Computability Theory

The Birth of Turing Machines

At the end of the 19th century, Gottlob Frege conjectured that mathematics could be built from fundamental logic.

In 1900 David Hilbert, accepting this view, asked whether there is *a process whereby it could be determined in a finite number of operations* whether a given polynomial has an integral root.

In 1931 Kurt Gödel showed that every logical system general enough to deal with arithmetic contains statements that cannot be proven true or false.

What classes of mathematical problems lack effective solutions?

The Birth (cont'd)

In 1936 several papers appeared that shed light on that question: Church's **Lambda calculus**, Kleene's **(general) recursive functions**, Turing's system, now called **Turing machines**, and Post's production systems, similar to Turing machines.

All these systems were proved equivalent. Church and Turing then proposed equivalent theses, viz., **"Church's Thesis,"**, **"Turing's Thesis,"**, or the **"Church-Turing Thesis,"** which states that the classes of mathematical problems that are effectively solvable in the intuitive sense are those that are effectively solvable in one of (and thus, any of) the above computational models.

The thesis also suggests that the notion of effective computability is model-independent.

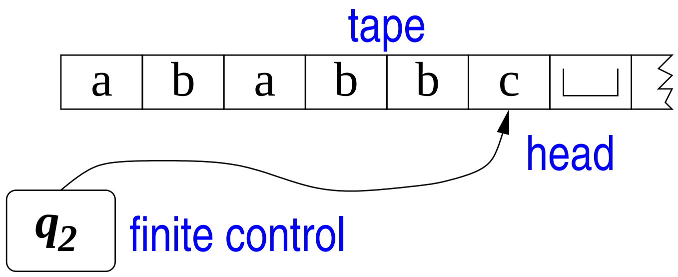
Turing Machines

A **Turing machine** is a step-wise computing device, which consists of

1. an infinitely long **tape** that is divided into **tape squares** (or **tape cells**),
2. a **head** that scans (is positioned at) a particular tape square at each time step, and
3. a **finite control** that maintains the current state.

Turing Machines (cont'd)

Each combination of **a state and a scanned tape symbol** determines **the next state, the symbol written on the scanned square, and the move (L or R)** of the tape head.



Turing Machines (cont'd)

A **Turing machine** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are finite sets,

1. Q is a set of states,
2. Σ is the input alphabet,
3. Γ is the tape alphabet, which includes $\sqcup$ (the special **blank** symbol), and where $\Sigma \subseteq \Gamma - \{\sqcup\}$,
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function,
5. q_0 is the initial state,
6. q_{accept} is the accept state, and
7. q_{reject} is the reject state

Computation of a Turing Machine

In one step, a Turing machine:

1. **Reads the symbol written on the tape square on which the head is located** and, according to the transition function
2. **Makes one move**: It
   (a) **updates its state**,
   (b) **writes on the current tape square**, and
   (c) **moves the head by one square either to the left or to the right**,

   where the head stays at the same position if it is **at the left end** of the tape and **a left move is specified by** δ

Computation of a Turing Machine (cont'd)

At the beginning

1. the tape contains the input on the leftmost squares,
2. the rest of the tape is filled with blank symbols,
3. the head is at the leftmost cell, and
4. the state is q_0.

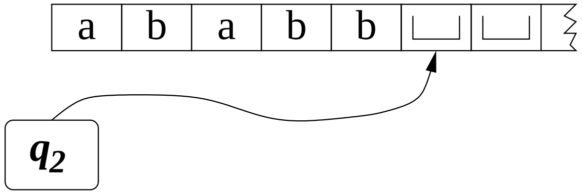
The Turing machine **halts** when it enters one of q_{accept} and q_{reject}, and in these states it **accepts** and **rejects**, respectively. (We could make this more explicit by defining the domain of δ to be $(Q - \{q_{\mathrm{accept}}, q_{\mathrm{reject}}\}) \times \Gamma$.)

Configurations

A **configuration** of a Turing machine consists of the contents of its tape, the head position, and the state. If the tape contents are $a_1, \ldots, a_m \sqcup \ldots$, the head is located on the kth square, and the state is q, then we write

$$a_1 \ldots a_{k-1} q a_k \ldots a_m$$

to denote the configuration. The segment of blanks filling the tape is omitted but at least one symbol (possibly $\sqcup$) is required after the state. Thus a configuration is a word in $\Gamma^* Q \Gamma (\Gamma - \{\sqcup\})^*$.



