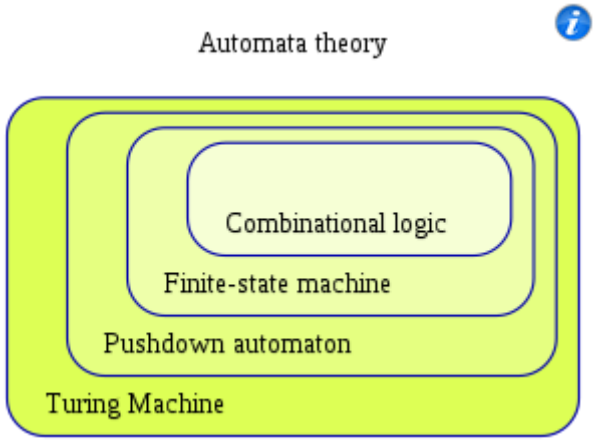
# Automata theory

**Automata theory** is the study of abstract machines and automata, as well as the computational problems that can be solved using them. It is a theory in theoretical computer science and discrete mathematics (a subject of study in both mathematics and computer science). The word *automata* (the plural of *automaton*) comes from the Greek word αὐτόματα, which means "self-making".

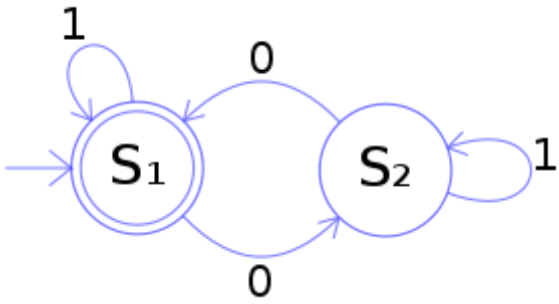
The figure at right illustrates a finite-state machine, which belongs to a well-known type of automaton. This automaton consists of states (represented in the figure by circles) and transitions (represented by arrows). As the automaton sees a symbol of input, it makes a transition (or jump) to another state, according to its transition function, which takes the current state and the recent symbol as its inputs.

Automata theory is closely related to formal language theory. An automaton is a finite representation of a formal language that may be an infinite set. Automata are often classified by the class of formal languages they can recognize, typically illustrated by the Chomsky hierarchy, which describes the relations between various languages and kinds of formalized logics.

Automata play a major role in theory of computation, compiler construction, artificial intelligence, parsing and formal verification.




Classes of automata




The study of the mathematical properties of such automata is automata theory. The picture is a visualization of an automaton that recognizes strings containing an even number of *0*s. The automaton starts in state *S1*, and transitions to the non-accepting state *S2* upon reading the symbol *0*. Reading another *0* causes the automaton to transition back to the accepting state *S1*. In both states the symbol *1* is ignored by making a transition to the current state.

## Contents

# Automata

Following is an introductory definition of one type of automaton, which attempts to help one grasp the essential concepts involved in automata theory/theories.

## Very informal description

An automaton is a construct made of *states* designed to determine if the input should be accepted or rejected. It looks a lot like a basic board game where each space on the board represents a state. Each state has information about what to do when an input is received by the machine (again, rather like what to do when you land on the *Jail* spot in a popular board game). As the machine receives a new input, it looks at the state and picks a new spot based on the information on what to do when it receives that input at that state. When there are no more inputs, the automaton stops and the space it is on when it completes determines whether the automaton accepts or rejects that particular set of inputs.

## Informal description

An automaton *runs* when it is given some sequence of *inputs* in discrete (individual) *time steps* or steps. An automaton processes one input picked from a set of *symbols* or *letters*, which is called an *alphabet*. The symbols received by the automaton as input at any step are a finite sequence of symbols called *words*. An automaton has a finite set of *states*. At each moment during a run of the automaton, the automaton is *in* one of its states. When the automaton receives new input it moves to another state (or transitions) based on a function that takes the current state and symbol as parameters. This function is called the *transition function*. The automaton reads the symbols of the input word one after another and transitions from state to state according to the transition function until the word is read completely. Once the input word has been read, the automaton is said to have stopped. The state at which the automaton stops is called the final state. Depending on the final state, it's said that the automaton either *accepts* or *rejects* an input word. There is a subset of states of the automaton, which is defined as the set of *accepting states*. If the final state is an accepting state, then the automaton *accepts* the word. Otherwise, the word is *rejected*. The set of all the words accepted by an automaton is called the *language recognized by the automaton*.

