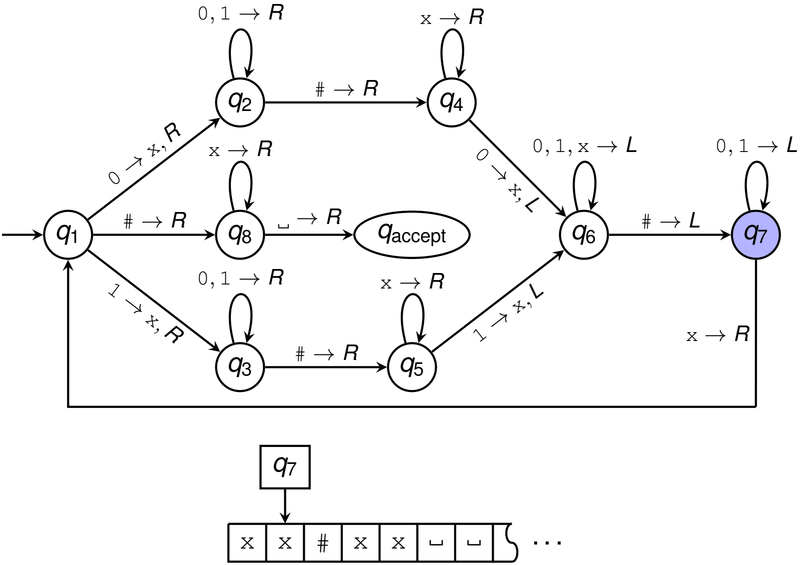
Example: testing whether `01#01` $\in \mathcal{L}_{EQ}$