In the above, the configuration is $\mathtt{ababb}\, q_2 \sqcup$.

Configurations (cont'd)

The action of a TM can be viewed as rewriting of the configuration.

A configuration C_1 **yields** C_2 if the Turing machine can go from C_1 to C_2 in a single step. Let $a, b, c \in \Gamma$ and $u, v \in \Gamma^*$:

- uaq_ibv yields uq_jacv if $\delta(q_i, b) = (q_j, c, L)$.
- q_ibv yields q_jcv if $\delta(q_i, b) = (q_j, c, L)$.
- uq_ibv yields ucq_jv' if $\delta(q_i, b) = (q_j, c, R)$, where v' is v if $|v| > 0$ and $\sqcup$ if $|v| = 0$.

An accepting configuration (**A rejecting configuration**) is one in which the state is q_{accept} (q_{reject}). Accepting configuration and rejecting configurations are **halting configurations**.

The Languages of Turing Machines

A Turing machine M **accepts** (**rejects**) a string x if it eventually enters an accepting (a rejecting) state for input x.

A Turing machine M **recognizes** a language L, if for every input x, M **on** x **accepts if and only if** $x \in L$.

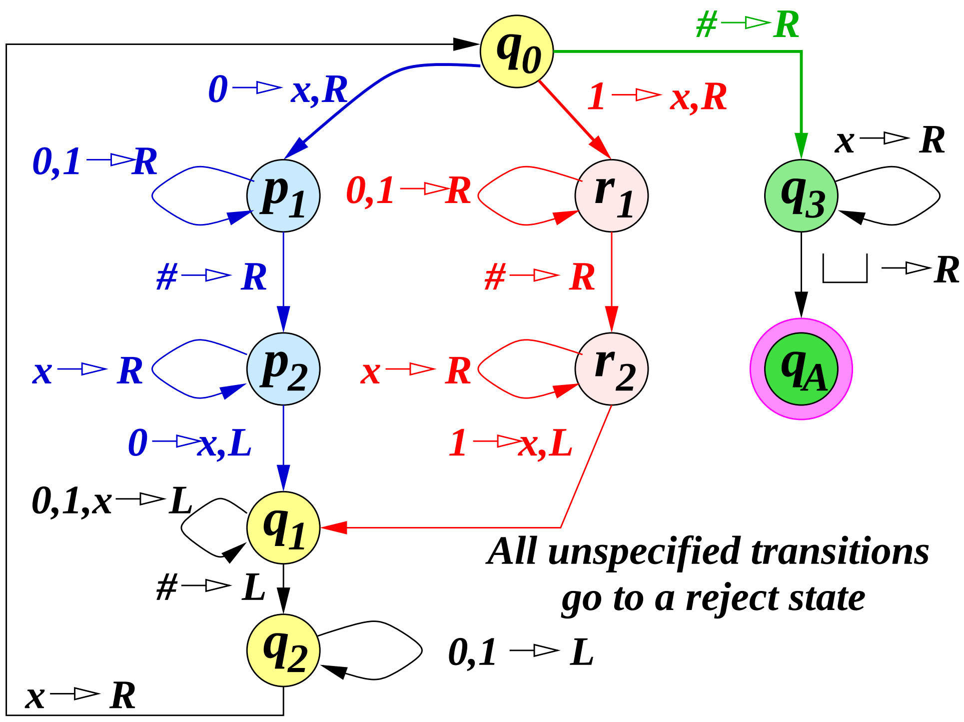
A Turing machine M **decides** a language L, if M recognizes L and halts on all inputs.

A language is **Turing-recognizable** (or **recursively enumerable**) if there is a Turing machine that recognizes it.

A language is **Turing-decidable** (or simply **decidable**) if there is a Turing machine that decides it.

Example

A TM for recognizing $\{w\#w \mid w \in \{0,1\}^*\}$



Multitape TMs & Nondeterministic TMs

A **multitape Turing machine** is a Turing machine with additional tapes where each tape is accessible individually, with the input on the first tape, and with the others blank at the beginning.