In short, an automaton is a mathematical object that takes a word as input and decides whether to accept it or reject it. Since all computational problems are reducible into the accept/reject question on inputs, (all problem instances can be represented in a finite length of symbols), automata theory plays a crucial role in computational theory.

## Formal definition

### Automaton

#### definition of finite state automata

A deterministic finite **automaton** is represented formally by a 5-tuple **<Q, Σ, δ,$q_0$,F>**, where:

- Q is a finite set of *states*.
- Σ is a finite set of *symbols*, called the *alphabet* of the automaton.
- δ is the **transition function**, that is, δ: Q × Σ → Q.
- $q_0$ is the *start state*, that is, the state of the automaton before any input has been processed, where $q_0 \in Q$.
- F is a set of states of Q (i.e. $F \subseteq Q$) called **accept states**.

**Input word**

An automaton reads a finite string of symbols $a_1,a_2,....,a_n$ , where $a_i \in \Sigma$, which is called an *input word*. The set of all words is denoted by Σ*.

**Run**

A sequence of states $q_0,q_1,q_2,....,q_n$, where $q_i \in Q$ such that $q_0$ is the start state and $q_i = \delta(q_{i-1},a_i)$ for $0 < i \leq n$, is a *run* of the automaton on an input word $w = a_1,a_2,....,a_n \in \Sigma^*$. In other words, at first the automaton is at the start state $q_0$, and then the automaton reads symbols of the input word in sequence. When the automaton reads symbol $a_i$ it jumps to state $q_i = \delta(q_{i-1},a_i)$. $q_n$ is said to be the *final state* of the run.

**Accepting word**

A word $w \in \Sigma^*$ is accepted by the automaton if $q_n \in F$.

**Recognized language**

An automaton can recognize a formal language. The language $L \subseteq \Sigma^*$ recognized by an automaton is the set of all the words that are accepted by the automaton.

**Recognizable languages**

The recognizable languages are the set of languages that are recognized by some automaton. For the above definition of automata the recognizable languages are regular languages. For different definitions of automata, the recognizable languages are different.

# Variant definitions of automata

Automata are defined to study useful machines under mathematical formalism. So, the definition of an automaton is open to variations according to the "real world machine", which we want to model using the automaton. People have studied many variations of automata. The most standard variant, which is described above, is called a deterministic finite automaton. The following are some popular variations in the definition of different components of automata.

### Input

- *Finite input*: An automaton that accepts only finite sequence of symbols. The above introductory definition only encompasses finite words.
- *Infinite input*: An automaton that accepts infinite words (ω-words). Such automata are called *ω-automata*.
- *Tree word input*: The input may be a *tree of symbols* instead of sequence of symbols. In this case after reading each symbol, the automaton *reads* all the successor symbols in the input tree. It is said that the automaton *makes one copy* of itself for each successor and each such copy starts running on one of the successor symbols from the state according to the transition relation of the automaton. Such an automaton is called a tree automaton.
- *Infinite tree input* : The two extensions above can be combined, so the automaton reads a tree structure with (in)finite branches. Such an automaton is called an infinite tree automaton

### States

- *Finite states*: An automaton that contains only a finite number of states. The above introductory definition describes automata with finite numbers of states.
- *Infinite states*: An automaton that may not have a finite number of states, or even a countable number of states. For example, the quantum finite automaton or topological automaton has uncountable infinity of states.
- *Stack memory*: An automaton may also contain some extra memory in the form of a stack in which symbols can be pushed and popped. This kind of automaton is called a *pushdown automaton*

