Two Induction-Free Constructions For Finite Automata Determinization

Matheus do Ó
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
matheusost@ic.ufrj.br

Hugo Musso Gualandi
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
hugomg@ic.ufrj.br

Abstract

The standard algorithm to determinize a finite automaton is the subset construction, however its description and the proof of correctness are usually presented separately. We derive the subset construction from the specification of a deterministic automaton by expressing the language recognized by the automaton as a fixed point of a function. We do this in two ways, one based on the right languages of the states, and the other based on the left languages. The derivation based on right languages resembles a derivation made by Kozen using matrix techniques and Kleene algebra. The derivation based on left languages, on the other hand, uses non-linear functions and has no matrix analogue. Our proofs are induction-free and instead we rely on the uniqueness of fixed-points. In summary, we highlight that an algebraic model of an automaton allows an elegant derivations of the subset construction and highlight the duality between the left and right languages of an automaton.

Keywords

Determinism, Fixed-point, Minimization

1 Introduction

Every finite automaton can be converted to a deterministic finite automaton by the subset construction, a standard algorithm which produces a new automaton with 2^n states [4]. However, most presentations from this algorithm presents its proof of correctness as a separate step from its construction. This does not need to be the case. In this work, we present two derivations of the determinization algorithm based on the concepts of left and right languages of a finite automaton.

The left language of a state describes the set of paths from the initial states to that state, and its right language of a state describes the set of paths from that state to the final states of the automaton. In Section 2, we present those definitions and we show how to use linear systems of equations and their fixed points to compute these languages and, therefore, to describe the semantics of the automaton as a whole.

Under this framework, we present the two derivations for the determinization algorithm, one based on unions of right languages of the states (Section 3.2), and the other based on intersections of left languages (Section 3.3). It turns out both of these derivations lead to the subset construction, but in dual ways. Furthermore, because the derivation manipulates the language of the automaton, it can be seen that the language of the transformed automaton is the same as the original automaton.

Finally, in Section 4 we talk about related work in algebraic methods for automata proofs and in Section 5 we conclude our paper presenting future works, as well.

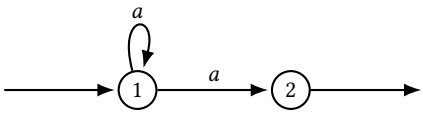
2 Linear System of Equations

We begin by recalling the definition of our main study object, a Non-deterministic Finite Automaton (NFA). A NFA is a finite graph with labelled transitions.

Definition 2.1. A **Non-deterministic Finite Automaton** (NFA) $\mathcal{M} = \langle A, Q, I, T, \to \rangle$ is composed of

- An alphabet A, a finite set;
- A state set Q, a finite set;
- A set of initial states I, a subset of Q;
- A set of final states T, a subset of Q;
- A transition relation $\to \subseteq Q \times A \times Q$.

In the example Automaton 1, the alphabet is the singleton $\{a\}$, the state set is $\{1, 2\}$, the set of initial states is $\{1\}$ and set of final states is $\{2\}$. The set of transitions is $\{1 \xrightarrow{a} 1, 1 \xrightarrow{a} 2\}$.



Automaton 1: A NFA for the language a^*a.

The global behaviour of a finite automaton can be denoted by the language it recognizes. Let $\mathcal{M} = \langle A, Q, I, T, \to \rangle$ be a NFA.

Definition 2.2. The **language recognized** by $\mathcal{M}$ is the set of words that label paths which lead from an initial state to a final state.

$$\mathcal{L}(\mathcal{M}) = \{w \mid i \xrightarrow{w} t,\ i \in I, t \in T\}$$

To write proofs about automata it is often useful to use a fine-grained definition of its behaviour that looks at individual states.

