Finite Automata as Graphs, Part II
Non-Deterministic Finite Automata

Richard Cole
September 12, 2006

Non-deterministic Finite Automata, NFAs for short, are a generalization of the machines we have already defined, which are often called Deterministic Finite Automata by contrast, or DFAs for short. The reason for the name will become clear later.

As with a DFA, an NFA is simply a graph with edges labeled by single letters from the input alphabet $\Sigma$. There is one structural change.

For each vertex $v$ and each character $a \in \Sigma$, the number of edges exiting $v$ labeled with $a$ is unconstrained; it could be 0, 1, 2 or more edges.

This obliges us to redefine what an automaton $M$ is doing, given an input $x$. Quite simply, $M$ on input $x$ determines all the vertices that can be reached by paths starting at $q_0$ labeled $x$. If any of these reachable vertices is in the set of Recognized or Final vertices then $M$ is defined to accept $x$. Another way of looking at this is that $M$ recognizes $x$ exactly if there is some path (and possibly more than one) labeled $x$ from $q_0$ to a vertex (state) $y \in F$.

As we will see later, the collection of languages recognized by NFAs is exactly the collection of languages recognized by DFAs. That is, for each NFA $M$ there is a DFA $M_1$ with $L(M) = L(M_1)$. Thus NFAs do not provide additional computational power. However, they can be more compact and easier to understand, and as a result they are quite useful.

We need to redefine the destination (transition) functions $\delta$ for NFAs. Now $\delta$ takes as its arguments a set of vertices (states) $R \subseteq V$ and a character $a \in \sigma$ and produces as output another set of vertices (states) $S \subseteq V$: $\delta(R, a) = S$. The meaning is that the set of vertices (states) reachable from $R$ on reading $a$ is exactly the set of vertices (states) $S$.

$\delta^*$ is also redefined. $\delta^*(R, x) = S$ means that the set of vertices (states) $S$ are the possible destinations on reading $x$ on starting from a vertex (state) in $R$. So, in particular, $x$ is recognized by $M$, $x \in L(M)$ exactly if $\delta^*((q_0), x) \cap F \neq \emptyset$; i.e., when starting in state $q_0$ on reading $x$, at least one destination vertex (state) is an accepting or final vertex (state).

$\delta$ is often defined formally as follows: $\delta: 2^\Sigma \times \Sigma \rightarrow 2^V$. In this context, $2^V$ means the collection of all possible subsets of $V$, and so $\delta$ does exactly what it should. Its inputs are (i) a set of vertices (states), that is one of the elements of the collection $2^V$, and (ii) a character in $\Sigma$; its output is also a set of vertices (states).
Example. $A = \{w \mid$ the third to last character in $w$ is a 1 $\}$.

$L(M) = A$, where

\[ M : \]

\[ \begin{array}{c}
Q = \{ q_0, q_1, q_2, q_3, q_4 \} \\
\delta : Q \times \Sigma \rightarrow Q \\
\lambda : Q \rightarrow Q \\
\end{array} \]

Next, we add one more option to NFAs: edges labeled with the empty string:

\[ \begin{array}{c}
D \xrightarrow{\varepsilon} \end{array} \]

The meaning is that if vertex (state) $p$ is reached vertex (state) $q$ can also be reached without reading any more input. Again, the same languages as before are recognized by NFAs and DFAs; however, the $\varepsilon$-labeled edges are very convenient in creating understandable machines.

Example. Let $A$ and $B$ be languages over the alphabet $\Sigma$ that are accepted by NFAs $M_A$ and $M_B$, respectively. Then the following NFA $M$ accepts the language $A \cup B$.

\[ \begin{array}{c}
M_{\text{Shut}} \xrightarrow{\varepsilon} \\
M_{\text{AShtat \&}} \\
M_{\text{Shut \&}} \\
M_{\text{Shut}} \\
M_A \\
M_B \\
\end{array} \]

The first step, prior to reading any input, is to go to the start states for $M_A$ and $M_B$. Then the computation in these two machines is performed simultaneously. The set of final states for $M$ is the union of the final states for $M_A$ and $M_B$; thus $M$ reaches a final state on input $x$ exactly if at least one of $M_A$ or $M_B$ reaches a final state on input $x$. In other words $L(M) = L(M_A) \cup L(M_B)$.

**Deterministic vs Nondeterministic.** Another way of viewing the computation of an NFA $M$ on input $x$ is that the task is to find a path labeled $x$ from the start state to a final state if there is one, and this is done correctly by (inspired) guessing. This process of correct guessing is called *Nondeterministic* computation. Correspondingly, if there is no choice or uncertainty in the computation, it is said to be *Deterministic*. 

2