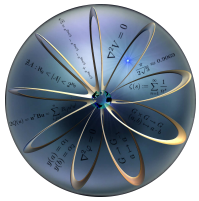


planetmath.org

Math for the people, by the people.

non-deterministic finite automaton

Canonical name	NondeterministicFiniteAutomaton
Date of creation	2013-03-22 12:26:40
Last modified on	2013-03-22 12:26:40
Owner	mps (409)
Last modified by	mps (409)
Numerical id	14
Author	mps (409)
Entry type	Definition
Classification	msc 68Q05
Classification	msc 68Q42
Classification	msc 03D10
Synonym	NDFA
Related topic	DeterministicFiniteAutomaton
Related topic	Automaton
Related topic	RegularLanguage
Related topic	ContextFreeLanguage
Related topic	Language
Related topic	PushdownAutomaton

A non-deterministic finite automaton (or NDFA) can be formally defined as a 5-tuple $(S, \Sigma, \delta, q_0, F)$, where

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

Some authors also relax the fourth condition by permitting multiple starting states (see remark below).

Note how this definition differs from that of a deterministic finite automaton (DFA) only by the definition of the transition function δ. Operation of the NDFA begins at q_0, and movement from state to state is governed by the transition function δ.

The transition function takes the first symbol of the (remaining) input string and the current state as its input, and after the transition this first symbol is removed only if the transition is defined for a symbol in Σ instead of λ. Conceptually, all possible transitions from a current state are followed *simultaneously* (hence the non-determinism). Once every possible transition has been executed, the NDFA is halted. If any of the states reached upon halting are in F for some input string, *and* the entire input string is consumed to reach that state, then the NDFA accepts that string.

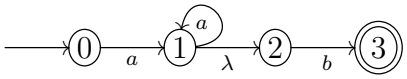
An NDFA can be represented visually as a directed graph called the state diagram. Circular vertices denote states, and the set of directed edges, labelled by symbols in $\Sigma \cup \lambda$, denotes T. The starting state q_0 is usually denoted by an arrow pointing to it that points from no other vertex. States in F are usually denoted by double circles.

NDFAs accept regular languages, and can be used to test whether any string in Σ^* is in the language it represents. Given an NDFA M, the language accepted by M is denoted by $L(M)$.

Consider the following regular language over the alphabet $\Sigma := \{\mathtt{a}, \mathtt{b}\}$ (represented by the regular expression `aa*b`):

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

This language can be represented by the NDFA with state diagram:



The vertex 0 is the initial state q_0, and the vertex 3 is the only state in F.

If given the string `aaab` as input, operation of the NDFA is as follows. Let $X \subseteq S \times \Sigma^*$ indicate the set of "current" states and the remaining input associated with them. Initially

$$X := \{(0, \texttt{aaab})\}\,.$$

For state 0 with a leading `a` as its input, the only possible transition to follow is to 1 (which consumes the `a`). This transforms X to

$$\{(1, \texttt{aab})\}\,.$$

Now there are two possible transitions to follow for state 1 with a leading `a`. One transition is back to 1, consuming the `a`, while the other is to 2, leaving the `a`. Thus X is then

$$\{(1, \texttt{ab}), (2, \texttt{aab})\}\,.$$

Again, the same transitions are possible for state 1, while no transition at all is available for state 2 with a leading `a`, so X is then

$$\{(1, \texttt{b}), (2, \texttt{aab}), (2, \texttt{ab})\}\,.$$

At this point, there is still no possible transition from 2, and the only possible transition from 1 is to 2 (leaving the input string as it is). This then gives

$$\{(2, \texttt{aab}), (2, \texttt{ab}), (2, \texttt{b})\}\,.$$

Only state 2 with remaining input of `b` has a transition leading from it, giving

$$\{(2, \texttt{aab}), (2, \texttt{ab}), (3, \lambda)\}\,.$$

At this point no further transitions are possible, and so the NDFA is halted. Since 3 is in F, and the input string can be reduced to λ when it reached 3, the NDFA accepts `aaab`.

If the input string were instead `aaaba`, processing would occur as before until

$$\{(2, \texttt{aaba}), (2, \texttt{aba}), (3, \texttt{a})\}$$

is reached and the NDFA halts. Although 3 is in F, it is not possible to reduce the input string completely before reaching 3. Therefore `aaaba` is *not* accepted by this NDFA.

Any regular grammar can be represented by an NDFA. Any string accepted by the NDFA is in the language represented by that NDFA. Furthermore, it is a straight-forward process to generate an NDFA for any regular grammar. Actual operation of an NDFA is generally intractable, but there is a simple process to transform any NDFA into a DFA, the operation of which is very tractable. Regular expression matchers tend to operate in this manner.

Remarks.

- Instead of a single starting state, one may more generally consider an NDFA M with a set I of starting states. However, this is not necessary, as an NDFA M' with a single starting state can be constructed from M such that $L(M') = L(M)$. This is done by adding an extra symbol σ not in the state set Q of M, and using it as the starting state of M'. Additionally, the transition function T' of M' extends the transition function T of M such that $T'(\sigma, a) = I$ for all $a \in \Sigma \cup \lambda$.

- Another possible generalization is to include the so-called `http://planetmath.org/EpsilonT` *transitions*. Nevertheless, it can be shown that any NDFA with ϵ-transitions is equivalent to one without any ϵ-transitions.