Decidability Boundaries in Finite Automata: The Equivalence Problem for DFAs and NFAs

Maisha Rahman Chowdhury
Department of Computer Science
Iowa State University

October 2, 2025

Abstract

This paper is an expository exploration of the *equivalence problem* for finite automata. Deterministic finite automata (DFAs) and nondeterministic finite automata (NFAs) recognize the same class of languages—the regular languages—but the complexity of deciding equivalence differs sharply between them. For DFAs, equivalence is solvable efficiently in polynomial time. For NFAs, the problem remains decidable but is PSPACE-complete. We review the relevant definitions, outline the classical algorithms and complexity results, and provide a worked example to illustrate the gap between the two models.

1 Introduction

This paper is written as an expository study rather than as new research. Our goal is to present the equivalence problem clearly, explain why it is easy for DFAs and hard for NFAs, and illustrate these ideas with examples. In doing so, we aim to highlight how nondeterminism preserves decidability while increasing computational complexity.

The study of decidability is central to theoretical computer science. A decision problem asks a yes-or-no question about an object, such as whether two automata accept the same language. A problem is decidable if there exists an algorithm that always produces the correct answer in finite time.

Finite automata provide a clean setting to study decidability. They are simple computational models, yet their decision problems reveal deep boundaries in computational complexity. Importantly, deterministic finite automata (DFAs) and nondeterministic finite automata (NFAs) are equivalent in expressive power: both recognize exactly the class of regular languages. However, this equivalence does not extend to the complexity of reasoning about them.

My own interest in this topic grew from coursework at Iowa State University. In Fall 2023 I completed *Discrete Mathematics*, followed by *Introduction to the Design and Analysis of Algorithms* the semester after. Currently I am taking *Theory of Computing*, where I was first introduced to formal models like DFAs and NFAs. I found that I especially enjoy drawing automata, working through equivalence problems, and reasoning about their behavior. This project grew naturally out of that experience, and it reflects how these classes inspired me to explore the equivalence problem more deeply.

This paper focuses on the **equivalence problem**: given two automata A_1 and A_2, decide whether $L(A_1) = L(A_2)$. We show that for DFAs, the problem is solvable in polynomial time, but for NFAs it is PSPACE-complete.

2 Background

2.1 Deterministic and Nondeterministic Finite Automata

A **deterministic finite automaton (DFA)** is a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

where Q is a finite set of states, Σ is the input alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the start state, and $F \subseteq Q$ is the set of accepting states.

A **nondeterministic finite automaton (NFA)** is defined similarly, except that $\delta : Q \times \Sigma \to 2^Q$, mapping to a set of possible next states. NFAs may also include ϵ-transitions. Both DFAs and NFAs recognize the regular languages.

2.2 Decidability and Complexity

A problem is **decidable** if there exists an algorithm to solve it on all inputs. Decidability, however, does not imply efficiency. Some problems are solvable in **polynomial time (P)**, while others may require **polynomial space (PSPACE)** or more. A problem is **PSPACE-complete** if it is as hard as the hardest problems solvable in polynomial space.

3 The Equivalence Problem

3.1 DFA Equivalence

For DFAs, equivalence is decidable in polynomial time.

Algorithm (Product construction):

1. Construct the product DFA that accepts the symmetric difference $L(A_1) \triangle L(A_2)$.
2. Test whether this DFA's language is empty.
3. If empty, the DFAs are equivalent; otherwise, they differ.

Emptiness test: Perform reachability analysis from the start state; if no accepting state is reachable, the language is empty. This takes $O(|Q| + |E|)$ time.

Alternative algorithm: Minimize both DFAs using Hopcroft's algorithm ($O(n \log n)$) and check isomorphism.

Thus DFA equivalence is decidable in polynomial time.

3.2 NFA Equivalence

For NFAs, equivalence is still decidable but **PSPACE-complete**.

One approach: Convert each NFA into a DFA via subset construction, then apply the DFA algorithm. But subset construction can cause an exponential blowup: an NFA with n states can yield a DFA with up to 2^n states.

Meyer and Stockmeyer (1972) proved that NFA equivalence is PSPACE-complete, using reductions from problems such as regular expression equivalence.

