## 5CCS2FC2: Foundations of Computing II

# Automata and Turing Machines

**Dr Christopher Hampson**

*Department of Informatics*

*King's College London*

## Warm-up : What do the following programs do?

```
public static int prog1 (int k) {
    if (k == 1 || k == 2) { return 1; }
    else { return prog1(k-1) + prog1(k-2); }
}
```

```
public static int prog2 (int a, int b) {
    if (b == 0) { return a; }
    else { return prog2(b, (a%b)); }
}
```

```
public static boolean prog3 (int k) {
    if (k == 1) { return true; }
    if (k%2 == 0) { return prog3(k/2);}
    else { return prog3(3*k+1); }
}
```

- Do they ever get stuck in an **infinite loop**?
- How can you be sure?

## Who I am and where to find me?

### Dr Christopher Hampson

**Office:** Bush House (N)7.02 (North Wing)

**Office Hours:** Monday 2pm – 4pm

**Email:** christopher.hampson@kcl.ac.uk

---

Panos

Federico

Miznah

Yantong

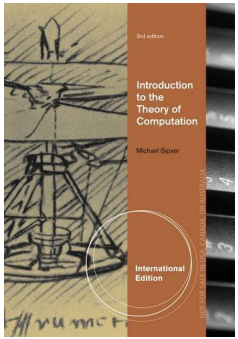
## Course Syllabus

- Finite Automata and Turing Machines
- Decidability and Undecidability
  - The Halting Problem
  - Mapping Reductions
- Complexity and Algorithms
  (why some problems are harder than others?)
  - Complexity Classes: P vs NP
  - NP-completeness,
  - Polynomial reductions
- Algorithms on Graphs,
  - Traversal algorithms (DFS and BFS)
  - Shortest path algorithms

## Course Syllabus

- Recursive algorithms
  - Divide-and-Conquer technique
  - Solving recurrence relations
- Efficient SAT solving
  - The DPLL Algorithm
  - Solving 2SAT
- Linear Programming
  - The Simplex Method
  - Integer Progamming
  - Branch-and-Bound technique

## Recommended Reading




### Introduction to the Theory of Computation

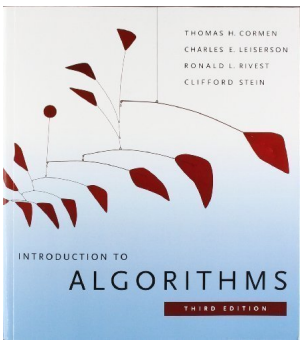
Michael Sipser

Cengage Learning, 3rd Edition, (2013)

### Introduction to Algorithms

Thomas H. Cormen, *et al.*

The MIT Press, 3rd Edition, (2009)




**Both books are widely available in the Maughan Library and online.**

## Assessment & Feedback

- **Assessment**
  - **Final Exam** **100%** of your final grade for this module.
    - Will consist of bookwork and exercises similar to those covered in the Tutorials,
    - A **mock exam** wil be published on KEATS towards the end of the term.
- **Feedback**
  - **Small Group Tutorials (SGTs)** **(Starting Week 2)**
    - **Tutorial sheets** will be posted on KEATS,
    - You should attempt the problems on your own **before** your SGT for maxmial benefit!
  - **KEATS forum**

## Reminder from FC1:

# Finite Automata

# Languages

- **Alphabet / Words / Languages**
  - **Alphabet** $\Sigma = \{a_0, a_1, \ldots, a_k\}$

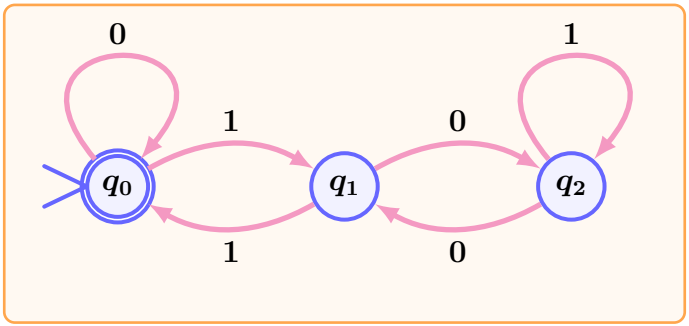
    A finite set of symbols. (*e.g.* $\{a, b, c, \ldots, z\}$ or $\{0, 1\}$ or $\{\square, \diamond\}$, *etc.*)

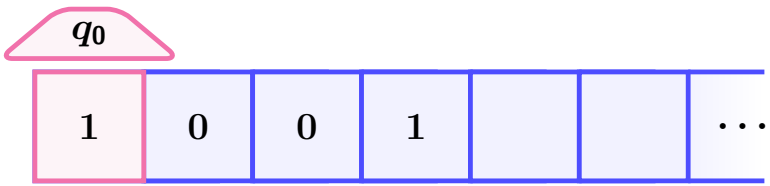
  - **Words / Strings** $\Sigma^* = \{$ all finite strings $\}$

    The set of all finite strings comprised of letters from the alphabet $\Sigma$.
    - **The empty string** $\varepsilon \in \Sigma^*$
  - **Languages** $L \subseteq \Sigma^*$

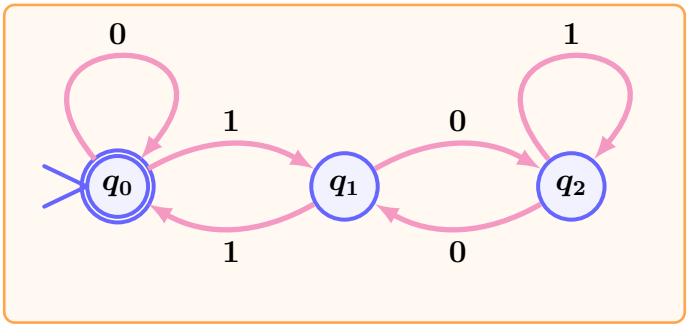
    *Any* subset (finite of infinite) of words is consider a language.
    - **The empty language,** $\emptyset \subseteq \Sigma^*$
    - **All English words,**
    - **All binary strings representing prime numbers, *etc.***

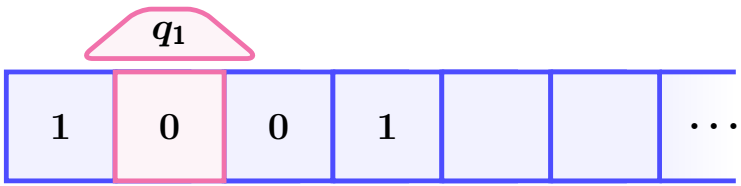
# Deterministic Finite Automata (DFA)