Definition 2.3. The **left language** of $q \in Q$ is the set of words that label paths which lead from an initial state of $\mathcal{M}$ to q.

$$[\![q]\!]_L = \{w \mid i \xrightarrow{w} q,\ i \in I\}$$

Definition 2.4. The **right language** of $q \in Q$ is the set of words that label paths which lead from q to a final state of $\mathcal{M}$.

$$[\![q]\!]_R = \{w \mid q \xrightarrow{w} t,\ t \in T\}$$

The language recognized by $\mathcal{M}$ can be expressed in terms of the left or right languages of its states.

$$\mathcal{L}(\mathcal{M}) = \bigcup_{i \in I} [\![i]\!]_R = \bigcup_{t \in T} [\![t]\!]_L \tag{1}$$

2.1 The Left Language Equations

To compute the left or right languages of a given automaton, we can solve a linear system of equations. Consider the Automaton 1. Its left languages $[\![1]\!]_L = a^*$ and $[\![2]\!]_L = a^*a$ are the unique solution of the system

$$\begin{aligned} x_1 &= x_1 a \cup \varepsilon; \\ x_2 &= x_1 a. \end{aligned} \tag{2}$$

From the structure of the automaton we can deduce that the left languages should be a solution of the above system. There are only two ways for a path to reach state 1: either we take the empty path, because 1 is a starting state, or we extend another path that arrives at 1, because $1 \xrightarrow{a} 1$. Thus, we should have $[\![1]\!]_L = [\![1]\!]_L \cdot a \cup \varepsilon$. The only way to arrive at state 2 is via $1 \xrightarrow{a} 2$, so, by similar logic, we should have $[\![2]\!]_L = [\![1]\!]_L \cdot a$. In general, the equation for x_q has a term $x_p a$ for each transition $p \xrightarrow{a} q$ and has a term ε if q is an initial state.

In Section 2.3 we will show that the above solution is in fact the unique solution to the system of equations.

2.2 The Right Language Equations

Dually, we can also create equations for the right languages. Continuing with the same automaton as before, the right languages $[\![1]\!]_R = a^*a$ and $[\![2]\!]_R = \varepsilon$ are solutions to the system of equations

$$\begin{aligned} x_1 &= a x_1 \cup a x_2; \\ x_2 &= \varepsilon. \end{aligned} \tag{3}$$

We can verify that the right languages satisfy this system of equations. There are only two ways to leave from state 1: the transitions $1 \xrightarrow{a} 1$ and $1 \xrightarrow{a} 2$. Thus, $[\![1]\!]_R = a[\![1]\!]_R \cup a[\![2]\!]_R$. State 2 is a final state that has no transitions that leave it. Thus, $[\![2]\!]_R = \varepsilon$.

So if in the left system we are aware about the *origins* of the transitions, here the awareness is about its *destinations*.

2.3 Uniqueness of Fixed-Points

The solutions to the systems of equations (2) and (3) can be seen as fixed points of functions of vectors of languages,

$$f_L\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1 a \cup \varepsilon \\ x_1 a \end{bmatrix} \quad \text{and} \quad f_R\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} a x_1 \cup a x_2 \\ \varepsilon \end{bmatrix}. \tag{4}$$

We already established that f_L and f_R have fixed points: the vector of left languages and the vector of right languages, respectively. But we also want to prove that those fixed points are unique. We'll do so for f_R, but the argument for f_L would be the same. The approach we take asks for a notion of similarity between languages:

Definition 2.5. Two languages are **n-similar** when they contain the same words of length less than n.

$$x \equiv_n y \iff \forall w\big(|w| < n \implies (w \in x \iff w \in y)\big)$$

The following properties follow directly from the definition:

Lemma 2.6. *$x = y$ if and only if $x \equiv_n y$ for all n.*

Lemma 2.7. *If $x \equiv_n y$, then $ax \equiv_{n+1} ay$.*

Lemma 2.8. *If $x_1 \equiv_n y_1$ and $x_2 \equiv_n y_2$, then $(x_1 \cup x_2) \equiv_n (y_1 \cup y_2)$.*

We can extend n-similarity to vectors of languages, by element-wise comparison. Under this definition we can state that applying f_R to a pair of language vectors results in language vectors that are more similar than before.

