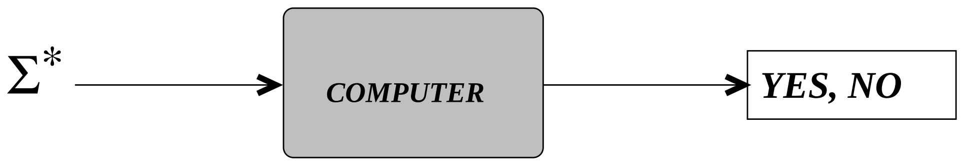
1 Computing

Given an alphabet $\Sigma = \{a_1, a_2, \ldots, a_n\}$, a **language** is a subset of the set Σ^* of all finite strings w over Σ.

Computing is **accepting** a language of Σ^* using a computational device, an **algorithm** or a **computer**.



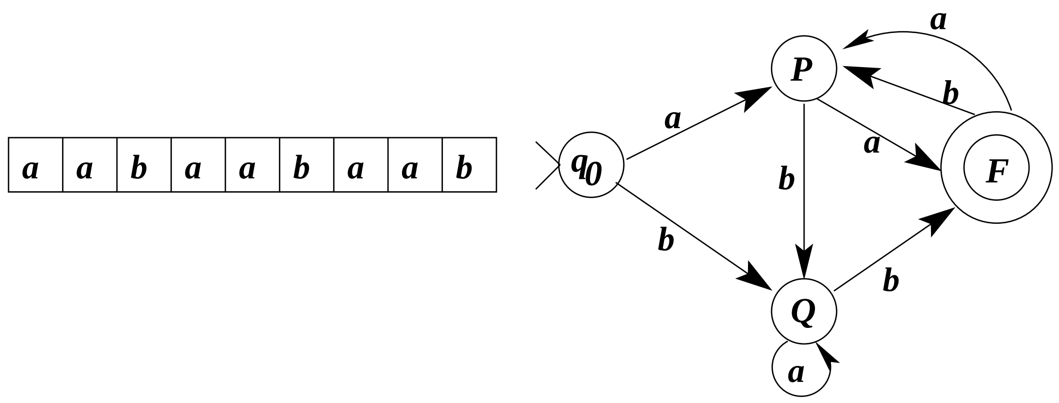
For every input w, the computer
either accepts (YES),
or rejects (NO),
or enters an infinite loop

Computer M accepts language L if
L is the set of all strings w with
M(w) = YES

2 Finite Automata

A **deterministic finite automaton** is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where

Q	a finite set of states
Σ	an alphabet
$q_0 \in Q$	the initial (start) state
$F \subseteq Q$	the set of final (accept) states
δ	the transition function $\delta : Q \times \Sigma \to Q$.



An automaton M accepts a string $w \in \Sigma^*$ if there is a path from q_0 to a final state labeled by the symbols of w.

The language L accepted by M is the set of all strings accepted by M.

$$\begin{tikzcd}
q_0 \arrow[r, "a"] & P \arrow[r, "b"] & Q \arrow[r, "b"] & F \arrow[r, "b"] & P \arrow[r, "a"] & F
\end{tikzcd}$$

3 Turing Machines.

An informal definition of an algorithm

a set of rules that precisely defines a sequence of operations

Church-Turing thesis:

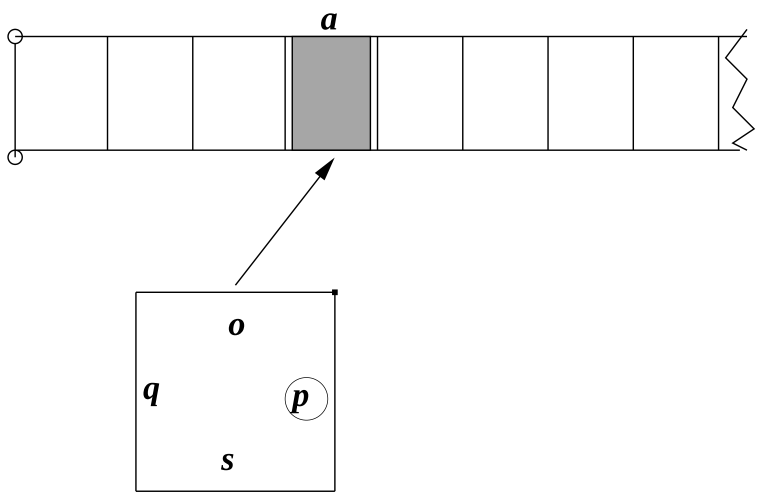
Turing machines are formal versions of algorithms; no computational procedure is an algorithm unless it can be presented as a Turing machine.