Thus while decidable, NFA equivalence is computationally intractable in the general case.

Worked example: NFA → DFA by subset construction

NFA definition. Consider the NFA $N = (Q, \Sigma, \delta, q_0, F)$ with

$$Q = \{0, 1, 2\}, \qquad \Sigma = \{a, b\}, \qquad q_0 = 0, \qquad F = \{2\},$$

and transition function δ given by

$$\begin{aligned} \delta(0, a) &= \{0, 1\} \\ \delta(0, b) &= \{0\} \\ \delta(1, a) &= \varnothing \\ \delta(1, b) &= \{2\} \\ \delta(2, a) &= \varnothing \\ \delta(2, b) &= \varnothing. \end{aligned}$$

Goal. Construct an equivalent DFA D using the subset construction, and identify its reachable states and accepting states.

Subset construction (reachable subsets). Start state of the DFA is the ϵ-closure of $\{0\}$ (there are no ϵ-moves here), so start $S_0 = \{0\}$. Compute transitions on a and b from each reachable subset:

DFA state (subset)	on a	on b
$\{0\}$	$\delta(\{0\}, a) = \{0, 1\}$	$\delta(\{0\}, b) = \{0\}$
$\{0, 1\}$	$\delta(\{0, 1\}, a) = \{0, 1\}$	$\delta(\{0, 1\}, b) = \{0, 2\}$
$\{0, 2\}$	$\delta(\{0, 2\}, a) = \{0, 1\}$	$\delta(\{0, 2\}, b) = \{0\}$

No other subsets are reachable from the start. Thus the reachable DFA states are exactly

$$\{0\}, \quad \{0, 1\}, \quad \{0, 2\}.$$

A DFA subset is accepting iff it contains the NFA accepting state 2. Therefore the only accepting DFA state is $\{0, 2\}$.

Resulting DFA (informal description). The DFA D has three states (the subsets above), start state $\{0\}$, accepting state $\{0, 2\}$, and transitions described in the table. By construction D accepts precisely the same language as N.

Why this demonstrates equivalence but not blowup. This example shows the subset construction in action: every NFA is convertible to an equivalent DFA whose states are subsets of the NFA states. Here the conversion produced only three reachable DFA states, so no exponential blowup occurred. However, the construction guarantees that the number of *possible* DFA states is at most $2^{|Q|}$, and there exist standard families of NFAs (see Hopcroft & Ullman, Sipser) for which many or all of those subsets are reachable. Those families demonstrate the *worst-case* exponential blowup that underlies the PSPACE hardness results for some decision problems on NFAs.

4 Discussion

Although DFAs and NFAs recognize the same class of languages, the complexity of the equivalence problem differs sharply. DFAs, with their deterministic structure, allow efficient polynomial-time algorithms. NFAs, with nondeterminism and ϵ-transitions, cause exponential growth in the state space, pushing the problem into PSPACE.

This boundary illustrates an important theme in theoretical computer science: expressiveness may remain the same, but the cost of reasoning about a model can change drastically. These results also explain why DFA-based methods are commonly used in compilers and model checking, where efficient analysis is essential.

5 Conclusion and Future Work

This expository exploration shows how a seemingly small change in machine model—from determinism to nondeterminism—does not affect expressive power but dramatically alters the complexity of analysis. We studied the equivalence problem for finite automata. For DFAs, it is solvable in polynomial time; for NFAs, it remains decidable but is PSPACE-complete. This reveals a boundary where nondeterminism makes analysis significantly harder without changing recognition power.

Future work could explore related problems, such as NFA minimization (also PSPACE-hard), or equivalence of regular expressions with additional operators. Another direction is to connect these theoretical results with practical applications in verification and language processing, where automata play a central role.

References

- Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation*. Pearson.
- Sipser, M. (2012). *Introduction to the Theory of Computation* (3rd ed.). Cengage Learning.
- Meyer, A. R., & Stockmeyer, L. J. (1972). The equivalence problem for regular expressions with squaring requires exponential space. *13th Annual Symposium on Switching and Automata Theory*.
- Kozen, D. C. (1977). Lower bounds for natural proof systems. *Foundations of Computer Science (FOCS)*.