Lemma 2.9 (Contraction). *If $x \equiv_n y$, then $f_R(x) \equiv_{n+1} f_R(y)$.*

Proof. The terms in the equations for f_R all have the form $f_R(x)_i = a_1 x_1 \cup \ldots . a_n x_n \cup \varepsilon$, with the ε term being optional. Thus, if we start with $x_i \equiv_n y_i$ for all i, then by Lemmas 2.7 and 2.8 we will end up with $f(x)_i \equiv_{n+1} f(y)_i$ for all i. □

An immediate corollary is that f_R has at most one fixed point.

Corollary 2.10. *If $f_R(x) = x$ and $f_R(y) = y$, then $x = y$.*

Proof. We can show that $x \equiv_n y$ for all n, which implies $x = y$. The argument is by induction on n. In the base case, we trivially have $x \equiv_0 y$. In the inductive case, if we assume that $x \equiv_n y$ then by Lemma 2.9 we also have $f_R(x) \equiv_{n+1} f_R(y)$. But since x and y are fixed points, that also means $x \equiv_{n+1} y$. □

Crucially, the above proofs assume that the automaton does not have ε-transitions. Equations such as $x = x$ and $x = x \cup \varepsilon$ have multiple solutions and would break Lemma 2.9. Thankfully, it is always possible to remove the ε-transitions from an NFA and that is also the first step in the procedure for determinizing them.

3 Deterministic Finite Automata

In many situations it is desirable to have a deterministic automaton, one in which each word leads to a single destination.

Definition 3.1. A **Deterministic Finite Automaton** (DFA) is an automaton that:

- Has only one initial state;
- For every state p and label a, there is exactly one state q such that $p \xrightarrow{a} q$.

We can always transform a non-deterministic automaton into a deterministic automaton that recognizes the same language. The classic way to do this is the subset construction, which we discuss in Section 3.1. However, it is not obvious that this process preserves the language recognized by the automaton, and usually a separate proof is necessary to assert this result. In Sections 3.2 and 3.3 we present two derivations of the subset construction from its specification, respectively based on the right and left languages.

3.1 Subset Construction

The key insight of the subset construction is to construct an automaton with one state for each subset of states from the non-deterministic automaton.

Definition 3.2. The **subset construction** is an automata transformation $\varphi\,(\langle A, Q, I, T, \to\rangle) = \langle A, \widehat{Q}, \widehat{I}, \widehat{T}, \rightsquigarrow\rangle$ such that

- $\widehat{Q} = \mathcal{P}(Q)$;
- $\widehat{I} = \{I\}$;
- $\widehat{T} = \{P \in \widehat{Q} \mid P \cap T \neq \emptyset\}$;
- $P \overset{a}{\rightsquigarrow} R \iff R = \left\{r \in Q \mid \exists p \in P \left(p \xrightarrow{a} r\right)\right\}$.

The new set of states is the power set of the states. The deterministic automaton has a single initial state, which is the set of initial states from the non-deterministic automaton. The final states are those that contain at least one final state from the non-deterministic automaton. The transition relation is a function that receives a source state P and a label a and outputs the set of states reachable via a from states in P.

It is clear that the new automaton is deterministic, since it has only one initial state and its transition relation is a function. However, it is not so clear that the language of the new automaton is the same as the original one. In the following sections we will use algebraic methods, based on the systems of equations that we described, to derive the subset construction in ways that unambiguously preserve the language.

To ensure this preservation, a high-level strategy is to start with the system of equations describing the left (or right) languages of the original automaton and convert it to the system of equations describing the left (or right) languages of the new automaton. Since we are dealing with languages the whole time, it is easy to assert that the language of the automaton as a whole is preserved.

3.2 Unions

One way to make an automaton deterministic is to focus on transforming the transition relation into a function. The right languages system of equations is well suited for this, because all the transitions that leave a state are grouped in the equation for that state.