### Transition function

- *Deterministic*: For a given current state and an input symbol, if an automaton can only jump to one and only one state then it is a *deterministic automaton*.
- *Nondeterministic*: An automaton that, after reading an input symbol, may jump into any of a number of states, as licensed by its transition relation. Notice that the term transition function is replaced by transition relation: The automaton *non-deterministically* decides to jump into one of the allowed choices. Such automata are called *nondeterministic automata*.
- *Alternation*: This idea is quite similar to tree automaton, but orthogonal. The automaton may run its *multiple copies* on the *same* next read symbol. Such automata are called *alternating automata*. Acceptance condition must satisfy all runs of such *copies* to accept the input.

### Acceptance condition

- *Acceptance of finite words*: Same as described in the informal definition above.
- *Acceptance of infinite words*: an *omega automaton* cannot have final states, as infinite words never terminate. Rather, acceptance of the word is decided by looking at the infinite sequence of visited states during the run.
- *Probabilistic acceptance*: An automaton need not strictly accept or reject an input. It may accept the input with some probability between zero and one. For example, quantum finite automaton, geometric automaton and metric automaton have probabilistic acceptance.

Different combinations of the above variations produce many classes of automaton.

Automata theory is a subject matter that studies properties of various types of automata. For example, the following questions are studied about a given type of automata.

- Which class of formal languages is recognizable by some type of automata? (Recognizable languages)
- Are certain automata *closed* under union, intersection, or complementation of formal languages? (Closure properties)
- How expressive is a type of automata in terms of recognizing a class of formal languages? And, their relative expressive power? (Language hierarchy)

Automata theory also studies the existence or nonexistence of any effective algorithms to solve problems similar to the following list:

- Does an automaton accept any input word? (Emptiness checking)
- Is it possible to transform a given non-deterministic automaton into deterministic automaton without changing the recognizable language? (Determinization)
- For a given formal language, what is the smallest automaton that recognizes it? (Minimization)

# Classes of automata

The following is an incomplete list of types of automata.

| Automaton | Recognizable language |
|---|---|
| Nondeterministic/Deterministic Finite state machine (FSM) | regular languages |
| Deterministic pushdown automaton (DPDA) | deterministic context-free languages |
| Pushdown automaton (PDA) | context-free languages |
| Linear bounded automaton (LBA) | context-sensitive languages |
| Turing machine | recursively enumerable languages |
| Deterministic Büchi automaton | ω-limit languages |
| Nondeterministic Büchi automaton | ω-regular languages |
| Rabin automaton, Streett automaton, Parity automaton, Muller automaton | |

## Discrete, continuous, and hybrid automata

Normally automata theory describes the states of abstract machines but there are analog automata or continuous automata or hybrid discrete-continuous automata, which use analog data, continuous time, or both.

# Hierarchy in terms of powers

The following is an incomplete hierarchy in terms of powers of different types of virtual machines. The hierarchy reflects the nested categories of languages the machines are able to accept.[1]

| Automaton |
| --- |
| Deterministic Finite Automaton (DFA) -- Lowest Power<br>(same power) || (same power)<br>Nondeterministic Finite Automaton (NFA)<br>(above is weaker) ∩ (below is stronger)<br>Deterministic Push Down Automaton (DPDA-I)<br>with 1 push-down store<br>∩<br>Nondeterministic Push Down Automaton (NPDA-I)<br>with 1 push-down store<br>∩<br>Linear Bounded Automaton (LBA)<br>∩<br>Deterministic Push Down Automaton (DPDA-II)<br>with 2 push-down stores<br>||<br>Nondeterministic Push Down Automaton (NPDA-II)<br>with 2 push-down stores<br>||<br>Deterministic Turing Machine (DTM)<br>||<br>Nondeterministic Turing Machine (NTM)<br>||<br>Probabilistic Turing Machine (PTM)<br>||<br>Multitape Turing Machine (MTM)<br>||<br>Multidimensional Turing Machine |