A **Turing machine** M is a 7-tuple

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, where

1. Q is a finite set of **states**, including q_0, q_a, q_r;
2. Σ is **finite input** alphabet not including the **blank** symbol $\sqcup$;
3. Γ is the **tape alphabet**, containing Σ and $\sqcup$;
4. δ is the **transition** function defined on $Q \times \Gamma$;
$$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\};$$
5. $q_0 \in Q$ is the initial (start) state;
6. $q_a \in Q$ is the accept state;
7. $q_r \in Q$ is the reject state; $q_a \neq q_r$.

A Turing machine is visualized as a finite state machine, **control**, with an infinite **tape,** and a **tape-head**.



The tape is subdivided into sections called **tape squares**; the contents of every square is a symbol from the tape-alphabet.

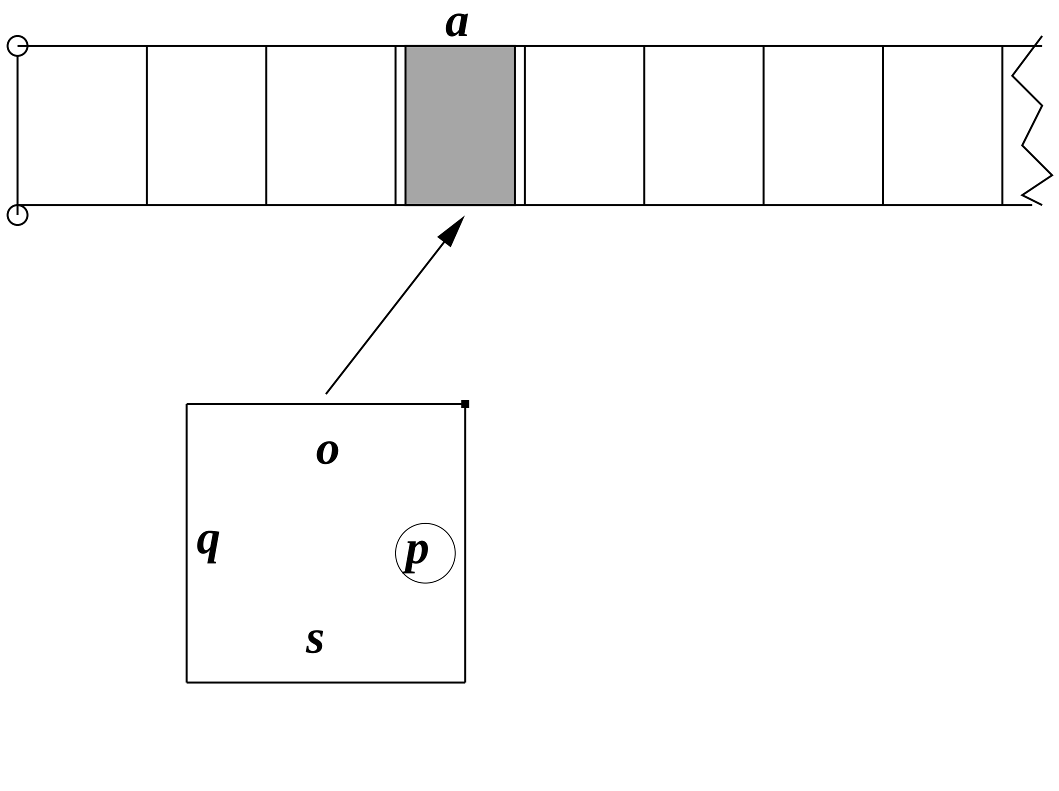
At every moment, only a finite number of tape squares has a non-blank symbols; all other squares contain the blank symbol $\sqcup$.

The tape-head can read from and write into tape squares; it can move left and right along the tape, one square at a time, according to the directional cursors.

The **current** string of M is the string composed of the symbols inside the squares that were ever visited by the head. A TM **operates** as follows:

if (1) the machine is in a state q;
  (2) the head scans a tape square containing σ; and
  (3) $\delta(q, \sigma) = (p, \gamma, D)$, then

- the machine replaces the symbol σ with γ;
- changes the state of the control to the state p;
- moves the head according to the directional cursor.