Consider the Automaton 1, which is non-deterministic because state 1 has two transitions labeled by a. In the system of equations for the right languages of this automaton, shown in (3), the equation for that state is $x_1 = ax_1 \cup ax_2$. There are two terms that begin with a; to fix that, we can apply the distributivity rule and rewrite the equation as $x_1 = a(x_1 \cup x_2)$.

This suggests that we should also have an equation for $x_1 \cup x_2$. In the worst case scenario we will need all possible unions of languages, which are provided by the unifier function

$$h_\cup\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1 \cup x_2 \\ x_1 \\ x_2 \\ 0 \end{bmatrix}. \tag{5}$$

In general, $h_\cup$ receives a n-dimensional vector and outputs a 2^n-dimensional one.

Definition 3.3. Let A and Q be finite sets, **the unifier function** is a mapping $h_\cup : \mathcal{P}(A^*)^Q \to \mathcal{P}(A^*)^{2^Q}$ such that

$$h_\cup(x)_P = \bigcup_{p \in P} x_p.$$

To construct the complete set of equations for the new automaton, we can apply $h_\cup$ to both sides of (3), giving $h_\cup(x) = h_\cup(f_L(x))$. Then, using the distributivity rule, we can factor out the common labels to assure that each equation has exactly one term for each label a. In equations that do not have terms with that label, we insert a transition to the empty language, 0.

$$\begin{array}{lll} x_1 \cup x_2 & = ax_1 \cup ax_2 \cup \varepsilon & = a\,(x_1 \cup x_2) \cup \varepsilon \\ x_1 & = ax_1 \cup ax_2 & = a\,(x_1 \cup x_2) \\ x_2 & = \varepsilon & = a0 \cup \varepsilon \\ 0 & = 0 & = a0 \end{array} \tag{6}$$

This set of equations is almost the system for an automaton. The only problem is that in the first column the terms are not variables, but unions of them. We can obtain a suitable system of equations by replacing each of these unions by a variable.

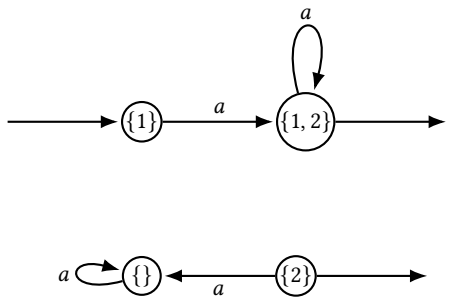
$$\begin{aligned} y_{\{1,2\}} &= ay_{\{1,2\}} \cup \varepsilon; \\ y_{\{1\}} &= ay_{\{1,2\}}; \\ y_{\{2\}} &= ay_{\{\}} \cup \varepsilon; \\ y_{\{\}} &= ay_{\{\}}. \end{aligned} \tag{7}$$

In this new system of equations, variable y_P corresponds to the union of the variables indexed by P. For example, $y_{\{1,2\}}$ corresponds to the union $x_1 \cup x_2$.

Clearly, (7) is a system of right languages for an automaton. Its states are the indices of the variables, and the transitions and final states are determined by the system of equations. We have not yet chosen the initial state.

The right languages of the new automaton are solutions of system (7) and may be seen as a fixed point $y = g_R(y)$. But how can we compute these right languages? We know that the set of right languages of the original automaton is the unique fixed point of $x = f_R(x)$. From the first and third column of (6), we know that $h_\cup(x) = g(h_\cup(x))$ and therefore $h_\cup(x)$ is a fixed point of g. Since g has an unique fixed point, we conclude that the right languages of the new automaton are equal to $h_\cup(x)$.

We can now choose an initial state. The language recognized by the original automaton is the union of the right languages of its initial states, in this case x_1, which is equal to $y_{\{1\}}$. This suggests that we choose $\{1\}$ as the initial state. The result is the Automaton 2.



Automaton 2: A DFA for the language a^*a.

By construction we know that the Automaton 2 is deterministic and that it recognizes the same language as the Automaton 1. In fact, we can verify that the new automaton is isomorphic to the one constructed by the subset construction.