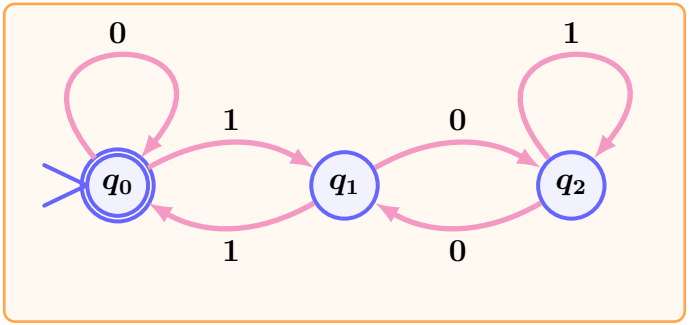



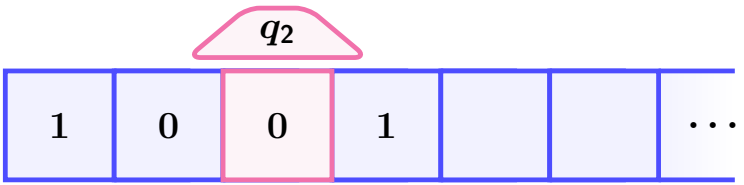
# Deterministic Finite Automata (DFA)



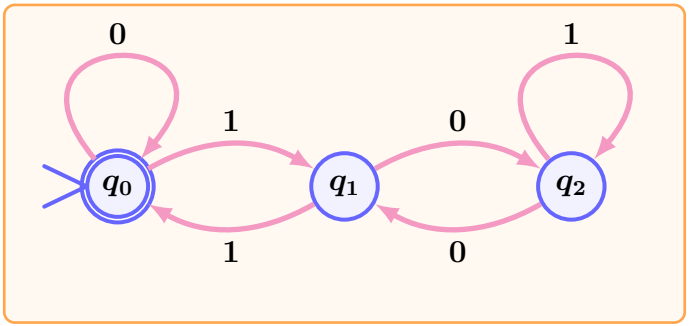



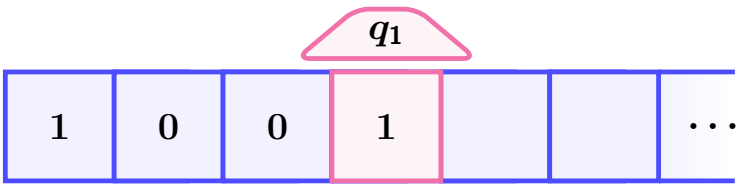
# Deterministic Finite Automata (DFA)



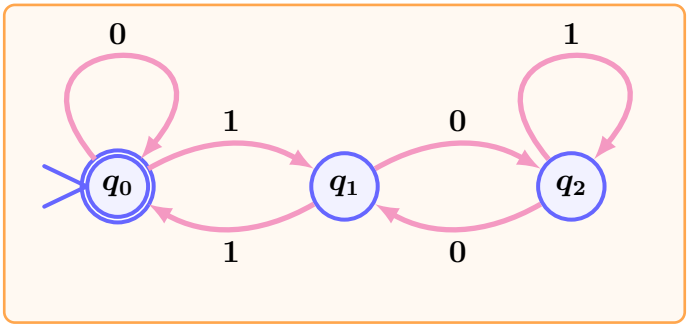



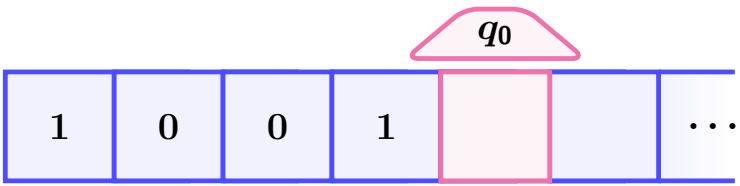
# Deterministic Finite Automata (DFA)



# Deterministic Finite Automata (DFA)



# Deterministic Finite Automata (DFA)

- **Deterministic Finite Automaton (DFA)** $\mathcal{A} = \langle\ \Sigma,\ Q,\ q_{\text{init}},\ F,\ \delta\ \rangle$
  - **Finite alphabet of symbols** $\Sigma = \{a_1, a_2, \ldots, a_k\}$
  - **Set of control states** $Q = \{q_0, q_1, \ldots, q_n\}$
  - **Initial state** $q_{\text{init}} \in Q$
  - **Final (accepting) states** $F \subseteq Q$
  - **Transition function** $\delta : Q \times \Sigma \to Q$

$\delta$( current state, read symbol ) $\mapsto$ new state

## Deterministic Finite Automata (DFA)

- **Transition Table:** We can describe a transition function with a table, listing all the automata transitions

| Current State | Read Symbol | New State |
|---|---|---|
| $q_0$ | 0 | $q_0$ |
| $q_0$ | 1 | $q_1$ |
| $q_1$ | 0 | $q_2$ |
| $q_1$ | 1 | $q_0$ |
| $q_2$ | 0 | $q_1$ |
| $q_2$ | 1 | $q_2$ |

**(Deterministic because every input pair appears only once in the table!)**

## Deterministic Finite Automata (DFA)

- **Configuration** A configuration is a pair listing the **current state** and the portion of the input tape to the **right hand side** of the tape head.

  (current state, right substring)

- **Computation / Run** The sequence of configurations obtained by running the DFA on a given input word, from the initial state.
  - **Example:**

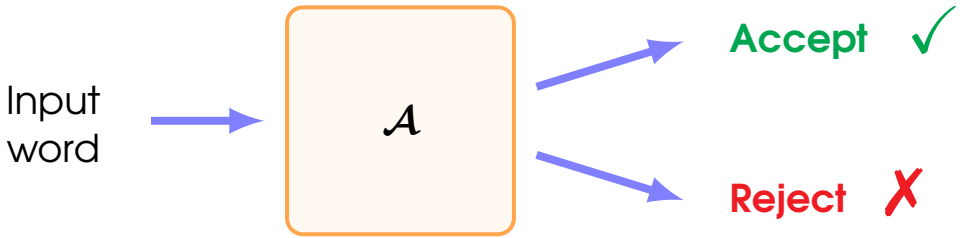
    $$(q_0, 1001) \rightarrow (q_1, 001) \rightarrow (q_2, 01) \rightarrow (q_1, 1) \rightarrow (q_0, \varepsilon)$$

    **(This is the computation illustrated in the above example.)**

- **Accept Criterion** A word $w$ is **accepted** if the final state of the (unique) computation is an Accepting state.