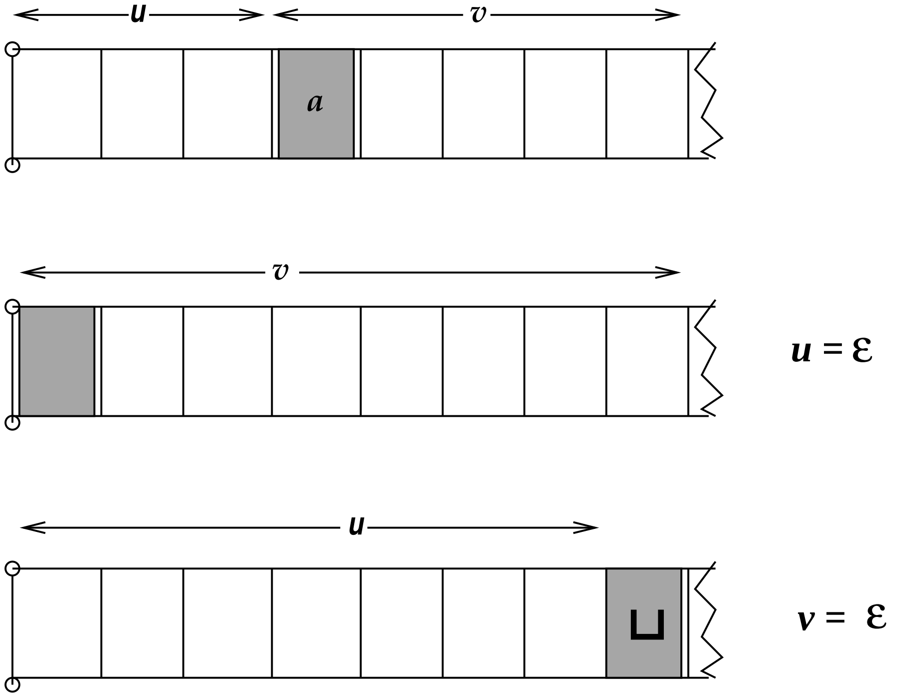


A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ **computes** as follows:

- receives an input $w = \sigma_1\sigma_2 \ldots \sigma_n \in \Sigma^*$ placed into the n squares of the tape starting with the leftmost square; the rest of the tape is filled with blank symbols;
- the head starts at the leftmost square of the tape; the first blank appearing on the tape marks the end of the input;
- once M starts, the computation proceeds according to the rules described by the transition function; if the head of M is scanning the leftmost square and M tries to move it left, then it stays at the same square even though the direction is to go left;
- the computation continues until it enters one of the halting states q_a or q_r, at which point it halts; if neither occurs, M goes on **forever.**

A **configuration** of a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ is a 3-tuple $u\ q\ v$, where

- uv is the current tape contents (*not including the infinite string of blanks*);
- $q \in Q$ is the current state of M;
- the current head location is the leftmost symbol of v;



A configuration

$$C = u\ q\ v$$

is said to **yield** configuration

$$C' = u'\ q'\ v',$$

written $C \to C'$, if the Turing machine can legally go from C to C' in a single step according to the transition function.

Current	Transition	Next
$ua\ q\ \sqcup$	$\delta(q, \sqcup) = (p, c, R)$	$uac\ p\ \sqcup$

The **start configuration** of M on input w is the configuration

$$q_0\ w.$$

The **accepting** (resp. **rejecting**) configuration of M is

$$u\ q_a\ v\ \ \text{(resp. }\ u\ q_r\ v\text{)}.$$

Accepting and rejecting configurations are called **halting** configurations.

4 Turing machines as algorithms.

Definition 1 *A* **string** $w \in \Sigma^*$ *is* **accepted** *(resp.* **rejected***) by a Turing machine M if $\exists$ 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. *$\forall i \in [1, k-1]$, C_i yields C_{i+1};*
3. *C_k is an accepting (resp. rejecting) configuration.*

Definition 2 *Let $L \subseteq \Sigma^*$. L is accepted by Turing machine M, written $L = L(M)$, if L is the collection of all strings that are accepted by M.*

If $L = L(M)$ for some Turing machine M, then L is **Turing-acceptable/recognizable**, (**recursively-enumerable**).
Notice that, for some or all $w \notin L(M)$, it can be that M is **not rejecting** w.

