Minimization II: Brzozowski Strikes Back

Nelma Moreira Rogério Reis Gonçalo Teixeira *

CMUP & DCC, Faculdade de Ciências da Universidade do Porto,
Rua do Campo Alegre, 4169-007 Porto, Portugal
{nelma.moreira, rogerio.reis, goncalo.teixeira}@fc.up.pt

Abstract

Acyclic Deterministic Finite Automata (ADFAs) are an efficient data structure for representing finite sets of strings, with applications ranging from pattern matching to natural language processing and syntax checking. In this paper, we identify a class of regular expressions representing finite languages whose Brzozowski automaton is guaranteed to be minimal, and show that any ADFA can be efficiently converted into an equivalent expression from this class. By combining these results, we derive a new minimization algorithm for ADFAs that achieves minimality through a reduction to regular expressions in a suitable form, thus avoiding the explicit computation of the Nerode equivalence classes.

Keywords: Acyclic Deterministic Finite Automata, Minimal Automata, Finite Languages, Brzozowski Derivatives.

1 Introduction

Regular languages and their representations are among the most fundamental and well-studied objects in theoretical computer science. In particular the study of finite languages and regular expressions have found particular importance. On the one hand, finite languages and their automata representations in the form of Acyclic Deterministic Finite Automata (ADFAs)

*This work was partially supported by CMUP, which is financed by national funds through FCT under the project with DOI 10.54499/UID/00144/2025; and by AVATAR through the FCT project 2023.12169.PEX. And also project URBAN-NET, with the reference NORTE2030-FEDER-02697300, co-financed by the European Union, through the NORTE 2030 Regional Program, of Portugal 2030.text

have found application on a diverse range of fields from natural language processing to bioinformatics, where they serve as an efficient data structure for syntax checking and pattern searching problems that are ubiquitous in those areas. Regular expressions, on the other hand, provide an algebraic characterization of regular languages and have become indispensable in the formal verification of imperative programs and network behavior.

A central tool in the study of regular expressions is the Brzozowski derivative [Brz64], which provides a syntactic method for constructing a Deterministic Finite Automaton (DFA) directly from a regular expression. Although experimental evidence suggests that the resulting Brzozowski automaton is typically close to minimal [ORT09], and is in fact minimal in many cases, no syntactic or algebraic characterization of the class of regular expressions for which it is always minimal is known. Closing this gap is of both theoretical and practical interest: understanding what structural properties of a regular expression guarantee a minimal automaton could directly improve the performance of algorithms that rely on the construction and manipulation of automata and regular expressions.

In this paper, we introduce a syntactic normal form for regular expressions representing finite languages, which we call *head normal form*, and show that the Brzozowski automaton of any expression in this form is minimal. We further show that this normal form is of practical interest by presenting a new minimization algorithm for ADFAs.

The remainder of the paper is organized as follows. In Section 2 we fix notation and recall the necessary background on Brzozowski derivatives and ADFAs. In Section 3 we introduce head normal form and prove that the Brzozowski automaton of any expression in this form is minimal. In Section 4 we present a linear-time algorithm that converts an ADFA into an equivalent expression in head normal form. Finally, in Section 5 we combine these results to synthesize a minimization algorithm for ADFAs and prove its correctness.

2 Preliminaries

In this section we review some definitions and notation from the theory of automata and formal languages. For more details, we refer the reader to the standard references [HU79; Yu97].

An *alphabet* Σ is a finite, non-empty set of symbols. A *word* w over Σ is a finite sequence of symbols. The empty word is denoted by ε. The *length* of a word w is denoted by $|w|$. The set of all words (resp. non-empty words)

is denoted by $\Sigma^\star$ (resp. Σ^+) and corresponds to the free monoid (resp. free semigroup) generated by Σ. A subset $L \subseteq \Sigma^\star$ is called a *language*. A language is said to be *finite* if its cardinality is finite. Given a word $w \in \Sigma^\star$ and a language $L \subseteq \Sigma^\star$, the *(left) quotient* of L by w is $w^{-1}L = \{u \in \Sigma^\star \mid wu \in L\}$.