# Applications

Each model in automata theory plays important roles in several applied areas. Finite automata are used in text processing, compilers, and hardware design. Context-free grammar (CFGs) are used in programming languages and artificial intelligence. Originally, CFGs were used in the study of the human languages. Cellular automata are used in the field of biology, the most common example being John Conway's Game of Life. Some other examples which could be explained using automata theory in biology include mollusk and pine cones growth and pigmentation patterns. Going further, a theory suggesting that the whole universe is computed by some sort of a discrete automaton, is advocated by some scientists. The idea originated in the work of Konrad Zuse, and was popularized in America by Edward Fredkin. Automata also appear in the theory of finite fields: the set of irreducible polynomials which can be written as composition of degree two polynomials is in fact a regular language.[2]

# Automata simulators

Automata simulators are pedagogical tools used to teach, learn and research automata theory. An automata simulator takes as input the description of an automaton and then simulates its working for an arbitrary input string. The description of the automaton can be entered in several ways. An automaton can be defined in a symbolic language or its specification may be entered in a predesigned form or its transition diagram may be drawn by clicking and dragging the mouse. Well known automata simulators include Turing's World, JFLAP, VAS, TAGS and SimStudio.[3]

# Connection to category theory

One can define several distinct categories of automata[4] following the automata classification into different types described in the previous section. The mathematical category of deterministic automata, sequential machines or *sequential automata*, and Turing machines with automata homomorphisms defining the arrows between automata is a Cartesian closed category,[5][6] it has both categorical limits and colimits. An automata homomorphism maps a quintuple of an automaton $A_\mathrm{i}$ onto the quintuple of another automaton $A_\mathrm{j}$.[7] Automata homomorphisms can also be considered as *automata transformations* or as semigroup homomorphisms, when the state space, $\boldsymbol{S}$, of the automaton is defined as a semigroup $\mathbf{S}_\mathrm{g}$. Monoids are also considered as a suitable setting for automata in monoidal categories.[8][9][10]

### Categories of variable automata

One could also define a *variable automaton*, in the sense of Norbert Wiener in his book on *The Human Use of Human Beings* *via* the endomorphisms $\boldsymbol{A_i \rightarrow A_i}$. Then, one can show that such variable automata homomorphisms form a mathematical group. In the case of non-deterministic, or other complex kinds of automata, the latter set of endomorphisms may become, however, a *variable automaton groupoid*. Therefore, in the most general case, categories of variable automata of any kind are categories of groupoids or groupoid categories. Moreover, the category of reversible automata is then a 2-category, and also a subcategory of the 2-category of groupoids, or the groupoid category.

# History

The automata theory was developed in the mid-20th century in connection with finite automata.[11]

# See also

- Boolean differential calculus

# References


1. Yan, Song Y. (1998). *An Introduction to Formal Languages and Machine Computation* (https://books.google.com/books?id=ySOwQgAACAAJ). Singapore: World Scientific Publishing Co. Pte. Ltd. pp. 155–156. ISBN 9789810234225.
2. Ferraguti, A.; Micheli, G.; Schnyder, R. (2018), *Irreducible compositions of degree two polynomials over finite fields have regular structure*, The Quarterly Journal of Mathematics, **69**, Oxford University Press, pp. 1089–1099, arXiv:1701.06040 (https://arxiv.org/abs/1701.06040), doi:10.1093/qmath/hay015 (https://doi.org/10.1093%2Fqmath%2Fhay015)
3. Chakraborty, P., Saxena, P. C., Katti, C. P. 2011. Fifty Years of Automata Simulation: A Review. *ACM Inroads*, **2**(4):59–70. http://dl.acm.org/citation.cfm?id=2038893&dl=ACM&coll=DL&CFID=65021406&CFTOKEN=86634854

