DFAs and NFAs

Max Randall
Chapman University

March 8, 2025

Contents

1 Introduction

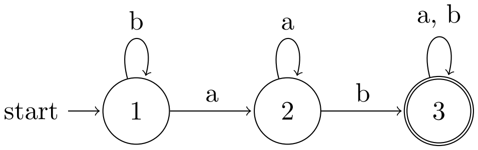
In this report, we explore fundamental aspects of both deterministic and nondeterministic finite automata (DFAs and NFAs), including extended transition functions, product automaton construction for intersection, and state modifications for union. We further demonstrate the subset construction to establish the equivalence of NFAs and DFAs, showcasing the broad utility of these automata concepts in both theoretical and practical contexts.

2 Exersizes

2.1 Homework 1

Let $\mathcal{A} = (Q, \Sigma, \delta : Q \times \Sigma \to Q, q_0, F)$ be a DFA. Explain in what way you can view $\mathcal{A}$ as an NFA by doing the following:

1. Let $\mathcal{A}$ denote the following DFA:



   Here:

$$\begin{aligned} Q = \{1,2,3\}, \quad \Sigma = \{a,b\}, \\ q_0 = 1, \quad F = \{3\}, \\ \delta(1,a) = 2,\ \delta(1,b) = 1, \\ \delta(2,a) = 2,\ \delta(2,b) = 3, \\ \delta(3,a) = 3,\ \delta(3,b) = 3. \end{aligned}$$

   Explain how you can understand $\mathcal{A}$ also as an NFA.

2. More generally, let $\mathcal{A} = (Q, \Sigma, \delta : Q \times \Sigma \to Q, q_0, F)$ be a DFA. Define an NFA

$$\mathcal{A}' = (Q', \Sigma, \delta' : Q' \times \Sigma \to \mathcal{P}(Q'), q_0', F')$$

   such that $L(\mathcal{A}) = L(\mathcal{A}')$.

3. Justify why your construction satisfies the desired condition.

Solutions

1) Viewing the Example DFA as an NFA

A deterministic finite automaton (DFA) can be seen as a special case of a nondeterministic finite automaton (NFA) by interpreting its transition function in the following way: in an NFA, the transition function

$$\delta' : Q \times \Sigma \to \mathcal{P}(Q)$$

produces *sets* of possible next states. In a DFA, however, for each state q and input symbol a, there is exactly one next state $\delta(q,a)$. To view the DFA as an NFA, we simply set:

$$\delta'(q,a) \;=\; \{\delta(q,a)\}.$$

Hence, each transition in the original DFA becomes a transition to a *singleton set* in the NFA. This preserves the language recognized, because in the NFA there is exactly one possible way to move from one state to another on a given symbol (i.e., no extra nondeterminism is introduced).

Concretely, for the illustrated DFA with states $1,2,3$, we define the NFA with the same states, same start state, same accepting states, and

$$\delta'(q,a) = \{\delta(q,a)\} \quad \text{and} \quad \delta'(q,b) = \{\delta(q,b)\}.$$

2) General Construction from a DFA to an NFA

Given any DFA

$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F),$$

we can define an NFA

$$\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$$

as follows:

- $Q' = Q$ (we use exactly the same set of states),
- $q_0' = q_0$ (same start state),
- $F' = F$ (same set of accepting states),
- For each $q \in Q$ and $a \in \Sigma$, set $\delta'(q, a) = \{\delta(q, a)\}$.

Thus,

$$\delta' : Q' \times \Sigma \;\to\; \mathcal{P}(Q'), \quad \delta'(q, a) = \big\{\delta(q, a)\big\}.$$

3) Why $L(\mathcal{A}) = L(\mathcal{A}')$?

The language recognized by a DFA is determined by the unique run from q_0 on any input string. In the constructed NFA, there is still exactly one possible transition at each step (the singleton set). So:

- If a string w is accepted by the original DFA $\mathcal{A}$, then following the same path in $\mathcal{A}'$ is not only possible but is in fact the *only* path. Thus w is also accepted by $\mathcal{A}'$.
- If a string w is not accepted by the original DFA, then there is no way to reach an accepting state via δ; hence in the NFA δ', which mimics these transitions in singleton sets, there is equally no path that leads to an accepting state.

Therefore, the two automata accept exactly the same set of strings, i.e., $L(\mathcal{A}) = L(\mathcal{A}')$.