## Deterministic Finite Automata (DFA)

- **Decision Procedure** An automata can be thought of as a *'decision procedure'* that sorts words into two piles: accept and reject.




- **The language accepted by $\mathcal{A}$**

$$L(\mathcal{A}) = \{w \in \Sigma^* : \mathcal{A} \text{ accepts } w\}$$

# Non-deterministic Finite Automata (NFA)

- **Non-deterministic Finite Automaton (NFA)** $\mathcal{A} = \langle\ \Sigma,\ Q,\ q_0,\ F,\ \delta\ \rangle$
  - **Finite alphabet of symbols** $\Sigma = \{a_1, a_2, \ldots, a_k\}$
  - **Set of control states** $Q = \{q_0, q_1, \ldots, q_n\}$
  - **Initial state** $q_0 \in Q$
  - **Final (accepting) states** $F \subseteq Q$

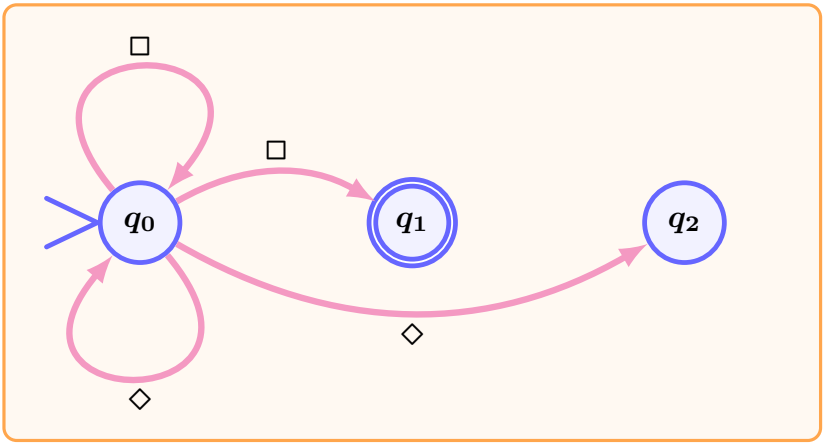
  } As before!

  - **Transition function** $\delta : Q \times \Sigma \to \mathcal{P}(Q)$

(where $\mathcal{P}(Q)$ denotes the powerset of $Q$)

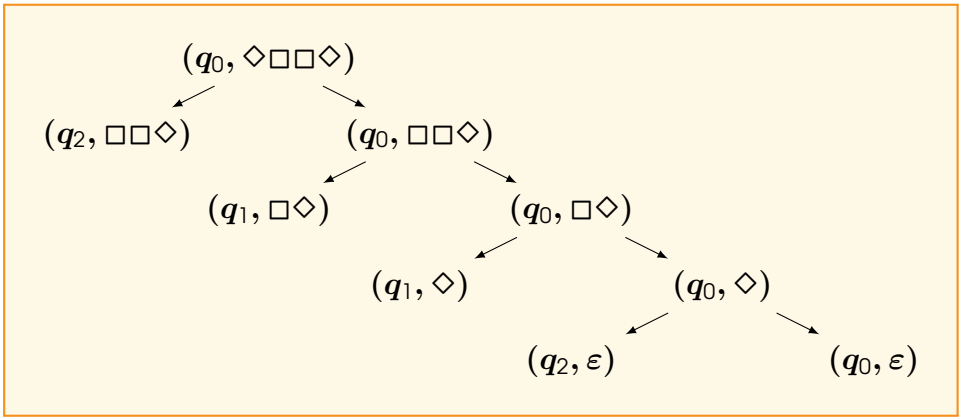
$\delta$( current state, read symbol ) $\mapsto$ { all possible new states }

# Non-deterministic Finite Automata (NFA)



## Non-deterministic Finite Automata (NFA)

- **Computation / Run** *Any* sequence of configurations obtained by running the NFA on a given input word, from the initial state.




**(Computations of an NFA form a branching tree structure)**

- **Accept Criterion** A word $w$ is **accepted** if there is **any** computation whose final state is an Accepting state.

## Regular Languages

- **Regular Expression** Any word that can be constructed using only the following rules:
  - **Concatenation** $wv$

    Take any two regular expressions and join them together sequentially,
  - **Alternation / Choice** $(w \cup v)$

    Make a *choice* between two alternatives $w$ and $v$
  - **Kleene Star** $w^*$

    Make zero or more *repetitions* of the regular expression $w$.

- **Regular Language** Any set of words that can be represented by a regular expression.

## Regular Languages

- **Some Examples:**
  - Language of all binary strings that start with a 1 and end with a zero,

$$1(1 \cup 0)^*0$$

  - Language of strings, containing only $\square$ and $\diamond$, whose length is even,

$$((\square \cup \diamond)(\square \cup \diamond))^*$$

  - **Less Obviously:** Language of all binary strings representing multiples of three,

$$(0 \cup 1(01^*0)^*1)^*$$

**(This represents all words accepted by the DFA we saw earlier!)**

## Are all languages regular?

## Proof that $a^n b^n$ is not regular

**Theorem** The lanuage $L = \{a^n b^n : n = 0, 1, 2, 3, \dots\}$ is not regular.

**Proof:**

**Step 1)** Suppose, to the contrary that there is some DFA / NFA $\mathcal{A}$ such that

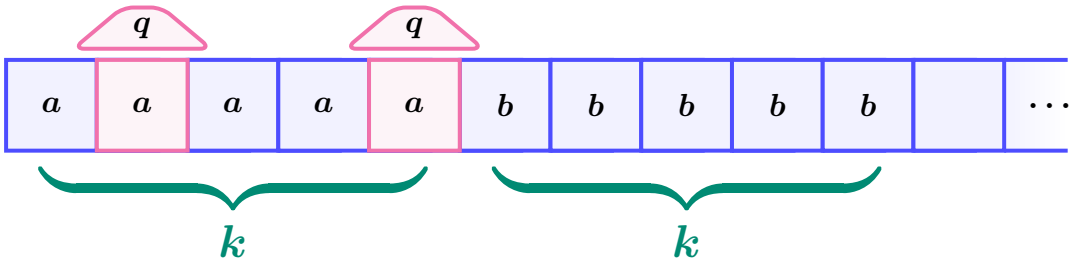
$$L(\mathcal{A}) = \{a^n b^n : n = 0, 1, 2, 3, \dots\}$$

**Step 2)** How many **different states** does $\mathcal{A}$ have. Let

$m$ = number of states in $\mathcal{A}$

## Proof that $a^n b^n$ is not regular

**Step 3)** Consider what the computation of $a^k b^k$ looks like for $k > m$.




$\mathcal{A}$ must **revisit** some state $q$ more than once!

