Nondeterministic Automata vs Deterministic Automata

We learned that NFA is a convenient model for showing the relationships among regular grammars, FA, and regular expressions, and designing them. However, we know that an NFA is a conceptual model that cannot directly be built because of the nondeterministic transition. Then what about all the NFA that appear in the examples and proofs? Are those nondeterministic automata remain as theoretical model that cannot bring down to the real world?

For context-free languages, there are languages that can only be recognized by NPDA, for example \( \{xx^R \mid x \in \{a, b\}^* \} \). As far as PDA are concerned, NPDA are strictly more powerful than DPDA. For LBA, it is open problem. (Looks like the space restriction is too much for a DLBA to do the same computation as an NLBA does.) For TM, any problem that can be solved by an NTM can also be solved by a DTM by tracing every possible transition of an NTM computation using its unlimited space available.
Fortunately, for NFA there is a straightforward way to transform them into DFA. (Actually it is based on the same idea that we used to eliminate $\varepsilon$-transitions.) The basic idea is to consider the set of states that can be reachable by a transition as a single state in deterministic transition. The following example will be enough to understand the technique. (We assume that the automaton has no $\varepsilon$-transitions.)

![NFA and DFA diagrams]

(a) An NFA

(b) Converted DFA

Notice that the state with label $\{0, 1, 2\}$ is from the set of states given by the nondeterministic transition $\delta(0, a) = \{0, 1, 2\}$. Also notice that any state whose label contains an accepting state is defined as an accepting state in the deterministic machine.
Minimization Technique for DFA

The number of states of an automaton has direct affect to the size of the machine realizing the automaton. Hence, it is very important to reduce the number of states, if possible. For PDA, LBA and TM, it is very difficult problem to reduce the number of states. However, for DFA there is very efficient algorithm for minimizing the number of states of a given DFA.

Figure (a) below is a part of the state transition graph of a DFA $M = (Q, \Sigma, \delta, q_0, F)$, where $\Sigma = \{a, b\}$. Clearly, for every $w \in \Sigma^*$, $\delta(q_3, w)$ is in an accepting state if and only if $\delta(q_4, w)$ is. Hence, we can merge $q_3$ and $q_4$ into a single state as shown in Figure (b) without affecting the language of the machine.
State Reduction by Partitioning

We say two states \( p \) and \( q \) are equivalent (or indistinguishable), if, for every string \( w \in \Sigma^* \), transition \( \delta(p, w) \) ends in an accepting state if and only if \( \delta(q, w) \) does. In the preceding slide states \( q_3 \) and \( q_4 \) are equivalent. There are efficient algorithms available for computing the sets of equivalent states of a given DFA.

The following example shows a procedure using the set partitioning technique. The technique is similar to one that they use for partitioning people into groups (each having certain preferences) based on their responses to questionnaire. The following two slides show the detailed steps for computing equivalent state sets of the DFA in Figure (a) and constructing the reduced DFA shown in Figure (b).
State Reduction by Partitioning (cont’ed)

• Step 0: Partition the states according to accepting/non-accepting.

\[ P_1 \quad P_2 \]
\[
\begin{array}{c}
\{ 3, 4, 5 \} \\
\{ 0, 1, 2 \}
\end{array}
\]

Figure (a) Initial partition

For a state \( q \) and symbol \( t \), let \( P_i \) be the response of \( q \) on \( t \), if \( \delta(q, t) \) enters a state in \( P_i \).

• Step 1: Get the response of each state for each input symbol. Notice that States 3 and 0 show different responses from the ones of the other states in the same set.

\[ P_1 \quad P_2 \]
\[
\begin{array}{ccc}
p_1 & p_1 & p_1 \\
a \rightarrow & \uparrow & \uparrow & \uparrow \\
\{3, 4, 5\} \\
b \rightarrow & \downarrow & \downarrow & \downarrow \\
p_2 & p_1 & p_1
\end{array}
\]

\[ P_2 \quad P_2 \]
\[
\begin{array}{ccc}
p_2 & p_1 & p_1 \\
a \rightarrow & \uparrow & \uparrow & \uparrow \\
\{0, 1, 2\} \\
b \rightarrow & \downarrow & \downarrow & \downarrow \\
p_2 & p_1 & p_1
\end{array}
\]

Figure (b) Record responses for each input symbol
• Step 2: Partition the sets according to the responses, and go to Step 1 until no partition occurs.

\[
\begin{array}{cccc}
P_{11} & P_{12} & P_{21} & P_{22} \\
p_{11} & p_{11} & p_{12} & p_{12} \\
{a} \rightarrow & ↑ & ↑ & a \rightarrow & ↑ & ↑ \\
\{4, 5\} & \{3\} & \{1, 2\} & \{0\} \\
b \rightarrow & ↓ & ↓ & b \rightarrow & ↓ & ↓ \\
p_{11} & p_{11} & p_{11} & p_{11} \\
\end{array}
\]

Figure (c) Partition the set, and record responses for each input symbol

No further partition is possible for the sets \(P_{11}\) and \(P_{21}\). So the final partition results are as follows.

\[
\begin{array}{cccc}
\{4, 5\} & \{3\} & \{1, 2\} & \{0\} \\
\end{array}
\]

(d) Final partition