2.2 Homework 2

1. Describe in words the language $L(A)$ accepted by the NFA $\mathcal{A}$ pictured below.
2. Specify the automaton $\mathcal{A}$ formally in the tuple form

$$(Q, \Sigma, \delta, q_0, F).$$

3. Using the extended transition function $\hat{\delta}$ of $\mathcal{A}$, compute the set of states

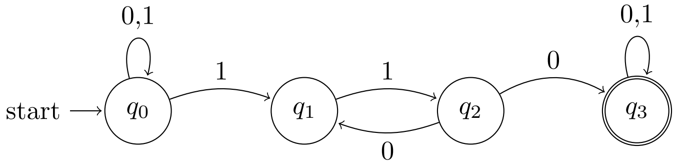
$$\hat{\delta}(q_0,\, 10110)$$

step by step.

4. Find *all* paths in $\mathcal{A}$ for the words $v = 1100$ and $w = 1010$. Represent each set of paths in a common diagram.
5. Construct the determinization $\mathcal{A}^D$ (the "power set automaton") of $\mathcal{A}$.
6. Verify that $L(\mathcal{A}) = L(\mathcal{A}^D)$. Is there a smaller DFA that accepts the same language?

Solutions

Below is the NFA $\mathcal{A}$:



1) Language Description

Intuitively, the automaton $\mathcal{A}$ accepts exactly those binary strings for which there is a path from q_0 to the accepting state q_3.

- A necessary part of reaching q_3 is taking the transition $q_2 \xrightarrow{0} q_3$.
- To get to q_2 in the first place, we must have used $q_1 \xrightarrow{1} q_2$.
- And to get to q_1, we must have used $q_0 \xrightarrow{1} q_1$ on some occurrence of the symbol 1 (rather than looping back to q_0 on that same 1).

Hence there must be at least two consecutive 1's used *as the path* $q_0 \to q_1 \to q_2$, and then at least one 0 from q_2 to q_3. Because q_0 loops on $\{0, 1\}$, those symbols can appear arbitrarily often before the transition $q_0 \to q_1$ is finally taken on a 1. And from q_2, we can read a 0 that returns us to q_1 (followed by a 1 that goes back to q_2), repeatedly. Eventually, a 0 from q_2 must go to q_3, after which any further symbols are accepted by the loop in q_3.

In simpler terms: $\mathcal{A}$ accepts all binary strings that, *on some nondeterministic choice of transitions*, contain two consecutive 1's used specifically to go from q_0 to q_1 to q_2, and eventually at least one 0 from q_2 to q_3. After that point, the rest of the string can be anything.

2) Formal NFA Specification

We specify $\mathcal{A}$ as:

$$\mathcal{A} \;=\; (\,Q,\; \Sigma,\; \delta,\; q_0,\; F\,),$$

where:

$$\begin{aligned}
Q &= \{\,q_0, q_1, q_2, q_3\},\\
\Sigma &= \{\,0, 1\},\\
q_0 &= q_0 \text{ (the start state)},\\
F &= \{\,q_3\},\\
\delta(q_0, 0) &= \{\,q_0\}, \quad \delta(q_0, 1) \;=\; \{\,q_0,\; q_1\},\\
\delta(q_1, 0) &= \varnothing, \quad \delta(q_1, 1) \;=\; \{\,q_2\},\\
\delta(q_2, 0) &= \{\,q_1,\; q_3\}, \quad \delta(q_2, 1) \;=\; \varnothing,\\
\delta(q_3, 0) &= \{\,q_3\}, \quad \delta(q_3, 1) \;=\; \{\,q_3\}.
\end{aligned}$$

3) Extended Transition Function on 10110

Recall that the extended transition function $\hat{\delta}(r, x)$ for a state r and input string x returns *all* possible states reachable from r by reading x. We compute

$$\hat{\delta}(q_0,\ 10110)$$

symbol by symbol:

$$\begin{aligned}
\hat{\delta}(q_0, \epsilon) &= \{\, q_0\}, \\
\hat{\delta}(q_0, 1) &= \delta(q_0, 1) = \{\, q_0, q_1\}, \\
\hat{\delta}(\{q_0, q_1\}, 0) &= \delta(q_0, 0)\ \cup\ \delta(q_1, 0) = \{\, q_0\}\ \cup\ \varnothing = \{\, q_0\}, \\
\hat{\delta}(\{q_0\}, 1) &= \delta(q_0, 1) = \{\, q_0, q_1\}, \\
\hat{\delta}(\{q_0, q_1\}, 1) &= \delta(q_0, 1)\ \cup\ \delta(q_1, 1) = \{\, q_0, q_1\}\ \cup\ \{\, q_2\} = \{\, q_0, q_1, q_2\}.
\end{aligned}$$

Thus after reading `10110`, the set of possible states is $\{\, q_0, q_1, q_2\}$. (Note: if we continued with another symbol, we would then look at transitions from each state in $\{q_0, q_1, q_2\}$ on that symbol.)