Definition 3 *A language $L \in \Sigma^*$, is called* **Turing-decidable**, *(***recursive***), if $\exists M$ s. t. $\forall w \in L$, M accepts w, and for every $w \notin L$, M rejects w. Such machine M is called a* **decider** *for L.*

Proposition 1 *If L is recursive, then it is recursively enumerable.*

Definition 4 *Let $f : \Sigma^* \to \Sigma^*$. A Turing machine M with alphabet Σ is said to* **compute** *f, if for any $x \in \Sigma^*$, $f(x)$ is the contents of the tape after M halts (not including the infinite string of blanks).*

5 String representations of finite objects.

Any finite object can be represented as a string in a finite (binary) alphabet.

All reasonable representations are "polynomially equivalent": if A and B are two representations, then there is a polynomial p such that if n is the length of A-representation of an object, then the length of the B-representation is at most $p(n)$.

An example of a **reasonable** and **unreasonable** representation:

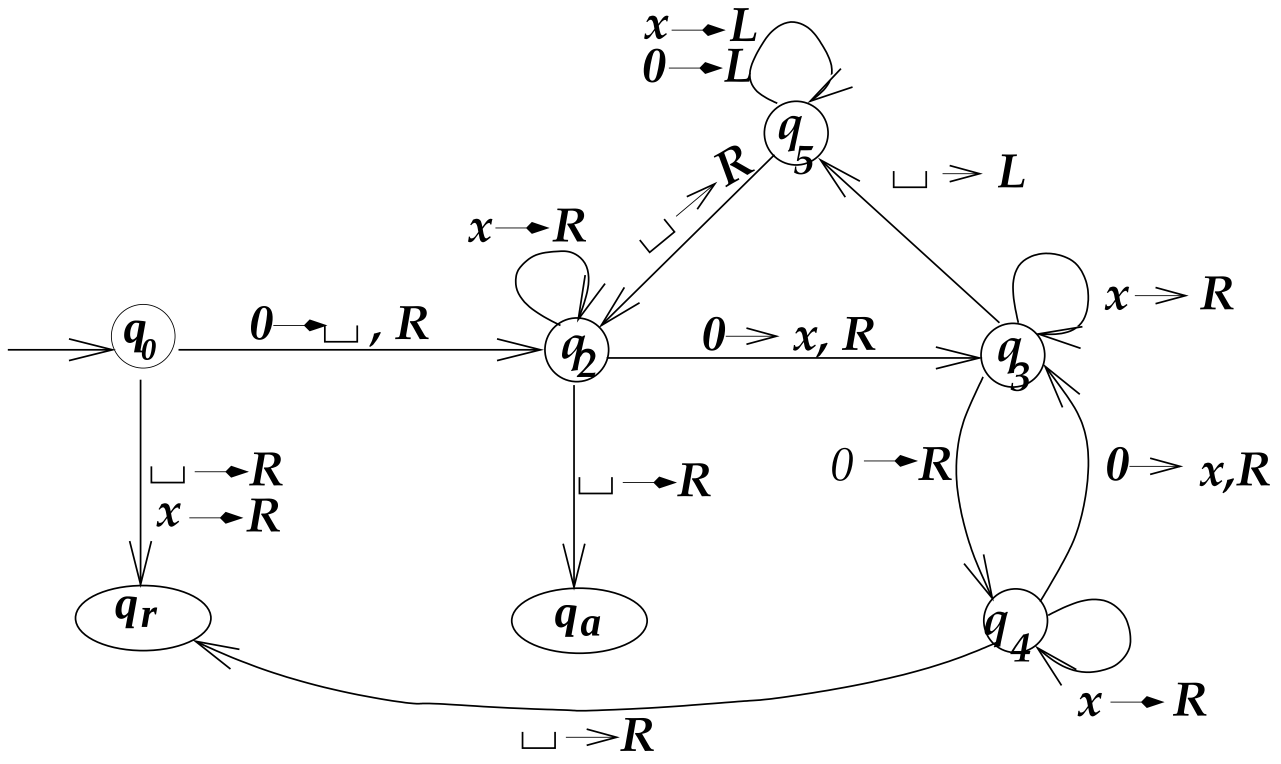
100 **binary** representation of integer 8

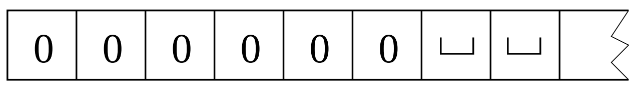
11111111 **unary** representation of integer 8

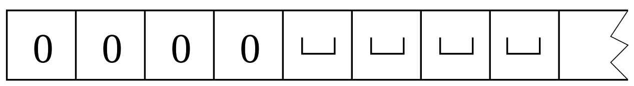
6 Examples of Turing Machines

Example 1 (a TM to compute a power of 2) *Describe a TM $M = (K, \Sigma, \delta, s)$ that decides the language consisting of all strings of* 0*s whose length is a power of 2:* $L = \{0^{2^n} : \ n \geq 0\}$.

