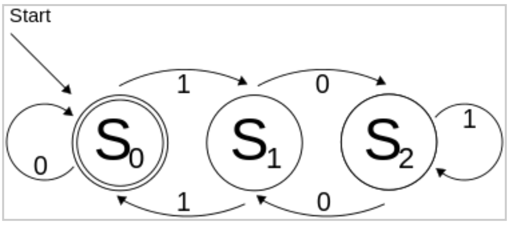
Deterministic finite automaton

From Wikipedia, the free encyclopedia

In the theory of computation, a branch of theoretical computer science, a **deterministic finite automaton** (**DFA**)—also known as a **deterministic finite accepter** (**DFA**) and a **deterministic finite state machine** (**DFSA**)—is a finite-state machine that accepts and 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 the simplest models to capture 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 illustrates a deterministic finite automaton using a state diagram. In the automaton, there are three states: S_0, S_1, and S_2 (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 one state to another by following the transition arrow. For example, if the automaton is currently in state S_0 and the current input symbol is 1, then it deterministically jumps to state S_1. 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 is often implemented 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.[4]

DFAs recognize exactly the set of regular languages,[1] 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.

Contents

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 ($\delta : Q \times \Sigma \rightarrow Q$)
- an initial or start state ($q_0 \in Q$)
- a set of accept states ($F \subseteq Q$)

Let $w = a_1a_2 \ldots a_n$ 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 that 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.

Complete and incomplete

According to the above definition, deterministic finite automata are always *complete*: they define a transition for each state and each input symbol.

While this is the most common definition, some authors use the term deterministic finite automaton for a slightly different notion: an automaton that defines *at most* one transition for each state and each input symbol; the transition function is allowed to be partial. When no transition is defined, such an automaton halts.

Example

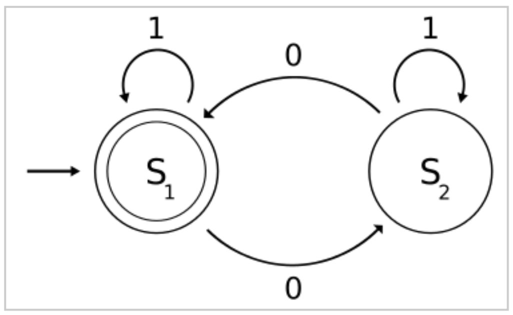
The following example is of a DFA M, with a binary alphabet, which requires that the input contains an even number of 0s.

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

	0	**1**
$\boldsymbol{S_1}$	S_2	S_1
$\boldsymbol{S_2}$	S_1	S_2



The state diagram for M

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)*, 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.

As a transition monoid

A run of a given DFA can be seen as a sequence of compositions of a very general formulation of the transition function with itself. Here we construct that function.

For a given input symbol $a \in \Sigma$, one may construct a transition function $\delta_a : Q \to Q$ by defining $\delta_a(q) = \delta(q, a)$ for all $q \in Q$. (This trick is called currying.) From this perspective, δ_a "acts" on a state in Q to yield another state. One may then consider the result of function composition repeatedly applied to the various functions δ_a, δ_b, and so on. Given a pair of letters $a, b \in \Sigma$, one may define a new function $\hat{\delta}_{ab} = \delta_a \circ \delta_b$, where $\circ$ denotes function composition.

Clearly, this process may be recursively continued, giving the following recursive definition of $\hat{\delta} : Q \times \Sigma^{\star} \to Q$:

$\hat{\delta}(q, \epsilon) = q$, where ϵ is the empty string and
$\hat{\delta}(q, wa) = \delta_a(\hat{\delta}(q, w))$, where $w \in \Sigma^*, a \in \Sigma$ and $q \in Q$.

$\hat{\delta}$ is defined for all words $w \in \Sigma^*$. A run of the DFA is a sequence of compositions of $\hat{\delta}$ with itself.

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 $\hat{\delta}$, 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.[5][6]

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.[5] The class of languages accepted by Myhill graphs is the class of local languages.[7]

Random

When the start state and accept states are ignored, a DFA of n states and an alphabet of size k can be seen as a digraph of n vertices in which all vertices have k out-arcs labeled $1, \ldots, k$ (a k-out digraph). It is known that when $k \geq 2$ is a fixed integer, with high probability, the largest strongly connected component (SCC) in such a k-out digraph chosen uniformly at random is of linear size and it can be reached by all vertices.[8] It has also been proven that if k is allowed to increase as n increases, then the whole digraph has a phase transition for strong connectivity similar to Erdős–Rényi model for connectivity.[9]

In a random DFA, the maximum number of vertices reachable from one vertex is very close to the number of vertices in the largest SCC with high probability.[8][10] This is also true for the largest induced sub-digraph of minimum in-degree one, which can be seen as a directed version of 1-core.[9]

Advantages and disadvantages

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.[11][12]

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 classic 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 Dyck language because DFAs are not capable of counting: a DFA-like automaton needs to have a state to represent any possible number of "currently open" parentheses, meaning it would need an unbounded number of states. Another simpler example is the language consisting of strings of the form a^nb^n *for some finite but arbitrary number of* a*'s, followed by an equal number of* b*'s.*[13]

See also

- Acyclic deterministic finite automata
- DFA minimization
- Monadic second-order logic
- NFA to DFA conversion
- Quantum finite automata
- Read-only right moving Turing machines
- Separating words problem
- Turing machine
- Two-way deterministic finite automaton

Notes

1. Hopcroft 2001:
2. McCulloch and Pitts (1943):
3. Rabin and Scott (1959):
4. Gouda, Prabhakar, *Application of Finite automata*
5. Lawson (2004) p.129
6. Sakarovitch (2009) p.228
7. Lawson (2004) p.128
8. Grusho, A. A. (1973). "Limit distributions of certain characteristics of random automaton graphs". *Mathematical Notes of the Academy of Sciences of the USSR*. **4**: 633–637. doi:10.1007/BF01095785.
9. Cai, X.S.; Devroye, L. "The graph structure of a deterministic automaton chosen at random: full version". arXiv:1504.06238.
10. Carayol, Arnaud; Nicaud,, Cyril (2012). "Distribution of the number of accessible states in a random deterministic automaton".
11. Sakarovitch (2009) p.105
12. Lawson (2004) p.63
13. Lawson (2004) p.46

References

- Hopcroft, John E.; Motwani, Rajeev; Ullman, Jeffrey D. (2001). *Introduction to Automata Theory, Languages, and Computation* (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.
- McCulloch, W. S.; Pitts, W. (1943). "A Logical Calculus of the Ideas Immanent in Nervous Activity". *Bulletin of Mathematical Biophysics*. **5** (4): 115–133.
- 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.
- 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 Operations,Examples,Minimization on scanftree.com (http://scanftree.com/automata/)
- DFA Simulator - an open source graphical editor and simulator of DFA (http://home.arcor.de/kai.w1986/dfas imulator/)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Deterministic_finite_automaton&oldid=741838082"

Categories: Finite automata