Theorem 3.4. *The union-based determinization produces the same automaton as the subset construction.*

Proof. We know that in the new system of equations, the equation for y_P comes from the union of the equations for x_p for all $p \in P$. We need to check that all components of the automaton match.

Set of states. The set of states, as prescribed by $h_\cup$, is $\mathcal{P}(Q)$.

Initial state. The initial state is $\{I\}$ because the language of the original automaton is equal to the union of the right languages of its initial states.

Final states. The equation for y_P contains ε precisely when at least one of its constituent equations x_p does so. This means that P is a final state of the new automaton precisely when it contains at least one final state of the original automaton.

Transition. Consider some transition $P \overset{a}{\rightsquigarrow} R$. The destination state R is determined by the ay_R term in the equation for y_P. When we compute the union of the constituent equations for x_p, the term that will give rise to ay_R is the union of all ax_r for all $p \overset{a}{\to} r$. Thus, $P \overset{a}{\rightsquigarrow} R$ precisely when R is the set of states that can be reached from P in the original automaton. □

3.3 Intersections

Another way to think about deterministic automata is to ask that each word leads to a single state. Left languages are ideal for describing these ideas. The following lemma precisely characterizes how determinism relates to left languages.

Definition 3.5. A finite automaton is **accessible** if it has no state whose left language is empty.

Lemma 3.6. *If an automaton is accessible, has only one initial state and the left languages of its states are pair-wise disjoint, then it is deterministic.*

Proof. Suppose the automaton has transitions $p \overset{a}{\to} q$ and $p \overset{a}{\to} r$. We can show that $q = r$. Since the automaton is accessible, there is a path $i \overset{w}{\to} p$, where i is the initial state. Thus, there are also paths $i \overset{wa}{\to} q$ and $i \overset{wa}{\to} r$. Since the left languages are disjoint, wa leads to a single state and therefore $q = r$. □

We use this idea to construct a deterministic automaton and our example will again be the Automaton 1. The system of equations for its left languages is shown in (2). To obtain disjoint left languages as asked by the Lemma 3.6, we can use a separator function that outputs disjoint values:

$$h_\cap\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1 \cap x_2 \\ x_1 \cap x_2^c \\ x_1^c \cap x_2 \\ x_1^c \cap x_2^c \end{bmatrix}, \tag{8}$$

where $\circ^c$ indicates the set complement. In general, we have

Definition 3.7. Let A and Q be finite sets, we define the **separator function** as $h_\cap : \mathcal{P}(A^*)^Q \to \mathcal{P}(A^*)^{2^Q}$ such that

$$h_\cap(x)_P = \left(\bigcap_{p \in P} x_p\right) \cap \left(\bigcap_{q \notin P} x_q^c\right). \tag{9}$$

Let us see what happens when we apply $h_\cap$ to both sides of (2).

$$\begin{aligned} x_1 \cap x_2 &= (x_1 a \cup \varepsilon) \cap (x_1 a) \\ x_1 \cap x_2^c &= (x_1 a \cup \varepsilon) \cap (x_1 a)^c \\ x_1^c \cap x_2 &= (x_1 a \cup \varepsilon)^c \cap (x_1 a) \\ x_1^c \cap x_2^c &= (x_1 a \cup \varepsilon)^c \cap (x_1 a)^c \end{aligned} \tag{10}$$

Next, we want to rewrite the right-hand side in terms of $h_\cap(x)$. We can rewrite x_1 and x_2 in terms of $h_\cap(x)$ as follows:

$$\begin{aligned} x_1 &= (x_1 \cap x_2) \cup \left(x_1 \cap x_2^c\right), \\ x_2 &= (x_1 \cap x_2) \cup \left(x_1^c \cap x_2\right). \end{aligned} \tag{11}$$