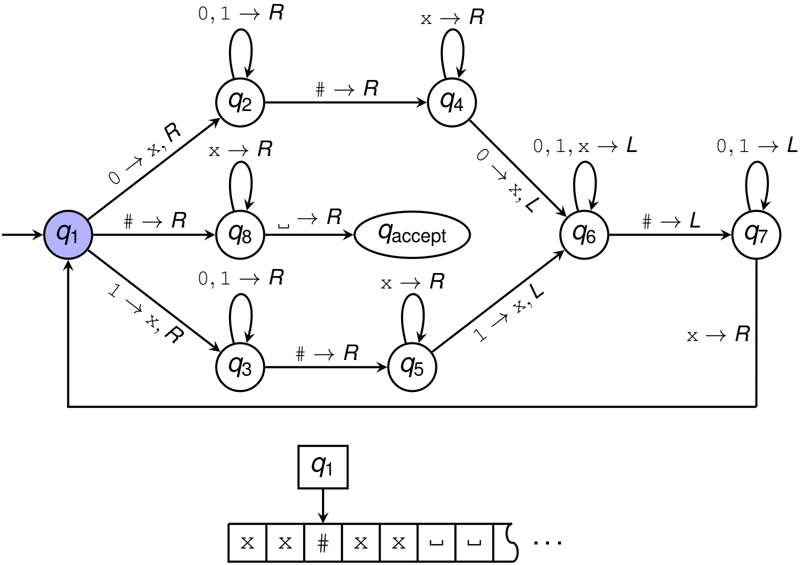
Ashley Montanaro
ashley@cs.bris.ac.uk

Example: testing whether $\texttt{01\#01} \in \mathcal{L}_{EQ}$

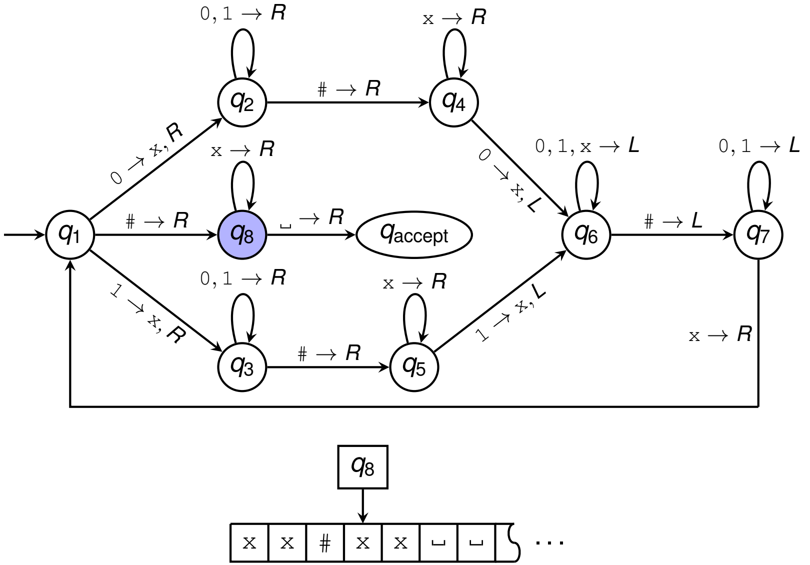


Ashley Montanaro
ashley@cs.bris.ac.uk

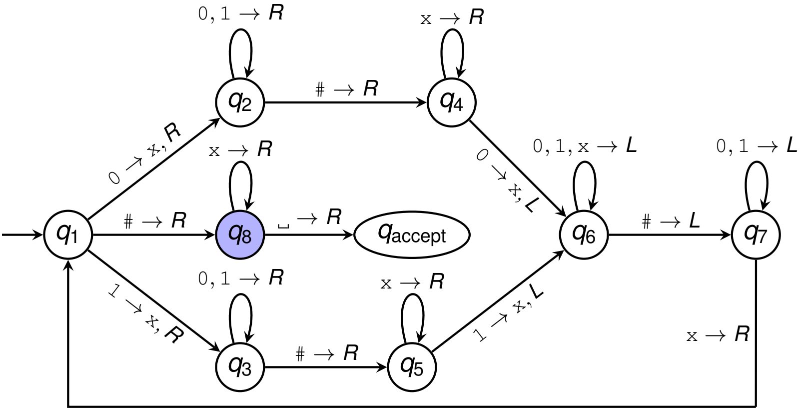
Example: testing whether $\texttt{01\#01} \in \mathcal{L}_{EQ}$

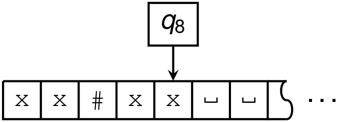


Example: testing whether 01#01 $\in \mathcal{L}_{EQ}$



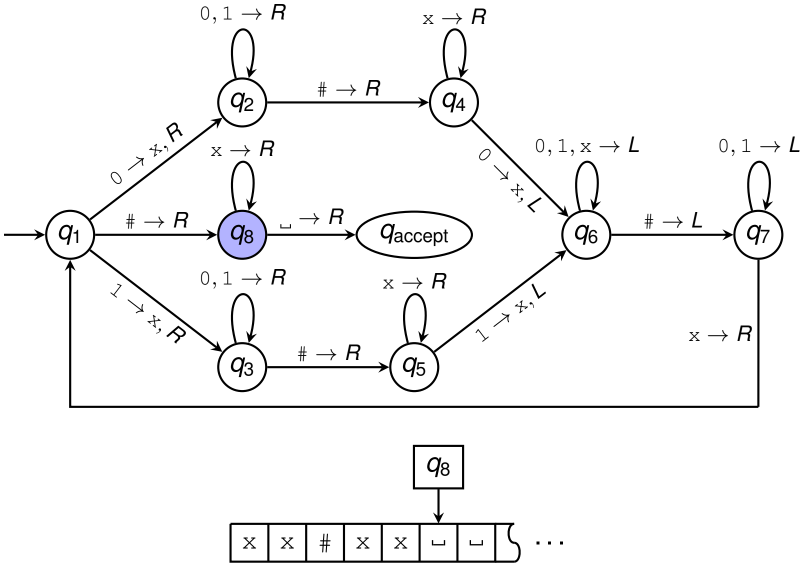
Example: testing whether $\texttt{01\#01} \in \mathcal{L}_{EQ}$



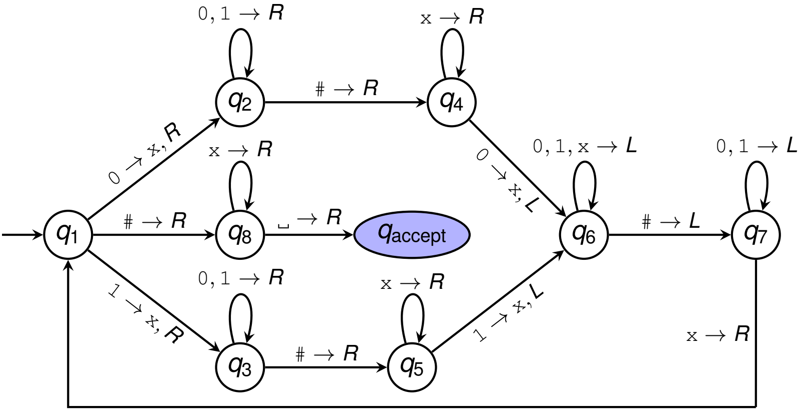


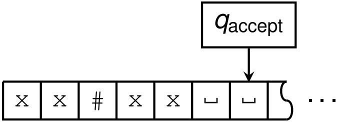
Ashley Montanaro
ashley@cs.bris.ac.uk

Example: testing whether $\texttt{01\#01} \in \mathcal{L}_{EQ}$



Example: testing whether $\texttt{01\#01} \in \mathcal{L}_{EQ}$





Turing machines: formal definition

Definition

A Turing machine is described by a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where:

1. Q is the set of states,
2. Σ is the input alphabet (which must not contain ␣),
3. Γ is the tape alphabet, where ␣ $\in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. q_{accept} is the accept state,
7. q_{reject} is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

Example

The Turing machine we just saw is described by

$$(\{q_1, \ldots, q_8, q_{\text{accept}}, q_{\text{reject}}\},\ \{\mathtt{0}, \mathtt{1}, \#\},\ \{\mathtt{0}, \mathtt{1}, \#, \mathtt{x}, \sqcup\},\ \delta,\ q_1,\ q_{\text{accept}},\ q_{\text{reject}})$$

where the transition function δ is defined by the table

	$\mathtt{0}$	$\mathtt{1}$	$\#$	$\mathtt{x}$	$\sqcup$
q_1	$q_2, \mathtt{x}, R$	$q_3, \mathtt{x}, R$	q_8, R		
q_2	q_2, R	q_2, R	q_4, R		
q_3	q_3, R	q_3, R	q_5, R		
q_4	$q_6, \mathtt{x}, L$			q_4, R	
q_5		$q_6, \mathtt{x}, L$		q_5, R	
q_6	q_6, L	q_6, L	q_7, L	q_6, L	
q_7	q_7, L	q_7, L		q_1, R	
q_8				q_8, R	q_{accept}, R

Blank entries in the table correspond to transitions where the machine rejects.

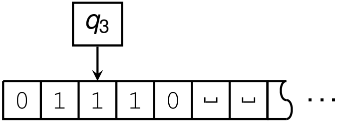
Turing machines: formal definition (2)

- The full description of what a Turing machine M is doing at any point in time is called its configuration.

- We write uqv for the configuration where:
  - the tape to the left of the current position contains u;
  - the tape to the right of the current position (and including the current position) contains v (followed by an infinite number of $\sqcup$'s);
  - the machine is in state q.

Turing machines: formal definition (2)

- The full description of what a Turing machine M is doing at any point in time is called its configuration.
- We write uqv for the configuration where:
  - the tape to the left of the current position contains u;
  - the tape to the right of the current position (and including the current position) contains v (followed by an infinite number of $\sqcup$'s);
  - the machine is in state q.
- For example, $\texttt{01}q_3\texttt{110}$ describes the following situation:



Turing machines: formal definition (3)

- Configuration C_1 yields C_2 if M can go from C_1 to C_2 in one step. So:
  - uaq_jbv yields uq_kacv if $\delta(q_j, b) = (q_k, c, L)$;
  - uaq_jbv yields $uacq_kv$ if $\delta(q_j, b) = (q_k, c, R)$.

Turing machines: formal definition (3)

- Configuration C_1 yields C_2 if M can go from C_1 to C_2 in one step. So:
  - uaq_jbv yields uq_kacv if $\delta(q_j, b) = (q_k, c, L)$;
  - uaq_jbv yields $uacq_kv$ if $\delta(q_j, b) = (q_k, c, R)$.

- If M is at the left-hand end of the tape, it cannot move any further to the left; but M can move arbitrarily far to the right (the tape is one-way infinite).

Turing machines: formal definition (3)

- Configuration C_1 yields C_2 if M can go from C_1 to C_2 in one step. So:
  - uaq_jbv yields uq_kacv if $\delta(q_j, b) = (q_k, c, L)$;
  - uaq_jbv yields $uacq_kv$ if $\delta(q_j, b) = (q_k, c, R)$.
- If M is at the left-hand end of the tape, it cannot move any further to the left; but M can move arbitrarily far to the right (the tape is one-way infinite).
- M accepts input x if there is a sequence of configurations $C_1, \dots, C_k$ such that:
  1. C_1 is the start configuration of M on input x;
  2. For all $1 \le i \le k-1$, C_i yields C_{i+1};
  3. C_k is an accepting configuration (M is in state q_{accept}).

Real-world implementations

As well as being a mathematical tool, a Turing machine is a real machine that we can build...

- `http://aturingmachine.com/index.php`
- `http://www.legoturingmachine.org`

Summary and further reading

- A Turing machine is a generalisation of a finite automaton which has access to an infinite tape which it can read from and write to.
- Turing machines can perform complicated computations (although writing these down formally can be a painful process).
- Further reading: Sipser §3.1.