Thesis

Ayden Lamparski Howard Straubing

November 2, 2025

Chapter 1

Finite Deterministic Automata

Definition 1 (Deterministic Finite Automaton (DFA))**.** Mathlib provides a general definition of deterministic finite automata that does not require the state space or alphabet to be finite or have decidable equality. A `DFA` α σ consists of:

- `step :` $\sigma \to \alpha \to \sigma$ - a transition function that maps a state and input symbol to a new state
- `start :` σ - an initial state
- `accept : Set` σ - a set of accepting states

The `DFA` structure provides methods such as:

- `eval : List` $\alpha \to \sigma$ - evaluates a word from the start state
- `evalFrom :` $\sigma \to$ `List` $\alpha \to \sigma$ - evaluates a word from a given state
- `accepts : Set (List` α`)` - the language accepted by the automaton

Decidable equality means that for any two elements of a type, we can computationally determine whether they are equal or not. This is essential for implementing algorithms that need to compare states or symbols.

Definition 2 (Computable Finite DFA)**.** We define `FinDFA` α σ as a computable version of `DFA` α σ that enables algorithmic manipulation. A `FinDFA` differs from a `DFA` in several key ways:

- It requires `Fintype` instances on both the alphabet α and state space σ. A `Fintype` is a type that has finitely many elements and provides a way to enumerate all of them.
- It requires `DecidableEq` instances on both types, enabling computational equality testing.
- The accepting states are represented as a `Finset` σ rather than a `Set` σ. A `Finset` is a finite set that can be computationally manipulated, unlike the more general `Set` which may be infinite or non-computable.

This structure allows for a decidable procedure to determine if a state is accepting - we can simply check membership in the finite set of accepting states. We provide a coercion from `FinDFA` to `DFA`, allowing us to use all the existing `DFA` definitions for evaluation and language acceptance.

Definition 3 (Accessible States and Accessible DFA)**.** A state s in a `FinDFA` is called accessible if there exists some word w that reaches s from the start state. Formally, `FinDFA.IsAccessibleState M s` holds when there exists a word w such that evaluating w from the start state of M results in state s.

An `AccessibleFinDFA` is a structure that extends `FinDFA` with the additional requirement that every state in the automaton is accessible from the start state. This ensures that the automaton contains no "dead" or unreachable states.

Lemma 4 (Short Access Words and Decidable Accessibility)**.** *A fundamental result for implementing accessibility checking is that if a state is accessible by any word, then it is accessible by some word of length at most the number of states in the automaton. This bound follows from the pigeonhole principle: if a longer word exists, it must revisit some state, creating a loop that can be removed.*

This theorem enables us to create a decidable procedure for determining state accessibility. Instead of searching the infinite space of all possible words, we only need to check words up to a finite length bound. Using the `getWordsLeqLength` function, we can enumerate all words of bounded length and test each one.

Furthermore, this allows us to implement a language-preserving conversion from any `FinDFA` to an `AccessibleFinDFA` by restricting the state space to only the accessible states. The resulting automaton accepts exactly the same language as the original.

Proof. The proof uses strong induction on the length of the access word. If the word length is already within the bound (at most the number of states), we are done. Otherwise, the word must be longer than the number of states, so by the pigeonhole principle, some state must be visited twice during the evaluation.

Using Mathlib's `DFA.evalFrom_split` lemma, we can decompose the long word into three parts: a prefix leading to the first occurrence of the repeated state, a middle section that forms a loop returning to the same state, and a suffix continuing from there to the final state. By removing the loop (middle section), we obtain a shorter word that still reaches the same final state.

We can then apply the induction hypothesis to this shorter word, eventually obtaining a word within the desired length bound. The decidability instance follows by checking membership in the finite set of states reachable by bounded-length words. □