# Finite Automata and Their Decision Problems

Summary of the paper by Michael O. Rabin and Dana Scott

Abhilasha Sharma Suman

## 1 Introduction

In 1959, Michael O. Rabin and Dana Scott published a pionerring paper titled *Finite Automata and Their Decision Problems*, which laid the foundation for Automata theory. This introduced a formal framework for understanding finite automata, deterministic and non-deterministic automata (shout out to CS228 coming up soon), and tackled the problem of decision-making in the context of regular languages.

## 2 Finite Automata

The paper begins by formalizing *finite automata* (FA), which are simple models consisting of a finite set of states, inputs, and transition functions that determine how the machine changes states upon reading the input. A finite automaton accepts or rejects an input depending on whether it finally ends up in an accepting (desirable) state or not.

The authors introduce two types of finite automata:

- **Deterministic Finite Automata (DFA)**: In a DFA, for every state and input symbol, there is exactly one transition to a new state. A DFA is formally defined as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:
  - $Q$: Finite set of states
  - $\Sigma$: Input
  - $\delta$: Transition function, $\delta : Q \times \Sigma \to Q$
  - $q_0$: Initial state
  - $F$: Set of accepting states, $F \subseteq Q$
- **Non-deterministic Finite Automata (NFA)**: An NFA allows for multiple possible transitions from a given state for the same input, including transitions without consuming any input (epsilon transitions or empty input). NFAs are also represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, but $\delta$ in this case maps to a set of states, $\delta : Q \times \Sigma \to 2^Q$ rather than a single state $Q$.

R & S (the authors) showed that these two models of computation are equivalent in their expressive power implying that we can convert any given NFA to a DFA and vice versa. But there is a catch- in going from NFA to DFA, we might explode (like DNF -¿ CNF explosion) to an infinite set of states.

# 3 Regular Languages

The paper focused on regular languages, which are the class of languages that can be represented by equivalent finite automata.

They provided formal methods to describe regular languages using finite automata and regular expressions. They also explored decision problems(somewhat detailed discussion coming up) associated with these languages, including the question of whether a given automaton accepts any strings at all (emptiness), whether two automata recognize the same language (equivalence), and whether one automaton's language is contained within another's.

## 3.1 The Formal Methods of Conversion from Language to Automata and more!

1. 1. Conversion from Regular Expressions to NFAs

   - **Concatenate:** For two regular expressions $R_1$ and $R_2$, the NFA for $R_1 \cdot R_2$ is constructed by connecting the accept state of the NFA for $R_1$ to the start state of the NFA for $R_2$ with an epsilon transition.
   - **Union:** For a regular expression $R_1 + R_2$, an NFA can be created by introducing a new start state that has epsilon transitions to the start states of the NFAs for $R_1$ and $R_2$.
   - **Kleene Star:** For a regular expression $R^*$, the NFA is built by creating a new start state with an epsilon transition to the start state of the NFA for $R$ and an epsilon transition from the accepting state back to the new start state.

2. NFAs to DFAs Conversion

   - **Subset Construction:** This algorithm creates states in the DFA that correspond to subsets of states in the NFA. For every state in the NFA, the DFA will represent all possible states that could be reached through non-deterministic transitions.
     - The start state of the DFA corresponds to the epsilon closure of the start state of the NFA.
     - For each input symbol, the transitions from each subset of states in the NFA are calculated to form the transitions of the DFA.

3. Finding the minimal DFA

   - **State Equivalence:** States in the DFA can be considered equivalent if they lead to the same set of accepting states for every input sequence.
   - **Partitioning:** The algorithm iteratively refines partitions of states based on their transitions until no further refinements can be made.

4. The closure properties of regular languages under union, intersection, and complementation, allowing the construction of automata that languages obtained as results of operations on other languages.

# 4 Decision Problems

This is the heart of the paper and it focusses on the identification and solution of decision problems related to finite automata and regular languages. The authors proved that the following problems are decidable:

1. **Membership Problem:** Given a finite automaton $A$ and a string $w$, is $w$ accepted by $A$? This problem is of **O(w)** complexity.

2. **Emptiness Problem:** Given a finite automaton $A$, is the language represented by $A$ empty? The paper shows that this can be checked by the reachability of any accepting states in $A$; if there are none then the language represented by $A$ is empty.

3. **Equivalence Problem:** Given two finite automata $A_1$ and $A_2$, do they recognize the same language? This problem is solved by constructing the symmetric difference of the two languages and checking for emptiness. (Symmetric difference of two automata is the input accepted by exactly one of the automata out of $A_1$ and $A_2$)

4. **Containment Problem:** Given two automata $A_1$ and $A_2$, is the language of $A_1$ a subset of that of $A_2$? This can be morphed into the emptiness problem for the intersection of $A_1$ with the complement of $A_2$.

The decision problems here have wide-ranging applications in the analysis of formal languages, compilers, and the theory of computation. These problems also laid the groundwork for understanding the broader computational complexity of decision procedures.

# 5 Nondeterminism and DFAs

One of the major contributions of the paper is the proof that nondeterministic finite automata (NFA) have the exact same expressibility power as deterministic finite automata (DFA) in terms of the languages they can recognize, which on a personal note, was very difficult idea to stomach. The key idea is that any NFA can be transformed into an equivalent DFA, though the resulting DFA may have exponentially more states. This transformation, known as the "subset construction," constructs the DFA by considering each state as a subset of NFA states.

While nondeterminism does not increase the computational power that is it cannot express anything above and beyond finite automata, it certainly does give a simpler representation of automata.

# 6 Impact of the Paper

The fact that this paper was written in 1959 and still as a beginner in 2024, this is highly recommended speaks to it's impact more than any words that I write in this report can. By formalizing the concept of finite automata and their decision problems, they established a rigorous mathematical framework that serves as the foundation for several areas in theoretical computer science. Their work paved the way for subsequent research on context-free languages, pushdown automata, and the Chomsky hierarchy(which I know absolutely nothing about, I just read this somewhere and felt like including it).

Later, very well deservedly, the authors were awarded the Turing Award for their work. The concepts introduced in this paper remain fundamental in the study of algorithms, compilers, and verification systems today.

# References

[1] M. O. Rabin and D. Scott, *Finite Automata and Their Decision Problems*, IBM Journal of Research and Development, 1959.