**(By the Pigeon-hole Principle)**

**Step 4)** Denote these two times as $t_1$ and $t_2$ respectively,

$t_1$ = time of first visit to $q$
$t_2$ = time of second visit to $q$

## Proof that $a^n b^n$ is not regular

**Step 5)** There is a accepting computation for $a^k b^k$ that looks like this

$$(q_{\text{init}},\ a^k b^k) \to \cdots \to (q,\ a^{(k-t_1)} b^k) \to \ldots$$
$$\ldots \to (q,\ a^{(k-t_2)} b^k) \to \cdots \to (q_{\text{accept}},\ \varepsilon)$$

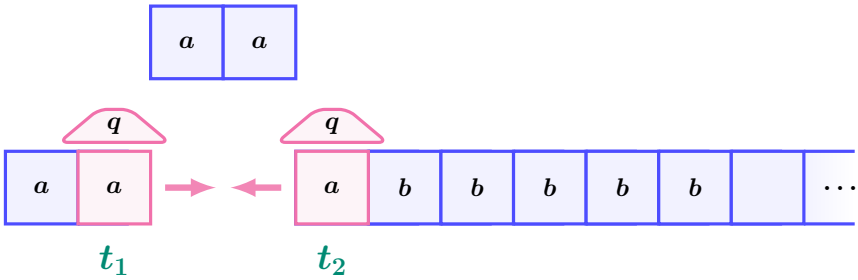
(where $q_{\text{accept}} \in F$ is an accepting state.)

**Step 6)** Note that:

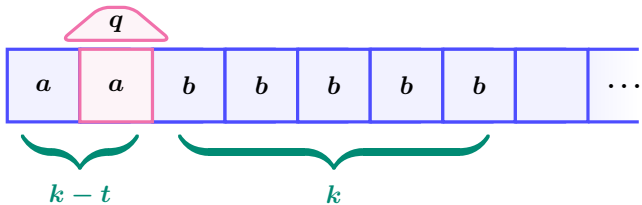
- We can reach $q$ from the $(q_{\text{init}}, a^{t_1})$,
- We can reach an accepting state from $(q, a^{k-t_2} b^k)$

## Proof that $a^n b^n$ is not regular

**Step 7)** By 'cutting out' the portion of the tape between the two revisits, and merging the two endpoints we have:




**Step 8)** As the **rightmost portion** of the tape has not been changed, the automata will **STILL** reach the accepting state



## Proof that $a^n b^n$ is not regular

**Step 9)** That is to say that there is an accepting computation for $a^{(k-t)}b^k$, where $t = (t_2 - t_1)$,

$$(q_{\text{init}},\ a^{k-t}b^k) \rightarrow \cdots \rightarrow (q,\ a^{(k-t_2)}b^k) \rightarrow \cdots \rightarrow (q_{\text{accept}},\ \varepsilon)$$

**Step 10)** Our proposed machine **incorrectly accepts** the string $a^{k-t}b^k$

**Conclusion:** Our automata does not work as advertised!

**(We do not have enough 'memory' to remember all the $a$'s we see!)**

**Step 11)** Hence, the language $a^n b^n$ is NOT regular!

**Q.E.D**

# Turing Machines

## Turing Machines

- **Finite Automata**
  - Can only **read** the tape,
  - Head can only move **right** along the tape,
  - Can be either **deterministic** or **non-deterministic**,
  - Can only read one tape cell at a time,

- **Turing Machine (TM)**
  - Can both **read** and **write** to the tape,
  - Head can move **left** or **right** along the tape, or **remain** where it is,
  - Can also be either **deterministic** or **non-deterministic**,
  - Can only read one tape cell at a time, **(Just like Finite Automata)**

## Turing Machines

- **Turing Machine (TM)** $\mathcal{T} = \langle\, \Sigma,\ Q,\ q_{\text{init}},\ q_{\text{accept}},\ \delta \,\rangle$
  - **Finite alphabet of symbols** $\Sigma = \{a_1, a_2, \ldots, a_k\}$
  - **Set of control states** $Q = \{q_0, q_1, \ldots, q_n\}$
  - **Initial state** $q_{\text{init}} \in Q$
  - **Halting accept state** $q_{\text{accept}} \in Q$

  **As before!**

  **(We only need one accepting state!)**

  - **Transition function** $\delta : (Q \times \Sigma) \to (Q \times \Sigma \times \{\leftarrow, -, \rightarrow\})$

$\delta$(current state, read symbol) $\mapsto$ (new state, write symbol, move)

**(TMs can also *write* symbols to the tape and *move* left or right or stay in place)**

- **Transition Table:** We can describe a transition function with a table, listing all the automata transitions
  - **Example:**

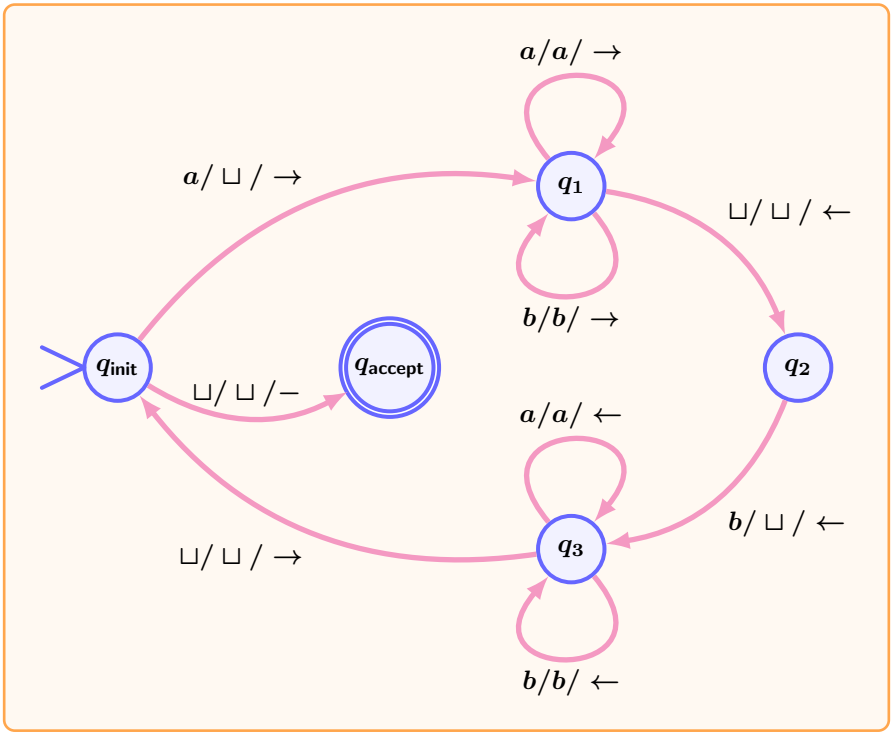
| Current State | Read Symbol | New State | Write Symbol | Move |
|---|---|---|---|---|
| $q_{\text{init}}$ | $a$ | $q_1$ | $\sqcup$ | $\rightarrow$ |
| $q_{\text{init}}$ | $\sqcup$ | $q_{\text{accept}}$ | $\sqcup$ | $\rightarrow$ |
| $q_1$ | $a$ | $q_1$ | $a$ | $\rightarrow$ |
| $q_1$ | $b$ | $q_1$ | $b$ | $\rightarrow$ |
| $q_1$ | $\sqcup$ | $q_2$ | $\sqcup$ | $\leftarrow$ |
| $q_2$ | $b$ | $q_3$ | $\sqcup$ | $\leftarrow$ |
| $q_3$ | $a$ | $q_3$ | $a$ | $\leftarrow$ |
| $q_3$ | $b$ | $q_3$ | $b$ | $\leftarrow$ |
| $q_3$ | $\sqcup$ | $q_{\text{init}}$ | $\sqcup$ | $\rightarrow$ |