Substituting these identities into (10), we obtain

$$\begin{aligned} x_1 \cap x_2 &= \left(\left[(x_1 \cap x_2) \cup \left(x_1 \cap x_2^c\right)\right] a \cup \varepsilon\right) \\ &\quad \cap \left((x_1 \cap x_2) \cup \left(x_1 \cap x_2^c\right)\right) a; \\ x_1 \cap x_2^c &= \left(\left[(x_1 \cap x_2) \cup \left(x_1 \cap x_2^c\right)\right] a \cup \varepsilon\right) \\ &\quad \cap \left(\left[(x_1 \cap x_2) \cup \left(x_1 \cap x_2^c\right)\right] a\right)^c; \\ x_1^c \cap x_2 &= \left(\left[(x_1 \cap x_2) \cup \left(x_1 \cap x_2^c\right)\right] a \cup \varepsilon\right)^c \\ &\quad \cap \left((x_1 \cap x_2) \cup \left(x_1 \cap x_2^c\right)\right) a; \\ x_1^c \cap x_2^c &= \left(\left[(x_1 \cap x_2) \cup \left(x_1 \cap x_2^c\right)\right] a \cup \varepsilon\right)^c \\ &\quad \cap \left(\left[(x_1 \cap x_2) \cup \left(x_1 \cap x_2^c\right)\right] a\right)^c. \end{aligned} \tag{12}$$

We can simplify the formulas taking advantage of the fact that the terms from $h_\cap$ are disjoint. In the $x_1 \cap x_2$ case we select only the common terms that appear in both x_1 and x_2, namely $(x_1 \cap x_2)\, a$ and $\left(x_1 \cap x_2^c\right) a$. In the $x_1 \cap x_2^c$ case we select only the terms that appear in x_1 but not in x_2, namely ε. In general, an intersection between complemented and non-complemented factors will contain the terms that appear in all the non-complemented factors but do not appear in any of the complemented factors.

$$\begin{aligned} x_1 \cap x_2 &= (x_1 \cap x_2)\, a \cup \left(x_1 \cap x_2^c\right) a; \\ x_1 \cap x_2^c &= \varepsilon; \\ x_1^c \cap x_2 &= 0; \\ x_1^c \cap x_2^c &= \left(x_1^c \cap x_2^c\right) a \cup \left(x_1^c \cap x_2\right) a. \end{aligned} \tag{13}$$

To turn this into a system, we can replace the left-hand side terms by variables y indexed by the subset of variables x_i that are not complemented.

$$\begin{aligned} y_{\{1,2\}} &= y_{\{1,2\}} a \cup y_{\{1\}} a; \\ y_{\{1\}} &= \varepsilon; \\ y_{\{2\}} &= 0; \\ y_{\{\}} &= y_{\{\}} a \cup y_{\{2\}} a. \end{aligned} \tag{14}$$

This system of equations describes the states, initial state, and transitions of an automaton. We know that this automaton is deterministic because it has only one initial state, marked by ε, and each term only appears in one equation, which means that each combination of source state and label only leads to one destination state.

We can solve the system of equations using a similar argument to the one we used in Section 3.2. Systems 12 and 13 tell us that $h_\cap(x)$ is a solution to the equations for (14) and therefore is equal to the left languages of the automaton described by (14).

Now that we know the left languages, we can pick a set of final states that preserves the language of the automaton. The language of the original automaton is the union of the left languages of its final states. To match that, in the new automaton we can select as final states all the terms for which a final state appears not-complemented.

It should come to no surprise that the automation we just constructed is isomorphic with the subset construction.

Theorem 3.8. *The intersection-based determinization produces the same automaton as the subset construction.*

Proof. The equation for y_R is built by intersecting all equations for x_r with $r \in R$ and subtracting all equations with $r \notin R$. This means that the words present in y_R are precisely the words that are present in every x_r with $r \in R$ and nowhere else. This guarantees that all the languages y_R are disjoint and that each term appears in at most one equation.

States. The set of states is decided by $h_\cap$. There are 2^n states that correspond to subsets of Q.

Final states. As discussed above, the final states in the new automaton are the ones whose subset of non-complemented variables contains a variable for a final state in the original automaton.

Initial state. The equation for y_R contains an ε term when the x_r equations for $r \in R$ are precisely the ones that contain an ε term. This means that R is an initial state precisely when it is equal to the set of initial states in the original automaton.