For a k-tape Turing machine, the transition function δ is a mapping from $Q \times \Gamma^k$ to $Q \times \Gamma^k \times \{L, R\}^k$.

A **nondeterministic Turing machine** is one in which the transition is mapping to the power set of $Q \times \Gamma \times \{L, R\}$.

A nondeterministic Turing machine **accepts** an input if it enters an accepting state **for some computation path.**

Equivalence Between Single Tape TMs and Multitape TMs

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

Proof From a k-tape TM M build a single tape simulator S.

For each $a \in \Gamma$, let $\widetilde{a}$ be a new symbol to signify that **a head is located on the symbol.** Then S **concatenates the contents of M's tapes**, with a new symbol $\#$ as a **delimiter.** The tape configuration of M in which the tape contents are $u_1, \ldots, u_m, \ldots, v_1, \ldots, v_n$ and the head positions are $s, \ldots, t$ is encoded as:

$$\#u_1, \ldots, u_{s-1}\widetilde{u_s}u_{s+1}, \ldots, u_m\# \cdots \#v_1, \ldots, v_{t-1}\widetilde{v_t}v_{t+1}, \ldots, v_n\#$$

S memorizes M's state using its own state.

Proof (cont'd)

On input $w = w_1 \ldots w_n$:

1. Convert w into its initial form:

$$\#\widetilde{w_1}w_2, \ldots, w_n\# \cdots \#\tilde{\sqcup}\#\tilde{\sqcup}\# \cdots \#\tilde{\sqcup}\#.$$

2. While M has not halted, repeat:
   (a) Make a sweeping scan on the tape to **find the symbols** scanned by the heads of M.
   (b) **Determine the next move** of M and **modify the encoding accordingly.** Insert symbols if necessary.
3. Accept or reject accordingly.

Equivalence Between NTMs and TMs

Theorem. *Every nondeterministic Turing machine has an equivalent deterministic Turing machine.*

Proof From an NTM N construct a three-tape simulator D. Let b be a constant such that each transition has at most b possible values. Let $\Sigma_b = \{1, 2, \ldots, b\}$. Use a word in Σ_b^* on Tape 3 to **encode a nondeterministic path,** where for each $i \geq 0$, if the ith symbol of the word is j, this signifies that for the ith move of N, the jth element among the available choices (if any) is selected.

The word is **invalid** if too few choices are available.

Proof (cont'd)

Use Tape 1 to keep the input, Tape 2 to simulate the tape of N, and Tape 3 to keep an encoding of a computation path.

Define the lexicographic order of paths:
$u_1, \ldots, u_s < v_1, \ldots, v_t \in (\Sigma_b)^*$ if and only if either $s < t$ or $s = t$ and there exists some k, $1 \leq k \leq s$, such that $u_1 = v_1, \ldots, u_{k-1} = v_{k-1}$, and $u_k < v_k$.

E.g., for $b = 3$, the possible computation paths in lexicographic order are

$1, 2, 3, 11, 12, 13, 21, 22, 23, 31, 32, 33, 111, 112, \ldots$

An algorithm for N

On input w, write the word $\#1$ on Tape 3, then repeat:

1. **Copy the input onto Tape 2.**
2. **Try to simulate N on w using the word on Tape 3** as the path. If successful and if N has accepted, then accept and halt.
3. **Modify the path to the next smallest path by incrementing it.**
4. **Erase Tape 2.**

■

Corollary. *A language is Turing-recognizable if and only if it is recognized by a multitape TM.*

Corollary. *A language is Turing-recognizable if and only if it is recognized by an NTM.*

Enumerators

An **enumerator** of a language A is a TM with a special **output tape** such that the machine writes on the output tape all the members of A with a special symbol $\#$ as a delimiter.

Theorem. *A language is Turing-recognizable if and only if it has an enumerator.*

Proof The "if" part: Simulate the enumerator, and accept if the input has been output.

The "only if" part: Simulate a recognizer R. For $i = 1, 2, \ldots$, for each w of lexicographic order of at most i, simulate M on w for i steps and output w if M accepts w in i steps. ∎