**( $\sqcup \notin \Sigma$ is a special symbol used to denote a *blank* tape cell)**

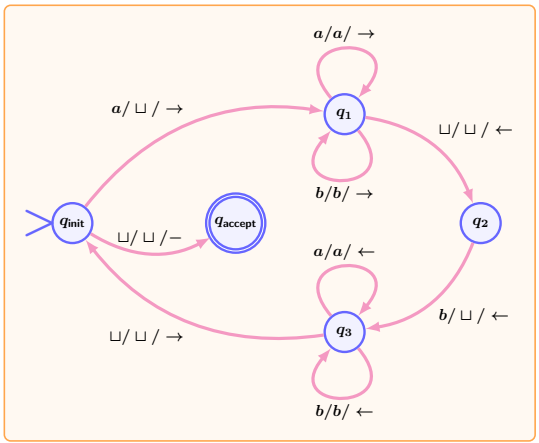
- **Transition Diagram:** Alternatively, we can *visualise* the transition function using a state transition diagram,

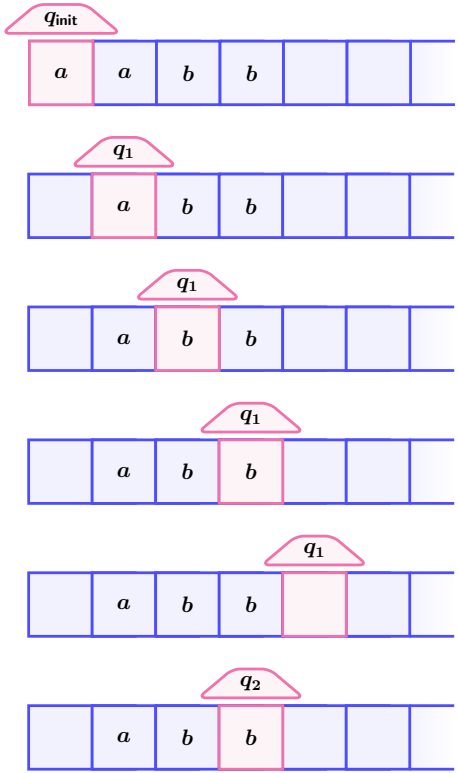
**(Each row of the transition table becomes a labelled edge from one state to another.)**

