deterministic finite automaton

Canonical name	DeterministicFiniteAutomaton
Date of creation	2013-03-22 12:26:37
Last modified on	2013-03-22 12:26:37
Owner	CWoo (3771)
Last modified by	CWoo (3771)
Numerical id	16
Author	CWoo (3771)
Entry type	Definition
Classification	msc 68Q42
Classification	msc 68Q05
Classification	msc 03D10
Synonym	dfa
Synonym	finite state machine
Synonym	fsm
Related topic	Automaton
Related topic	ContextFreeLanguage
Related topic	RegularLanguage
Related topic	Language
Related topic	NonDeterministicFiniteAutomaton
Related topic	SubsetConstruction

A *deterministic finite automaton* (or DFA) is a deterministic automaton with a finite input alphabet and a finite number of states. It can be formally defined as a 5-tuple $(S, \Sigma, \delta, q_0, F)$, where

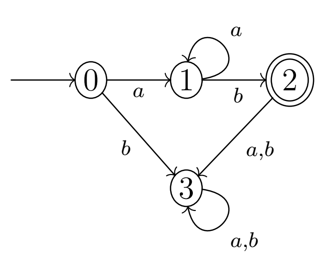
- S is a non-empty finite set of *states*,
- Σ is the alphabet (defining what set of input strings the automaton operates on),
- $\delta : S \times \Sigma \to S$ is the *transition function*,
- $q_0 \in S$ is the starting state, and
- $F \subseteq S$ is a set of final (or accepting states).

A DFA works exactly like a general automaton: operation begins at q_0, and movement from state to state is governed by the transition function δ. A word is accepted exactly when a final state is reached upon reading the last (rightmost) symbol of the word.

DFAs represent regular languages, and can be used to test whether any string in Σ^* is in the language it represents. Consider the following regular language over the alphabet $\Sigma := \{\mathtt{a}, \mathtt{b}\}$ (represented by the regular expression `aa*b`):

$$
\begin{aligned}
<S> \quad &\texttt{::=} \quad \mathtt{a}\,A \\
<A> \quad &\texttt{::=} \quad b \mid \mathtt{a}\,A
\end{aligned}
$$

This language can be represented by the DFA with the following state diagram:



The vertex 0 is the initial state q_0, and the vertex 2 is the only state in F. Note that for every vertex there is an edge leading away from it with a label for each symbol in Σ. This is a requirement of DFAs, which guarantees that operation is well-defined for any finite string.

If given the string `aaab` as input, operation of the DFA above is as follows. The first `a` is removed from the input string, so the edge from 0 to 1 is followed. The resulting input string is `aab`. For each of the next two `a`s, the edge is followed from 1 to itself. Finally, `b` is read from the input string and the edge from 1 to 2 is followed. Since the input string is now λ, the operation of the DFA halts. Since it has halted in the accepting state 2, the string `aaab` is accepted as a sentence in the regular language implemented by this DFA.

Now let us trace operation on the string `aaaba`. Execution is as above, until state 2 is reached with `a` remaining in the input string. The edge from 2 to 3 is then followed and the operation of the DFA halts. Since 3 is not an accepting state for this DFA, `aaaba` is *not* accepted.

Remarks.

- A DFA can be modified to include *ϵ-transitions*EpsilonTransitions. But the resulting DFA can be simulated by another DFA (without any epsilon transitions).

- Although the operation of a DFA is much easier to compute than that of a non-deterministic automaton, it is non-trivial to directly generate a DFA from a regular grammar. It is much easier to generate a non-deterministic finite automaton from the regular grammar, and then *transform* the non-deterministic finite automaton into a DFA.