Thesis

Ayden Lamparski Howard Straubing

November 3, 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)**.**

- 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. □

Chapter 2

DFA Morphisms and Partial Order on AccessibleFinDFAs

Definition 5 (DFA Morphism)**.**
A *morphism* between DFAs $M : \text{DFA}(\alpha, \sigma_1)$ and $N : \text{DFA}(\alpha, \sigma_2)$, denoted $M \to_\ell N$, is a function $f : \sigma_1 \to \sigma_2$ that preserves:

- The start state: $f(M.\text{start}) = N.\text{start}$
- The accepting states: $q \in M.\text{accept} \iff f(q) \in N.\text{accept}$
- State transitions: $f(M.\text{evalFrom}(q, w)) = N.\text{evalFrom}(f(q), w)$ for all states q and words w

An *equivalence* of DFAs, denoted $M \simeq_\ell N$, is a bijective morphism with an inverse morphism.

Lemma 6 (Morphism Preserves Language)**.**
If there exists a morphism $f : M \to_\ell N$, then $M.accepts = N.accepts$.

Definition 7 (Partial Order on AccessibleFinDFAs)**.**
We define a partial order on `AccessibleFinDFA`s by the existence of surjective morphisms between their underlying DFAs.
A *surjective morphism* $M \twoheadrightarrow N$ between accessible finite DFAs is a morphism on their underlying DFAs that is surjective on states.
We define $M \leq N$ if and only if there exists a surjective morphism $N \twoheadrightarrow M$.
This relation satisfies:

- **Reflexivity**: $M \leq M$ for all M
- **Transitivity**: If $M \leq N$ and $N \leq O$, then $M \leq O$
- **Antisymmetry up to equivalence**: If $M \leq N$ and $N \leq M$, then $M.\text{toDFA} \simeq_\ell N.\text{toDFA}$

An `AccessibleFinDFA` M is *minimal* if for every N with $N \leq M$, we have $M.\text{toDFA} \simeq_\ell N.\text{toDFA}$.

Proof of Partial Order Properties. **Reflexivity**: The identity morphism on $M.\text{toDFA}$ is surjective.

Transitivity: Given surjective morphisms $O \twoheadrightarrow N$ and $N \twoheadrightarrow M$, their composition $O \twoheadrightarrow M$ is also surjective.

Antisymmetry: Given surjective morphisms $f : N \twoheadrightarrow M$ and $g : M \twoheadrightarrow N$, we use the accessibility property to show these are mutual inverses. For any state s in M, there exists a word w such that $M.\text{evalFrom}(M.\text{start}, w) = s$. Using the morphism properties and surjectivity, we can show $g \circ f = \text{id}$ and $f \circ g = \text{id}$. □

Chapter 3

Nerode Equivalence on AccessibleFinDFAs

Definition 8 (Nerode Equivalence)**.**

Let M be an `AccessibleFinDFA` with alphabet α and state space σ.

A word w *indistinguishes* two states $s_1, s_2 \in \sigma$, denoted $\text{Indist}_M(w, s_1, s_2)$, if evaluating from both states with input w leads to the same acceptance outcome:

$$M.\text{evalFrom}(s_1, w) \in M.\text{accept} \iff M.\text{evalFrom}(s_2, w) \in M.\text{accept}$$

The *Nerode equivalence relation* $\sim_{\text{Nerode}}$ on states is defined by:

$$s_1 \sim_{\text{Nerode}} s_2 \iff \forall w : \text{List}(\alpha), \text{Indist}_M(w, s_1, s_2)$$

Two states are Nerode equivalent if and only if all words indistinguish them.

The *bounded Nerode equivalence relation* $\sim^{(n)}_{\text{Nerode}}$ at level n is defined by:

$$s_1 \sim^{(n)}_{\text{Nerode}} s_2 \iff \forall w : \text{List}(\alpha), |w| \leq n \implies \text{Indist}_M(w, s_1, s_2)$$

Both relations are equivalence relations (reflexive, symmetric, and transitive).

Since the alphabet α is finite, there are only finitely many words of length $\leq n$, making $\sim^{(n)}_{\text{Nerode}}$ decidable.

Lemma 9 (Bounded Nerode Stabilization)**.**

The bounded Nerode relation satisfies the following properties:

1. ***Monotonicity**: For $n \leq m$, we have $\sim^{(m)}_{Nerode} \subseteq \sim^{(n)}_{Nerode}$ (the relation becomes finer as the bound increases).*
2. ***Stabilization**: If $\sim^{(n)}_{Nerode} = \sim^{(n+1)}_{Nerode}$, then $\sim^{(n)}_{Nerode} = \sim^{(m)}_{Nerode}$ for all $m \geq n$.*
3. ***Finite stabilization**: The relation stabilizes at or before level $|\sigma|$, i.e., $\sim^{(|\sigma|)}_{Nerode} = \sim^{(|\sigma|+1)}_{Nerode}$.*
4. ***Equivalence with unbounded relation**: $\sim^{(|\sigma|)}_{Nerode} = \sim_{Nerode}$.*

Proof of Stabilization. We use finpartitions to prove stabilization. Each bounded Nerode relation $\sim^{(n)}_{\text{Nerode}}$ induces a partition of the state space σ into equivalence classes.

Key insight: If two bounded Nerode relations induce partitions with the same number of parts, then the relations are equal.

Monotonicity: Since longer words provide more distinguishing power, $\sim^{(n+1)}_{\text{Nerode}} \subseteq \sim^{(n)}_{\text{Nerode}}$.

Cardinality bound: Each partition has at most $|\sigma|$ parts since there are only $|\sigma|$ states.

Pigeonhole argument: Consider the sequence of partition cardinalities:

$$|\text{parts}(\sim^{(0)}_{\text{Nerode}})|, |\text{parts}(\sim^{(1)}_{\text{Nerode}})|, ...$$

This is a weakly increasing sequence (by monotonicity) bounded above by $|\sigma|$. Either:

- The sequence stabilizes at some level $n < |\sigma|$, or
- At level $|\sigma|$, we have $|\text{parts}(\sim^{(|\sigma|)}_{\text{Nerode}})| = |\sigma|$

In both cases, stabilization occurs by level $|\sigma|$.

Equivalence with unbounded relation: Once stabilized, the bounded relation equals the unbounded Nerode relation since no additional distinguishing power is gained from longer words. □