# Turing Machines

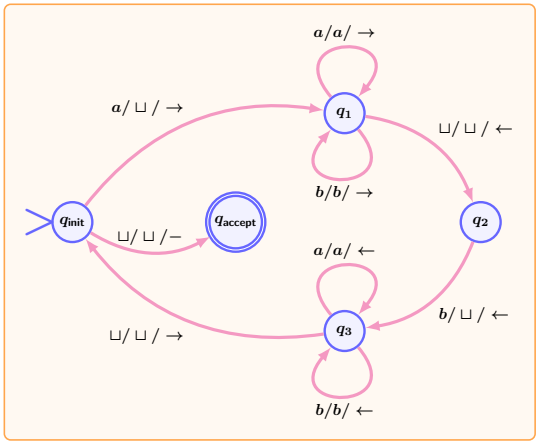


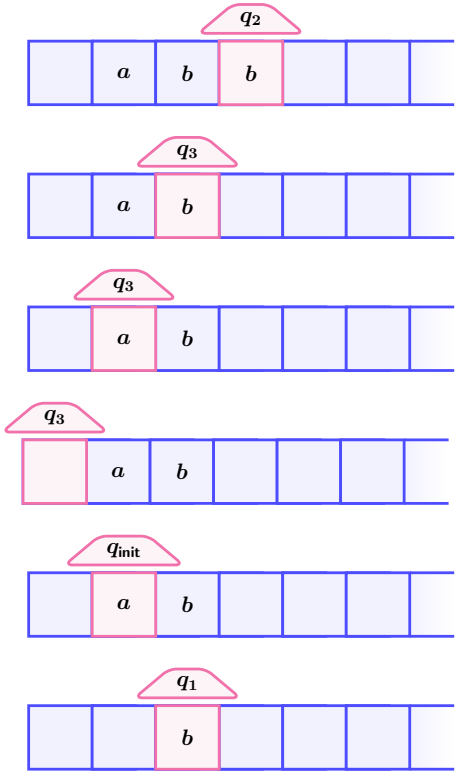
# Turing Machines



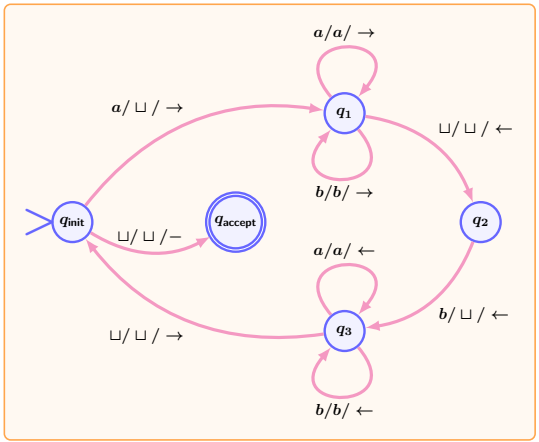



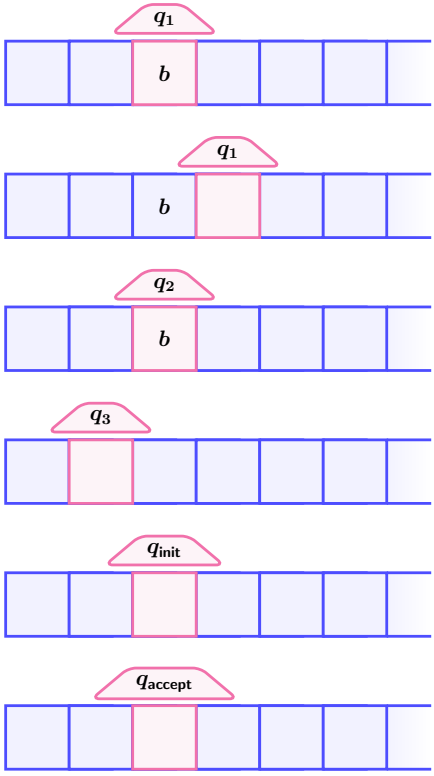
# Turing Machines



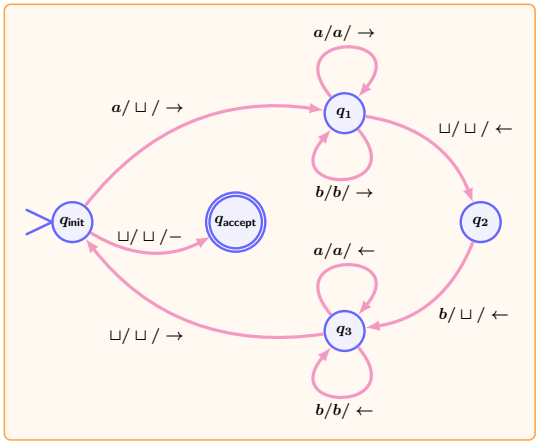
# Turing Machines

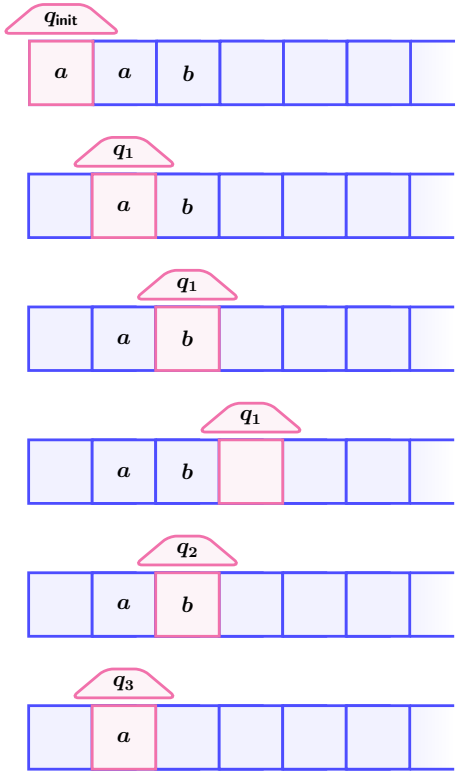


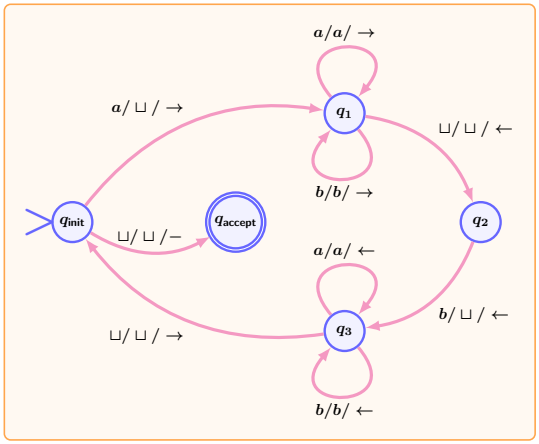

**Accept** ✓

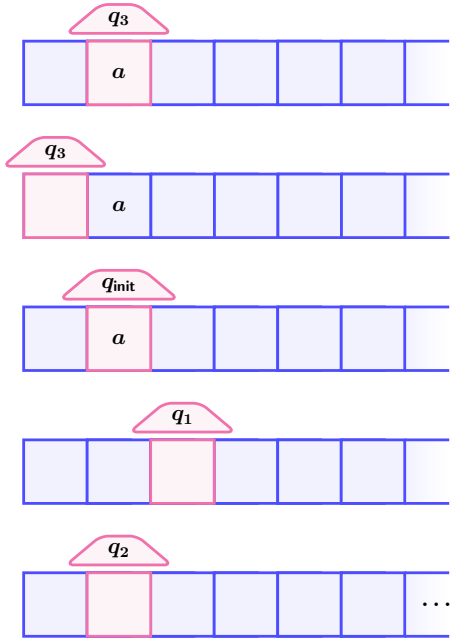
# Turing Machines



# Turing Machines




**Reject** ✗

## Turing Machines

- **Configuration** A configuration is a triplet listing the **current state**,the portion of the input tape to both the **left hand side** and **right hand side** of the tape head..

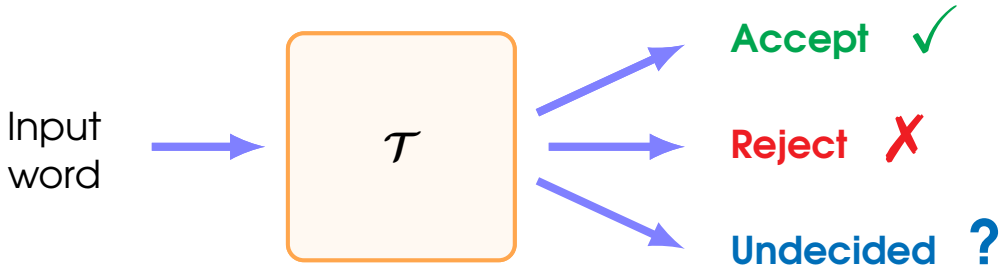
  (current state, left substring, right substring)

- **Computation / Run** The sequence of configurations obtained by running the TM on a given input word, from the initial state.
  - **Example:**

$$(q_{\text{init}}, \varepsilon, aabb) \rightarrow (q_1, \varepsilon, abb) \rightarrow (q_1, a, bb) \rightarrow (q_1, ab, b)$$
$$\rightarrow (q_1, abb, \varepsilon) \rightarrow (q_2, ab, b) \rightarrow (q_3, a, b) \rightarrow \ldots$$

- **Accept Criteria** A word $w$ is **accepted** if the machine terminates and the final state of the computation is $q_{\text{accept}}$.