4) All Paths for `v=1100` and `w=1010`

All paths for `1100`. Starting in q_0 and reading `1100` step by step:

From q_0 on first '1':

$q_0 \overset{1}{\to} q_0 \quad \text{or} \quad q_0 \overset{1}{\to} q_1.$

Then on second '1':

if the first transition stayed in q_0, we again have $q_0 \overset{1}{\to} \{q_0, q_1\}$,

if the first transition went to q_1, we must go $q_1 \overset{1}{\to} q_2$.

Then on first '0':

$\ldots$

One can systematically draw out the branching paths. Eventually, any path that leads to q_3 must include the sub-path

$$q_0 \overset{1}{\to} q_1 \overset{1}{\to} q_2 \overset{0}{\to} q_3$$

at some stage.

All paths for `1010`. A similar analysis shows that from q_0 on '1' we can stay in q_0 or move to q_1. Then reading '0', etc. You can depict these paths in a common diagram with branches for each nondeterministic choice.

5) Determinization via Power Set Construction

Define $\mathcal{A}^D = (Q^D,\, \Sigma,\, \delta^D,\, q_0^D,\, F^D)$ by:

$$\begin{aligned} Q^D &= \mathcal{P}(Q) = \{\varnothing,\ \{q_0\},\ \{q_1\},\ \dots,\ \{q_0, q_1\},\ \dots,\ \{q_0, q_1, q_2, q_3\}, \dots\}, \\ q_0^D &= \{q_0\}, \\ F^D &= \{\, S \subseteq Q : S \cap \{q_3\} \neq \varnothing\}, \\ \delta^D(S, a) &= \bigcup_{s \in S} \delta(s, a). \end{aligned}$$

Starting from $\{q_0\}$, compute systematically $\delta^D(\{q_0\}, 0)$, $\delta^D(\{q_0\}, 1)$, and so forth. You will obtain a transition table with subsets of Q as states, and those subsets containing q_3 are accepting.

6) Verification & Minimization

- By construction, $\mathcal{A}^D$ accepts exactly the same language as $\mathcal{A}$, i.e. $L(\mathcal{A}^D) = L(\mathcal{A})$.
- Often, the power-set DFA can be minimized further. One can apply the usual DFA minimization algorithm (Myhill-Nerode or partition-refinement). In general, yes, there will be a smaller DFA than $\mathcal{A}^D$ that still accepts the same language.

3 Chapter 3

Chapter 3 introduces regular expressions, a notation for describing languages. They define exactly the same languages as nondeterministic finite automata (NFAs) but can be more user-friendly. This chapter details union, concatenation, and Kleene star operations, common in text search tools and lexical analyzers. Regular expressions share the same expressive power as NFAs, DFAs, and ϵ-NFAs, collectively forming the class of regular languages. They obey algebraic laws like arithmetic, but with important differences. We cover closure, which repeatedly concatenates elements of a language, often producing infinite sets. Examples illustrate how regular expressions describe patterns like strings with at least one special symbol or alternating bits. The chapter explains the equivalence between DFAs and regular expressions: any regular expression can be converted to an NFA (then to a DFA), and any DFA can be transformed into a regular expression via state elimination. This fundamental equivalence underpins tools like `grep`, `lex`, and compilers, using regular expressions to specify tokens or locate patterns in textual input.

4 Conclusion

Throughout these problems and discussions, we expanded our grasp of the fundamental principles behind both deterministic and nondeterministic finite automata. We learned to use extended transition functions to formalize how automata parse and accept entire strings. We also constructed product automata to find the intersection of two languages and modified acceptance states to find unions. Most prominently, we employed the subset (power set) construction, which provides a systematic method for converting any NFA into a DFA that recognizes exactly the same language. Taken together, these core concepts highlight the depth and versatility of finite automata theory—critical not only in classical computability and formal language theory but also in practical applications such as lexical analysis, pattern matching, and software validation.

Interesting question: Efficiency of regular expressions.
How do regular expressions compare in efficiency to finite automata when processing large-scale pattern-matching tasks, such as those in search engines or language analyzers?

References

[HMU] . E. Hopcroft, R. Motwani, J. D. Ullman: *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*, Archive.org Link.