Let $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_k\}$ be a k-element alphabet. Then, the set RE_Σ of regular expressions over Σ is given by the following grammar:

$$\alpha \to \emptyset \mid \varepsilon \mid \sigma_1 \mid \cdots \mid \sigma_k \mid (\alpha + \alpha) \mid (\alpha \cdot \alpha) \mid (\alpha^\star),$$

where the symbol $\cdot$ and the parenthesis are often omitted. The language represented by a regular expression α is denoted by $\mathcal{L}(\alpha)$ and is inductively defined as follows:

$$\begin{gathered} \mathcal{L}(\emptyset) = \emptyset, \qquad \mathcal{L}(\varepsilon) = \{\varepsilon\}, \qquad \mathcal{L}(\sigma) = \{\sigma\}, \\ \mathcal{L}(\alpha + \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta), \qquad \mathcal{L}(\alpha \cdot \beta) = \mathcal{L}(\alpha)\mathcal{L}(\beta), \qquad \mathcal{L}(\alpha^\star) = \mathcal{L}(\alpha)^\star. \end{gathered}$$

Two regular expressions $\alpha, \beta \in \mathrm{RE}_\Sigma$ are *equivalent*, written $\alpha = \beta$, if $\mathcal{L}(\alpha) = \mathcal{L}(\beta)$. If they are syntactically equal we write $\alpha \equiv \beta$.

The algebraic structure $\langle \mathrm{RE}_\Sigma, \cdot, +, \emptyset, \varepsilon \rangle$ forms the free Kleene Algebra [Con12; Koz94; Sal66] over Σ. In particular it follows that, for all $\alpha \in \mathrm{RE}_\Sigma$

$$\alpha \cdot \emptyset = \emptyset = \emptyset \cdot \alpha, \qquad \alpha + \emptyset = \alpha = \emptyset + \alpha, \qquad \alpha \cdot \varepsilon = \alpha = \varepsilon \cdot \alpha.$$

A regular expression is said to be *trim* if it can't be simplified using any of the above rules. For the rest of this paper we only consider trim regular expressions since the simplification can be done in linear time. For any proposition φ, the *Iverson bracket* $[\varphi]$ is defined as ε if φ is true and $\emptyset$ if φ is false. [1]

A *deterministic finite automaton* (DFA) is a 5-tuple $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$, where Q is a finite set of states, Σ is an alphabet, $\delta : Q \to \Sigma \to Q$ is a transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. The transition function can be naturally extended to words $\delta : Q \to \Sigma^\star \to Q$.

A state $q \in Q$ is said to be *accessible* if there exists a word $w \in \Sigma^\star$ such that $\delta(q_0, w) = q$. Similarly, a state $q \in Q$ is *useful* if there is a word $w \in \Sigma^\star$ such that $\delta(q, w) \in F$. A DFA is said to be *trim* if all of its states are accessible and useful. The *right language* of a state $q \in Q$ is the set of words that take

[1] The usage of ε and $\emptyset$ instead of the more traditional 1 and 0 is justified by the injection of the Booleans into the regular expressions.

q to a final state, that is $\overrightarrow{\mathcal{L}}_{\mathcal{A}}(q) = \{w \in \Sigma^\star \mid \delta(q, w) \in F\}$. Two states are equivalent if the have the same right language.

The *language* accepted by a DFA $\mathcal{A}$ is the right language of its initial state, i.e., $\mathcal{L}(\mathcal{A}) = \overrightarrow{\mathcal{L}}_{\mathcal{A}}(q_0)$. Two DFAs are *equivalent* if they recognize the same language. Two isomorphic DFAs are equivalent.

A DFA $\mathcal{A}$ is *minimal* if there is no other equivalent DFA with fewer states. Equivalently, a DFA is minimal if and only if all of its states are accessible and have pairwise distinct right languages. The minimal DFA that recognizes any language L is unique up to isomorphism and can be computed in $\mathcal{O}(n \log n)$ time using Hopcroft's algorithm [Hop71].

2.1 Brzozowski Automaton

The concept of the *derivative* of a regular expression was introduced by Brzozowski [Brz64] as a clean and effective method for converting a regular expression into an equivalent DFA.