- **Decision Procedure** Turing Machines can also be used as a decision procedure that sorts words into two piles: accept and reject.



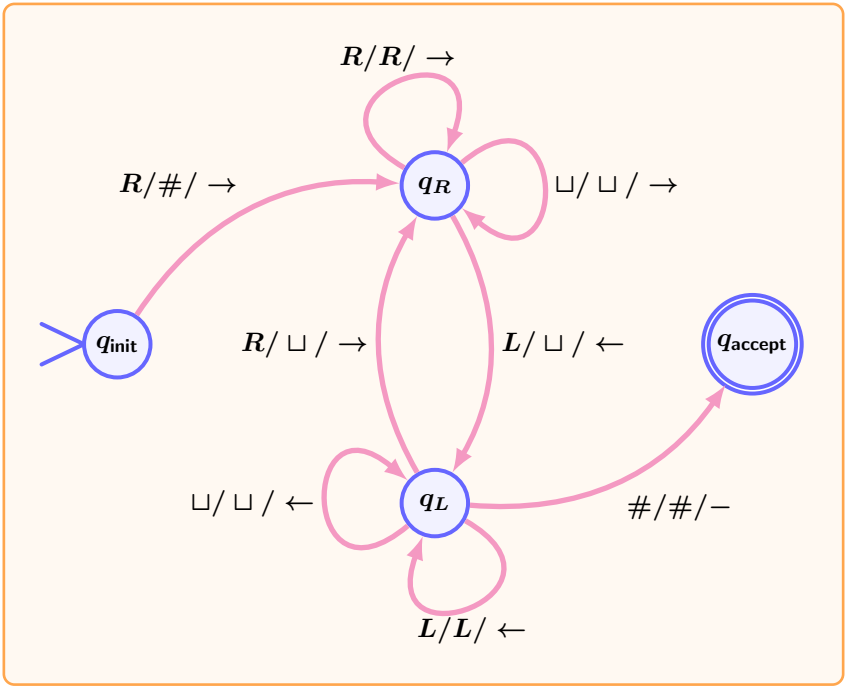

However, not all Turing Machines will **terminate** on every input!

Some words may never be sorted!

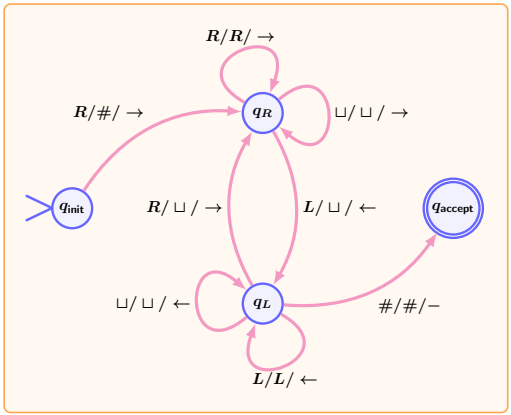
- **The language accepted by $\mathcal{T}$**

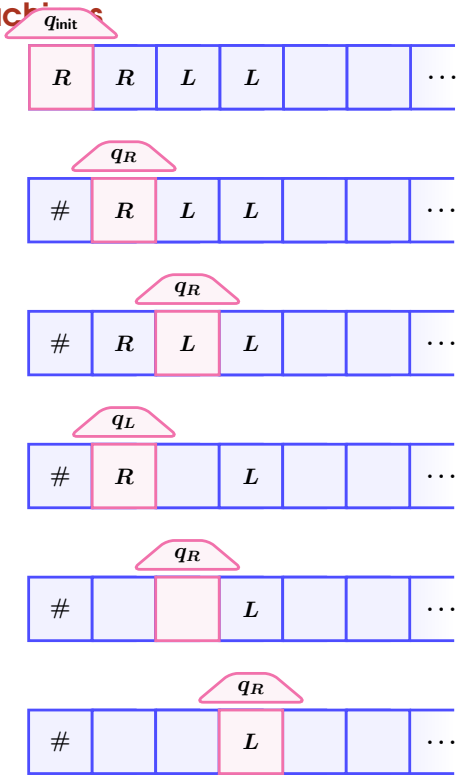
$$L(\mathcal{T}) = \{w \in \Sigma^* : \mathcal{T} \text{ accepts } w\}$$