Transitions. When we rewrite a variable in terms of $h_\cap$, as we did in (11), a variable x_p will be split into terms that correspond to y_P, one for each P that contains p. Thus, every term y_P ultimately originates from some term x_p with $p \in P$. With this in mind, we can deduce that the equation for y_R contains a term $y_P a$ when the equations for x_r for $r \in R$ are precisely the equations that contain some term $x_p a$ with $p \in P$. In conclusion, there is a transition $P \xrightarrow{a} R$ precisely when R is the set of states reachable from P via a. □

4 Related Work

The use of matrices and algebraic methods to reason about regular languages traces back to Conway [2]. A more recent treatment may be found in Sakarovitch [6].

An induction-free derivation of the subset construction based on Kleene algebra may be found and has been presented by Kozen [5]. His derivation features a matrix that computes a linear transformation X that is similar to our unifier function (Definition 3.3). However, there is no analogue of our separator function (Definition 3.7), which is not a linear transformation and cannot be represented by matrices.

One way to compute the left languages of an automaton is to reverse the automaton, compute the right languages of that, and then reverse the result again. For example, algorithms by Brzozowski [1] and Sengoku [7] do this to minimize non-deterministic automata. Because our construction can compute the left-language directly, we hypothesize that it may offer new ways to tackle such problems.

The proof of uniqueness of fixed points from Section 2.3 can be viewed through the lens of ultrametric semantics[3, 8]. The set of languages can be shown to be a complete metric space if we choose a metric based on the notion of n-similarity:

$$d(x,y) = \begin{cases} 0, & \text{if } x = y; \\ 2^{-n} & \text{if } x \neq y. \end{cases}$$

where n is the largest n such that $x \equiv_n y$. Under this metric, the functions f_L and f_R are contractions, with $d\big(f(x), f(y)\big) \leq \frac{1}{2} d(x,y)$, and Banach's fixed point theorem guarantees that every contraction in a non-empty complete metric space has an unique fixed point.

Proofs in the metric framework tend to exploit the uniqueness of fixed points, in contrast to proofs in the more common order-theoretical framework, which reason about least fixed points.

5 Conclusion

We presented one way in which the well-known subset construction procedure can be reduced to linear system transformation. Also, we proposed two different induction-free derivations approaches for the determinization problem through unions of right languages and intersections of left languages from an automaton. We showed that both derivations are symmetric and equivalent to the classic subset construction. We hope that such work can show how useful a fixed-point approach can be to formal language and automata theory.

Artifact Availability

This paper does not provide any artifacts.

Acknowledgements

Funded by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) – Project Code 403601/2023-1. Matheus do Ó received financial support from Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), with grant number 88887.317313/2026-00.

References

[1] Janusz A. Brzozowski. 1962. Canonical regular expressions and minimal state graphs for definite events. https://api.semanticscholar.org/CorpusID:118363215
[2] John Horton. Conway. 1971. *Regular algebra and finite machines, [by] J.H. Conway.* Chapman and Hall, London.
[3] J. W. de Bakker and Erik de Vink. 1996. *Control flow semantics.* The MIT Press City: Cambridge, Mass.
[4] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2001. Introduction to automata theory, languages, and computation, 2nd edition. *SIGACT News* 32, 1 (March 2001), 60–65. doi:10.1145/568438.568455
[5] D. Kozen. 1994. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. *Information and Computation* 110, 2 (1994), 366–390. doi:10.1006/inco.1994.1037
[6] Jacques Sakarovitch. 2009. *Elements of Automata Theory.* Cambridge University Press, Cambridge.
[7] Hiroaki Sengoku and Shuzo YAJIMA. 1992. Minimization of nondeterministic finite automata. *Master's thesis, Kyoto University* (1992).
[8] Franck van Breugel. 2001. An introduction to metric semantics: operational and denotational models for programming and specification languages. *Theoretical Computer Science* 258, 1 (2001), 1–98. doi:10.1016/S0304-3975(00)00403-5