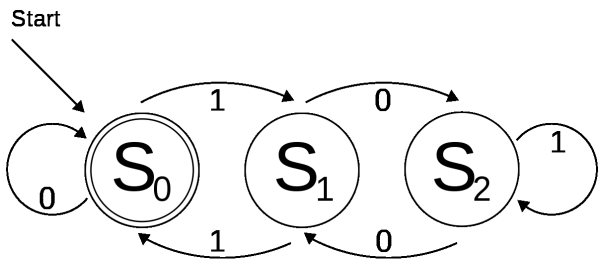
Deterministic finite automaton

"DFSA" redirects here. DFSA may also refer to Drug facilitated sexual assault.

In automata theory, a branch of theoretical computer science, a **deterministic finite automaton** (**DFA**)—also known as **deterministic finite state machine**—is a finite state machine that accepts/rejects finite strings of symbols and only produces a unique computation (or run) of the automaton for each input string.[1] 'Deterministic' refers to the uniqueness of the computation. In search of simplest models to capture the finite state machines, McCulloch and Pitts were among the first researchers to introduce a concept similar to finite automaton in 1943.[2][3]



An example of a deterministic finite automaton that accepts only binary numbers that are multiples of 3. The state S_0 is both the start state and an accept state.

The figure on the right illustrates a deterministic finite automaton using a state diagram. In the automaton, there are three states: S0, S1, and S2 (denoted graphically by circles). The automaton takes a finite sequence of 0s and 1s as input. For each state, there is a transition arrow leading out to a next state for both 0 and 1. Upon reading a symbol, a DFA jumps *deterministically* from a state to another by following the transition arrow. For example, if the automaton is currently in state S0 and current input symbol is 1 then it deterministically jumps to state S1. A DFA has a *start state* (denoted graphically by an arrow coming in from nowhere) where computations begin, and a set of *accept states* (denoted graphically by a double circle) which help define when a computation is successful.

A DFA is defined as an abstract mathematical concept, but due to the deterministic nature of a DFA, it is implementable in hardware and software for solving various specific problems. For example, a DFA can model software that decides whether or not online user-input such as email addresses are valid. (see: finite state machine for more practical examples).

DFAs recognize exactly the set of regular languages which are, among other things, useful for doing lexical analysis and pattern matching. DFAs can be built from nondeterministic finite automata (NFAs) using the powerset construction method.

Formal definition

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

- a finite set of states (Q)
- a finite set of input symbols called the alphabet (Σ)
- a transition function ($\delta : Q \times \Sigma \to Q$)
- a start state ($q_0 \in Q$)
- a set of accept states ($F \subseteq Q$)

Let $w = a_1 a_2 \ldots a_n$ be a string over the alphabet Σ. The automaton M accepts the string w if a sequence of states, $r_0, r_1, \ldots, r_n$, exists in Q with the following conditions:

1. $r_0 = q_0$
2. $r_{i+1} = \delta(r_i, a_{i+1})$, for $i = 0, \ldots, n-1$
3. $r_n \in F$.

In words, the first condition says that the machine starts in the start state q_0. The second condition says that given each character of string w, the machine will transition from state to state according to the transition function δ. The last condition says that the machine accepts w if the last input of w causes the machine to halt in one of the accepting

states. Otherwise, it is said that the automaton *rejects* the string. The set of strings M accepts is the language *recognized* by M and this language is denoted by $L(M)$.

A deterministic finite automaton without accept states and without a starting state is known as a transition system or semiautomaton.

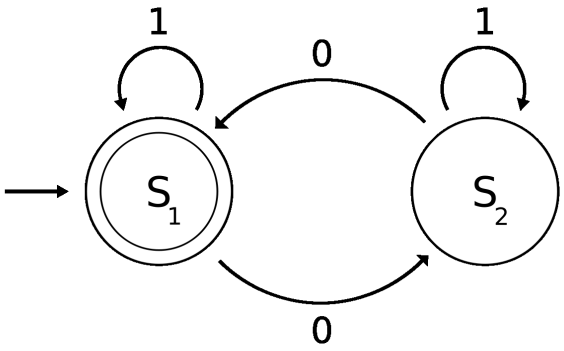
For more comprehensive introduction of the formal definition see automata theory.

Example

The following example is of a DFA M, with a binary alphabet, which requires that the input contains an even number of 0s.

$M = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = \{S_1, S_2\}$,
- $\Sigma = \{0, 1\}$,
- $q_0 = S_1$,
- $F = \{S_1\}$, and
- δ is defined by the following state transition table:



The state diagram for M

	0	**1**
S_1	S_2	S_1
S_2	S_1	S_2

The state S_1 represents that there has been an even number of 0s in the input so far, while S_2 signifies an odd number. A 1 in the input does not change the state of the automaton. When the input ends, the state will show whether the input contained an even number of 0s or not. If the input did contain an even number of 0s, M will finish in state S_1, an accepting state, so the input string will be accepted.