# Turing Machines

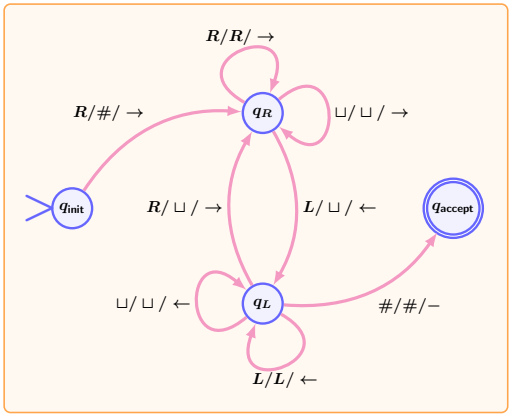


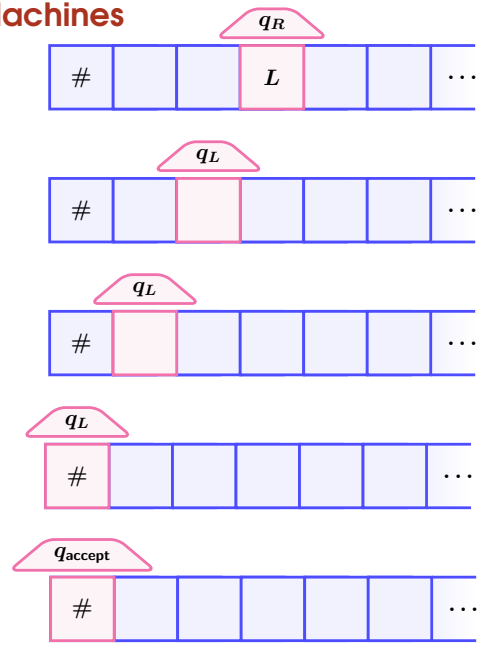
# Turing Machines



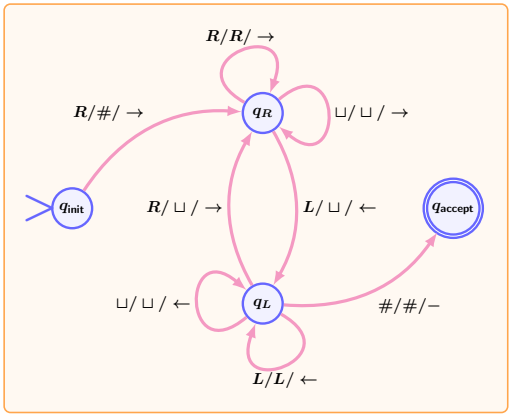



# Turing Machines



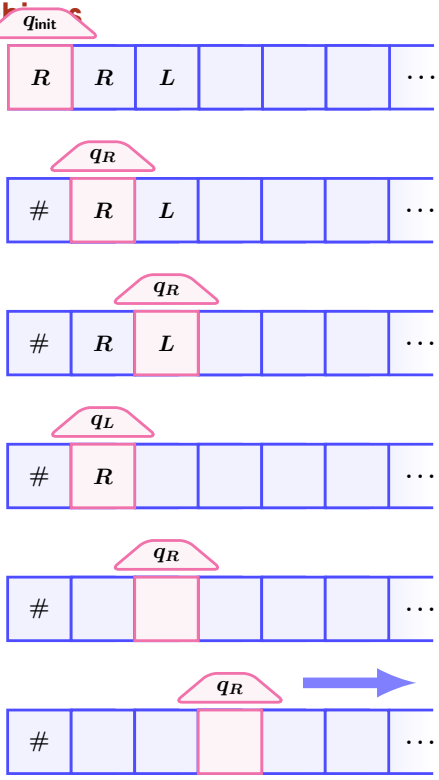




**Accept** ✓




# Undecided

## Does not terminate!



## Online Turing Machine Simulator

`https://turingmachinesimulator.com/`

## Non-deterministic Turing Machines

- **Non-deterministic Turing Machine** $\mathcal{A} = \langle\, \Sigma,\ Q,\ q_{\text{init}},\ q_{\text{accept}},\ \delta \,\rangle$
  - The **alphabet**, **set of states**, **Initial state**, and **final state** are as defined for deterministic Turing Machines,
  - **Non-deterministic transition function**

$$\delta : Q \times \Sigma \to \mathcal{P}\,(Q \times \Sigma \times \{\leftarrow, -, \rightarrow\})$$

$$\delta(\text{ current state, read symbol }) \mapsto \left\{ \begin{array}{c} \text{possible instructions} \\ \text{(state, write, move)} \end{array} \right\}$$

## Non-deterministic Turing Machines

- **Computation / Run** *Any* sequence of configurations obtained by running the TM on a given input word, from the initial state.

  **(Computations also form a branching tree structure)**

- **Accept Criterion** A word $w$ is **accepted** if there is **any** computation whose final state is an Accepting state.

A Non-deterministic Turing Machine behaves as if it has an arbitrary number of **parallel processors** — whenever there is a *choice* of possible transitions, each transition is followed on a new processor in parallel.

## Non-deterministic Turing Machines

**Theorem** Every Non-deterministic Turing Machine (NDTM) can be *simulated* on a Deterministic Turing Machine (DTM).

**(*i.e.* NDTMs cannot solve any additional problems that DTMs can't solve)**

**Proof (sketch):**

**Step 1)** Note the following processes that can be performed by a DTM:

- Remember the current state and head position by marking these on the tape,
- Clone the contents of the tape to another section of the tape, by writing out each symbol one at a time.

## Non-deterministic Turing Machines

**Step 2)** Given a NDTM $\mathcal{M}$, we can **construct** a DTM as follows:

**(i)** Apply all deterministic rules of $\mathcal{M}$ until there is a choice.

**(ii)** Mark the current state and head position and copy the contents of the tape somewhere else

**(iii)** Take turns to continue the computation on each copy separately,

**(iv)** Accept whenever a cloned computation accepts,

**(v)** Repeat from (i) unless all computation reject.

**Q.E.D**

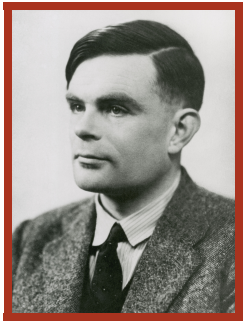
## What Languages can Turing Machines accept?

- All regular languages. **(Why?)**
- The language of all palindromes, **(*e.g.*, 101101101)**
- The language of all binary representations of all:
  - perfect squares,
  - prime numbers,
  - *etc.*
- The language of satisfiable formulas in propositional logic,
- Can simulate all Java programs!
- *etc.*

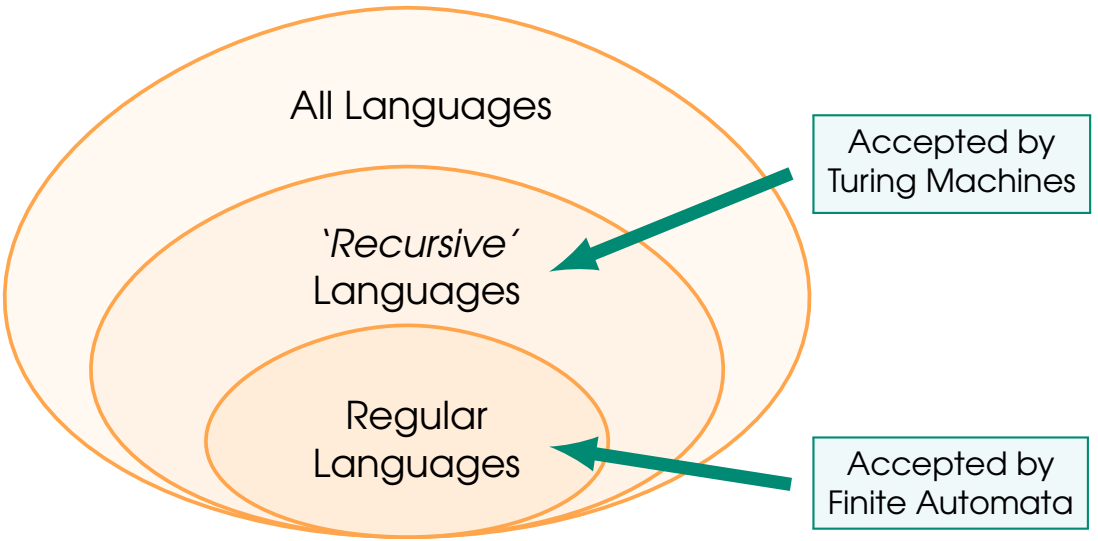
**Pretty much anything you can imagine...**

## What Languages can Turing Machines accept?

**Theorem (The Church-Turing Thesis)** Any language that can be recognised by an algorithm can be recognised by an appropriate Turing machine.



## What Languages can Turing Machines accept?




**More about this Next Week...**

# End of Slides!