Notes on Sipser: §1.2, “Nondeterminism”
CSC 341, “Automata, Formal Languages, and Computational Complexity”
Department of Computer Science - Grinnell College
August 31, 2020

In Figure 1.29, the arrows from $q_1$ states to $q_3$ are potentially misleading. They do not represent single transitions in the $N_1$ automaton, but double transitions during which only one symbol is read. (The first transition takes $N_1$ from $q_1$ to $q_2$ on input 1; the second takes it from $q_2$ to $q_3$ on input $\varepsilon$, that is, without reading a symbol.) If I were drawing the diagram, I’d leave out these double-transition arrows and instead put in horizontal arrows from $q_2$ to $q_3$. The direction of the arrows would indicate that no additional symbol was being read.

The bold arrows in Figures 1.28 and 1.29 show the paths leading from the start state to an accept state.

The transition function for a nondeterministic finite automaton can also be given as a collection of equations, specifying the function’s value for each possible combination of arguments. Here is such a specification for the transition function in $N_1$:

\[
\begin{align*}
\delta(q_1, 0) &= \{q_1\} \\
\delta(q_1, 1) &= \{q_1, q_2\} \\
\delta(q_1, \varepsilon) &= \emptyset \\
\delta(q_2, 0) &= \{q_3\} \\
\delta(q_2, 1) &= \emptyset \\
\delta(q_2, \varepsilon) &= \{q_3\} \\
\delta(q_3, 0) &= \emptyset \\
\delta(q_3, 1) &= \{q_4\} \\
\delta(q_3, \varepsilon) &= \emptyset \\
\delta(q_4, 0) &= \{q_4\} \\
\delta(q_4, 1) &= \{q_4\} \\
\delta(q_4, \varepsilon) &= \emptyset
\end{align*}
\]

In the definition of ‘accepts’ for nondeterministic finite automata (page 54), note that, in the expansion $w = y_1 y_2 \ldots y_m$, some or all of the $y$ components can be $\varepsilon$. Thus $w$ may contain fewer than $m$ symbols; when we write it out using the $y$ components, we can insert $\varepsilon$ anywhere we like without affecting the actual symbols in $w$ (since $\varepsilon$ is not itself a symbol, but an empty sequence of symbols).

Until we get to the proof of Theorem 1.39, we have not really used the idea that a state inside a finite automaton might have a hidden internal structure that makes the automaton easier to describe and construct.

Sipser hints at that possibility in the state diagram in Figure 1.32, where the subscripts on the state labels show that each state is keeping track of the three most
recently encountered input symbols. To some readers, that subscripting convention suggests a possible hardware implementation of the memory unity for that automaton using three two-position switches. Formally, the states could identified 3-tuples of switch positions.

Definitions 1.5 and 1.37 say nothing whatever about the nature of states, so a state in a deterministic finite automaton can be literally anything, including (as in the proof of Theorem 1.39) a set of states of a different automaton.

Some practical-minded readers of the proof of Theorem 1.39 are discouraged to learn that the construction used in the proof turns a \( k \)-state nondeterministic finite automaton into a \( 2^k \)-state deterministic finite automaton. This implies that the construction is infeasible even for modest values of \( k \) (\( k = 128 \), say).

However, real-world resource constraints do not affect the soundness of a mathematical proof, even a proof by construction. The theorem to be proven asserts the existence of an abstract entity (a deterministic finite automaton) that has a specified property (recognizing the same language as a given nondeterministic finite automaton). To put it another way: The proof of the theorem depends on the correctness of the algorithm for constructing the deterministic finite automaton, not on the efficiency or real-world feasibility of the application of that algorithm to any particular case.

The proof of Theorem 1.39 holds even in cases in which \( k \) itself is so large that even the initial nondeterministic finite automaton could not be constructed in the real world (\( k = 2^{256} \), say). A finite automaton can have any finite number of states; there is no upper bound.

The construction procedure that Sipser describes would not be used in an application program anyway, since in nearly all cases the deterministic finite automaton it produces contains many states that are unreachable (no sequence of transitions beginning from the start state ends with them).

For instance, if you apply that construction to the automaton \( N_1 \) (Figure 1.27, page 48), you get a deterministic finite automaton with sixteen states, one for each subset of the set of states of \( N_1 \). However, only six of these states (the ones corresponding to \( \{q_1\}, \{q_1, q_2, q_3\}, \{q_1, q_2, q_3, q_4\}, \{q_1, q_3\}, \{q_1, q_3, q_4\}, \) and \( \{q_1, q_4\} \)) are reachable, and with a little more optimization work you can get to a four-state deterministic finite automaton that recognizes the same language as \( N_1 \), though in a less obvious way:

![Diagram](image)

Sipser says, in the second paragraph of § 1.2, that “every deterministic finite automaton is automatically a nondeterministic finite automaton,” and he relies on this
premiss in the proof of Corollary 1.40. Formally speaking, however, it’s not quite true. Once again, a stickler for mathematical rigor would want to prove a small lemma to show that, for every deterministic finite automaton $M = (Q, \Sigma, \delta, q_0, F)$, there is a nondeterministic finite automaton $M'$ that recognizes the same language as $M$.

It’s a straightforward proof by construction: Let $M' = (Q, \Sigma, \delta', q_0, F)$, where

$$\delta'(q, a) = \begin{cases} \{\delta(q, a)\} & \text{if } a \in \Sigma, \\ \emptyset & \text{if } a = \varepsilon. \end{cases}$$

Now every transition that $M$ makes in processing some input string $w$ is matched by a transition in $M'$ and vice versa, so that $M'$ accepts $w$ if, and only if, $M$ accepts $w$ (since they have the same accept states).

Theorems 1.45 and 1.47 both presuppose that the nondeterministic finite automata that are being combined share an alphabet, so technically they depend on an alphabet extension lemma for nondeterministic finite automata. Fortunately, the nondeterministic version of this lemma is even easier than the deterministic one, since instead of adding the new state $q_{\text{dead}}$ to accommodate strings containing previously unrecognized symbols, the nondeterministic transition function can just yield $\emptyset$ in such cases.

There is a somewhat similar problem about the states of the automata that the constructions combine in Theorems 1.45 and 1.47: In each case, the proof assumes that $Q_1$ and $Q_2$ are disjoint. Here, though, it would be completely appropriate to introduce a “without loss of generality” clause to justify this assumption: We can always simply remap the states of $N_2$ an equal number of distinct values that are known not to be states of $N_1$, as a preliminary to the construction. (It would be possible to prove a “state disjointness lemma” showing that the automaton resulting from this remapping recognizes the same language as the original $N_2$ automaton, but I won’t do that here.)