4. Jirí Adámek and Věra Trnková. 1990. *Automata and Algebras in Categories*. Kluwer Academic Publishers:Dordrecht and Prague
5. S. Mac Lane, Categories for the Working Mathematician, Springer, New York (1971)
6. Cartesian closed category (http://planetmath.org/encyclopedia/CartesianClosedCategory.html) Archived (https://web.archive.org/web/20111116215852/http://planetmath.org/encyclopedia/CartesianClosedCategory.html) November 16, 2011, at the Wayback Machine
7. The Category of Automata (http://planetmath.org/encyclopedia/SequentialMachine3.html) Archived (https://web.archive.org/web/20110915074653/http://planetmath.org/encyclopedia/SequentialMachine3.html) September 15, 2011, at the Wayback Machine
8. http://www.math.cornell.edu/~worthing/asl2010.pdf James Worthington.2010.Determinizing, Forgetting, and Automata in Monoidal Categories. ASL North American Annual Meeting, March 17, 2010
9. Aguiar, M. and Mahajan, S.2010. *"Monoidal Functors, Species, and Hopf Algebras"*.
10. Meseguer, J., Montanari, U.: 1990 Petri nets are monoids. *Information and Computation* **88**:105–155
11. http://www.rutherfordjournal.org/article030107.html

# Further reading

- John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman (2000). *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*. Pearson Education. ISBN 978-0-201-44124-6.
- Michael Sipser (1997). *Introduction to the Theory of Computation*. PWS Publishing. ISBN 978-0-534-94728-6. Part One: Automata and Languages, chapters 1–2, pp. 29–122. Section 4.1: Decidable Languages, pp. 152–159. Section 5.1: Undecidable Problems from Language Theory, pp. 172–183.
- Elaine Rich (2008). *Automata, Computability and Complexity: Theory and Applications*. Pearson. ISBN 978-0-13-228806-4.
- Salomaa, Arto (1985). *Computation and automata* (https://archive.org/details/computationautom0000salo). Encyclopedia of Mathematics and Its Applications. **25**. Cambridge University Press. ISBN 978-0-521-30245-6. Zbl 0565.68046 (https://zbmath.org/?format=complete&q=an:0565.68046).
- Anderson, James A. (2006). *Automata theory with modern applications*. With contributions by Tom Head. Cambridge: Cambridge University Press. ISBN 978-0-521-61324-8. Zbl 1127.68049 (https://zbmath.org/?format=complete&q=an:1127.68049).
- Conway, J.H. (1971). *Regular algebra and finite machines*. Chapman and Hall Mathematics Series. London: Chapman & Hall. Zbl 0231.94041 (https://zbmath.org/?format=complete&q=an:0231.94041).
- John M. Howie (1991) *Automata and Languages*, Clarendon Press ISBN 0-19-853424-8 MR1254435 (https://mathscinet.ams.org/mathscinet-getitem?mr=1254435)
- Sakarovitch, Jacques (2009). *Elements of automata theory*. Translated from the French by Reuben Thomas. Cambridge University Press. ISBN 978-0-521-84425-3. Zbl 1188.68177 (https://zbmath.org/?format=complete&q=an:1188.68177).
- James P. Schmeiser, David T. Barnard (1995). *Producing a top-down parse order with bottom-up parsing*. Elsevier North-Holland.
- Igor Aleksander, F.Keith Hanna (1975). *Automata Theory : An Engineering Approach*. New York: Crane Russak. ISBN 978-0-8448-0657-0.
- Marvin Minsky (1967). *Computation : Finite and infinite machines* (https://archive.org/details/computationfinit0000mins). Princeton, N.J.: Prentice Hall.

- John C. Martin (2011). *Introduction to Languages and The Theory of Computation*. New York, NY 10020: McGraw Hill. ISBN 978-0-07-319146-1.


# External links

- Visual Automata Simulator (https://web.archive.org/web/20090503173000/http://www.cs.usfca.edu/~jbovet/vas.html), A tool for simulating, visualizing and transforming finite state automata and Turing Machines, by Jean Bovet
- JFLAP (http://www.jflap.org)
- dk.brics.automaton (http://www.brics.dk/automaton)
- libfa (http://www.augeas.net/libfa/index.html)