Solution. The TM is described by the following state diagram:







Notations. In the state diagram a label appearing on the transition from state p to state q of the form $a \to b, R$ represents the transition $\delta(p, a) = (q, b, \to)$; similarly for labels $a \to b, L$.

A label of the form $a \to R$ is a shortcut for $\delta(p, a) = (q, a, \to)$. Symbol x is an auxiliary symbol of our alphabet; this symbol is used to "cross out" a 0 during the computation.

The input string is always a string of 0s; the TM accepts those inputs whose length is 2^n for a non-negative integer n.

Here is a **high-level description** of the Turing machine.

1. Sweep left to right across the tape, replacing the first 0 with $\sqcup$ and every other 0 with x (*crossing it off*).
2. If in step 1 the tape contains a single 0, *accept*.
3. If in step 1 the tape contains more than a single 0 and the number of 0s was odd, *reject*.
4. Return the head to the leftmost end of the tape.
5. Go to step 1.

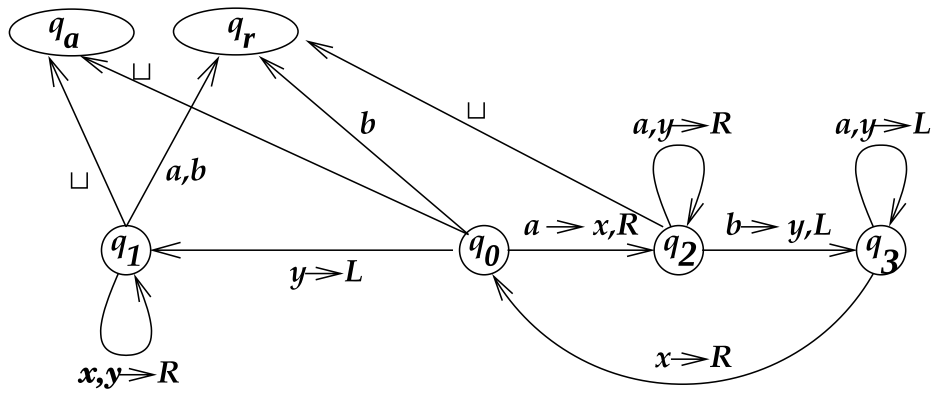
Each iteration of stage 1 cuts the number of 0s in half. As the machine sweep across the tape, it keeps track of whether the number of 0s seen is even or odd.

If the number of 0s is odd and greater than 1, the original number of 0s in the input could not have been a power of 2, therefore the machine rejects. If the number of 0s seen is 1, the original number must have been a power of 2, therefore in this case the machine accepts.

In order to find the the left-hand end of the tape in step 4, the machine starts by writing a blank symbol over the leftmost 0 on the tape.

Example 2 *Describe a TM accepting* $L = \{a^n b^n \mid n \geq 0\}$.

Solution. This Turing Machine is described by the following diagram:



It works as follows:

1. If input is empty (resp. starts with a b), `accept` (resp. `reject`).
2. Sweep left to right replacing one a with x and then one b with y.
3. Move left till the first x, after which move one step right.
4. Go to #2, if the scanned symbol is a; if the symbol is b, `reject`; if the symbol is y, pass all y till the first symbol different from y. If it is a or b, `reject`; if it is the blank symbol, `accept.`

Example 3 *Describe a TM deciding language*

$$L = \{w \in \Sigma \mid |w_a| = |w_b| = |w_c|\}.$$

Solution. The high-level description of the machine.

1. Sweep left to right across the tape, replacing exactly one a, one b and one c, occurred in any order, by #.
2. If after step 1 the tape contains only # symbols, accept.
3. If in step 1 TM replaced one or two of the symbols a, b, c, but didn't find the others to be replaced, reject.
4. return the head to the left-hand end of the tape.
5. Go to step 1.

Give a detailed description (such as formal description, or state diagram for this Turing machine).

Show the sequence of configurations that accept *abaabbccc*. Show why the TM does not accept *aabbc*.