The language recognized by M is the regular language given by the regular expression 1*(0 (1*) 0 (1*))*, where "*" is the Kleene star, e.g., 1* denotes any non-negative number (possibly zero) of symbols "1".

Closure properties

If DFAs recognize the languages that are obtained by applying an operation on the DFA recognizable languages then DFAs are said to be closed under the operation. The DFAs are closed under the following operations.

- Union
- Intersection
- Concatenation
- Negation
- Kleene closure
- Reversal
- Init
- Quotient
- Substitution

- Homomorphism

Since DFAs are equivalent to nondeterministic finite automata (NFA), these closures may be proved using closure properties of NFA.

Accept and Generate modes

A DFA representing a regular language can be used either in an accepting mode to validate that an input string is part of the language, or in a generating mode to generate a list of all the strings in the language.

In the accept mode an input string is provided which the automaton can read in left to right, one symbol at a time. The computation begins at the start state and proceeds by reading the first symbol from the input string and following the state transition corresponding to that symbol. The system continues reading symbols and following transitions until there are no more symbols in the input, which marks the end of the computation. If after all input symbols have been processed the system is in an accept state then we know that the input string was indeed part of the language, and it is said to be accepted, otherwise it is not part of the language and it is not accepted.

The generating mode is similar except that rather than validating an input string its goal is to produce a list of all the strings in the language. Instead of following a single transition out of each state, it follows all of them. In practice this can be accomplished by massive parallelism (having the program branch into two or more processes each time it is faced with a decision) or through recursion. As before, the computation begins at the start state and then proceeds to follow each available transition, keeping track of which branches it took. Every time the automaton finds itself in an accept state it knows that the sequence of branches it took forms a valid string in the language and it adds that string to the list that it is generating. If the language this automaton describes is infinite (ie contains an infinite number or strings, such as "all the binary string with an even number of 0s) then the computation will never halt. Given that regular languages are, in general, infinite, automata in the generating mode tends to be more of a theoretical construct Wikipedia:Citation needed.

DFA as a transition monoid

Alternatively a run can be seen as a sequence of compositions of transition function with itself. Given an input symbol $a \in \Sigma$, one may write the transition function as $\delta_a : Q \to Q$, using the simple trick of currying, that is, writing $\delta(q, a) = \delta_a(q)$ for all $q \in Q$. This way, the transition function can be seen in simpler terms: it's just something that "acts" on a state in Q, yielding another state. One may then consider the result of function composition repeatedly applied to the various functions δ_a, δ_b, and so on. Using this notion we define $\widehat{\delta} : Q \times \Sigma^* \to Q$. Given a pair of letters $a, b \in \Sigma$, one may define a new function $\widehat{\delta}$, by insisting that $\widehat{\delta}_{ab} = \delta_a \circ \delta_b$, where $\circ$ denotes function composition. Clearly, this process can be recursively continued. So, we have following recursive definition

$\widehat{\delta}(q, \epsilon) = q.$ where ϵ is empty string and

$\widehat{\delta}(q, wa) = \delta_a(\widehat{\delta}(q, w)).$ where $w \in \Sigma^*, a \in \Sigma$ and $q \in Q$.

$\widehat{\delta}$ is defined for all words $w \in \Sigma^*$. Repeated function composition forms a monoid. For the transition functions, this monoid is known as the transition monoid, or sometimes the *transformation semigroup*. The construction can also be reversed: given a $\widehat{\delta}$, one can reconstruct a δ, and so the two descriptions are equivalent.

Local automata

A **local automaton** is a DFA for which all edges with the same label lead to a single vertex. Local automata accept the class of local languages, those for which membership of a word in the language is determined by a "sliding window" of length two on the word.[4][5]

A **Myhill graph** over an alphabet A is a directed graph with vertex set A and subsets of vertices labelled "start" and "finish". The language accepted by a Myhill graph is the set of directed paths from a start vertex to a finish vertex: the graph thus acts as an automaton. The class of languages accepted by Myhill graphs is the class of local languages.[6]

Advantages and disadvantages

DFAs were invented to model *real world* finite state machines in contrast to the concept of a Turing machine, which was too general to study properties of real world machines.

DFAs are one of the most practical models of computation, since there is a trivial linear time, constant-space, online algorithm to simulate a DFA on a stream of input. Also, there are efficient algorithms to find a DFA recognizing:

- the complement of the language recognized by a given DFA.
- the union/intersection of the languages recognized by two given DFAs.

Because DFAs can be reduced to a *canonical form* (minimal DFAs), there are also efficient algorithms to determine:

- whether a DFA accepts any strings
- whether a DFA accepts all strings
- whether two DFAs recognize the same language
- the DFA with a minimum number of states for a particular regular language

DFAs are equivalent in computing power to nondeterministic finite automata (NFAs). This is because, firstly any DFA is also an NFA, so an NFA can do what a DFA can do. Also, given an NFA, using the powerset construction one can build a DFA that recognizes the same language as the NFA, although the DFA could have exponentially larger number of states than the NFA.[7][8]

On the other hand, finite state automata are of strictly limited power in the languages they can recognize; many simple languages, including any problem that requires more than constant space to solve, cannot be recognized by a DFA. The classical example of a simply described language that no DFA can recognize is bracket or Dyck language, i.e., the language that consists of properly paired brackets such as word "(()())". Intuitively, no DFA can recognize the bracket language because there is no limit to recursion, i.e., one can always embed another pair of brackets inside, and hence would require an infinite number of states to recognize. Another simpler example is the language consisting of strings of the form $a^n b^n$ *for some finite but arbitrary number of* a's, *followed by an equal number of* b's.[9]

Notes

[1] Hopcroft 2001:
[2] McCulloch and Pitts (1943):
[3] Rabin and Scott (1959):
[4] Lawson (2004) p.129
[5] Sakarovitch (2009) p.228
[6] Lawson (2004) p.128
[7] Sakarovitch (2009) p.105
[8] Lawson (2004) p.63
[9] Lawson (2004) p.46

References

- Hopcroft, John E.; Motwani, Rajeev; Ullman, Jeffrey D. (2001). *Introduction to Automata Theory, Languages, and Computation* (http://www.pearsonhighered.com/educator/product/Introduction-to-Automata-Theory-Languages-and-Computation/9780201441246.page) (2 ed.). Addison Wesley. ISBN 0-201-44124-1. Retrieved 19 November 2012.
- Lawson, Mark V. (2004). *Finite automata*. Chapman and Hall/CRC. ISBN 1-58488-255-7. Zbl 1086.68074 (http://www.zentralblatt-math.org/zmath/en/search/?format=complete&q=an:1086.68074).
- McCulloch, W. S.; Pitts, E. (1943). "A logical calculus of the ideas imminent in nervous activity". *Bulletin of Mathematical Biophysics*: 541–544.
- Rabin, M. O.; Scott, D. (1959). "Finite automata and their decision problems.". *IBM J. Res. Develop.*: 114–125.
- Sakarovitch, Jacques (2009). *Elements of automata theory*. Translated from the French by Reuben Thomas. Cambridge: Cambridge University Press. ISBN 978-0-521-84425-3. Zbl 1188.68177 (http://www.zentralblatt-math.org/zmath/en/search/?format=complete&q=an:1188.68177).
- Sipser, Michael (1997). *Introduction to the Theory of Computation*. Boston: PWS. ISBN 0-534-94728-X.. Section 1.1: Finite Automata, pp. 31–47. Subsection "Decidable Problems Concerning Regular Languages" of section 4.1: Decidable Languages, pp. 152–155.4.4 DFA can accept only regular language

External links

- DFA Simulator - an open source graphical editor and simulator of DFA (http://home.arcor.de/kai.w1986/dfasimulator/)

Article Sources and Contributors

Deterministic finite automaton *Source*: http://en.wikipedia.org/w/index.php?oldid=607409993 *Contributors*: 1ForTheMoney, 2001:db8, A3 nm, Afa86, Alexsmail, Andrasek, Ashutosh y0078, Beland, Bináris, Booyabazooka, Canderra, Charles Matthews, ChrisGualtieri, Crazytales, Curseofgnome, Cyrius, Dcoetzee, Decrypt3, Deltahedron, Epbr123, Eric119, FauxFaux, Frap, Grunt, Hermel, Insanity Incarnate, Interiot, JDCMAN, JameySharp, Jaredwf, Jason Quinn, Jiri 1984, Jonsafari, LOL, Linas, Loadmaster, Mararo, Marozols, Mike Cline, Mlwilson, Neurodivergent, Nivix, Ounsworth, Pacdude9, Paulbmann, Prabhakant Shukla, RPHv, Radiojon, Raghunandan ma, Rbonvall, Reyk, S.K., S.eijadi, Sahuagin, SaintNULL, Satellizer, Spoon!, Sviemeister, Tardis, Tbtietc, ThomasOwens, Thowa, Thumperward, Tobias Bergemann, Vevek, Wittylama, Wyverald, Xvr, ZeroOne, Zxombie, سعی, 83 anonymous edits

Image Sources, Licenses and Contributors

File:DFA example multiplies of 3.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:DFA_example_multiplies_of_3.svg *License*: Public Domain *Contributors*: Self-made

File:DFAexample.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:DFAexample.svg *License*: Public Domain *Contributors*: Cepheus

License