We begin by defining the auxiliary function $\varepsilon : \mathrm{RE}_\Sigma \to \{\emptyset, \varepsilon\}$, which determines whether a given expression accepts the empty word or not:

$$\varepsilon(\emptyset) = \emptyset, \qquad \varepsilon(\varepsilon) = \varepsilon, \qquad \varepsilon(\sigma) = \emptyset,$$
$$\varepsilon(\alpha + \beta) = \varepsilon(\alpha) + \varepsilon(\beta), \qquad \varepsilon(\alpha \cdot \beta) = \varepsilon(\alpha) \cdot \varepsilon(\beta), \qquad \varepsilon(\alpha^\star) = \varepsilon.$$

The *derivative* of a regular expression $\alpha \in \mathrm{RE}_\Sigma$ by a symbol $\sigma \in \Sigma$ is the expression $\mathrm{d}_\sigma(\alpha)$ whose language consists of the left quotient of the language denoted by α by σ, i.e., $\mathcal{L}(\mathrm{d}_\sigma(\alpha)) = \sigma^{-1}\mathcal{L}(\alpha)$. It is inductively defined by:

$$\mathrm{d}_\sigma(\emptyset) = \emptyset, \qquad \mathrm{d}_\sigma(\varepsilon) = \emptyset, \qquad \mathrm{d}_\sigma(\sigma') = [\sigma = \sigma'], \qquad \mathrm{d}_\sigma(\alpha^\star) = \mathrm{d}_\sigma(\alpha)\alpha^\star,$$
$$\mathrm{d}_\sigma(\alpha + \beta) = \mathrm{d}_\sigma(\alpha) + \mathrm{d}_\sigma(\beta), \qquad \mathrm{d}_\sigma(\alpha \cdot \beta) = \mathrm{d}_\sigma(\alpha)\beta + \varepsilon(\alpha)\mathrm{d}_\sigma(\beta).$$

The derivative is naturally extended to words $w \in \Sigma^\star$ by induction on the length of the word:

$$\mathrm{d}_\varepsilon(\alpha) = \alpha, \qquad \mathrm{d}_{\sigma w}(\alpha) = \mathrm{d}_w(\mathrm{d}_\sigma(\alpha)).$$

A word w is accepted by α if and only if the derivative by w accepts the empty word, i.e., $\varepsilon(\mathrm{d}_w(\alpha)) = \varepsilon$.

The following result shows that a regular expression can be decomposed into its nullability and its derivatives:

Theorem 2.1 ([Brz64]). *Let $\alpha \in \mathrm{RE}_\Sigma$ be a regular expression. Then*

$$\alpha = \varepsilon(\alpha) + \sum_{\sigma \in \Sigma} \sigma \mathrm{d}_\sigma(\alpha).$$

The set of dissimilar derivatives of a regular expression is, in general, not finite. Despite this, Brzozowski proved that by taking equivalence of regular expression modulo a subset of the axioms of Kleene Algebra, namely associativity, commutativity, and idempotency of union (know as the ACI axioms), the set of derivatives of a given regular expression is finite. If we extend the ACI axioms with associativity of concatenation $(\alpha \cdot \beta) \cdot \gamma = \alpha \cdot (\beta \cdot \gamma)$ and left distributivity $(\alpha + \beta) \cdot \gamma = \alpha\gamma + \beta\gamma$, we get the ACIAL axiom system [MMR14].

This result leads to a conversion algorithm that that bypasses the intermediate step of constructing and determinizing an NFA required by the Thompson [Tho68], Glushkov [Glu61], and partial derivative [Mir67; Ant96] methods. Let $\alpha \in \mathrm{RE}_\Sigma$ be a regular expression, then the *Brzozowski automaton* of α is the DFA $\mathrm{Brz}(\alpha) = \langle \mathcal{D}, \Sigma, \delta, [\alpha], F \rangle$, where $\mathcal{D}$ is the set of distinct derivatives of α modulo ACI, the transition function is given by $\delta([\beta], \sigma) = [\mathrm{d}_\sigma(\beta)]$, the initial state is the equivalence class of α and the final states are the derivatives that accept the empty word, i.e., $F = \{[\beta] \in \mathcal{D}(\alpha) \mid \varepsilon(\beta) = \varepsilon\}$. The correctness of the transformations follows from the properties of the derivative.

It is important to note that, in general, the Brzozowski automaton is not minimal. For example, Ilie and Yu [IY03] presented a family of regular expressions $\alpha_n = (a + b + \varepsilon)^n (a + b)^\star$ such that $\mathrm{Brz}(\alpha_n)$ has n states while its minimal equivalent DFA has a single state. Despite this, experimental results [ORT09] suggest that these automata are significantly smaller than those produced by other methods and are in a significant number of cases minimal.

2.2 Acyclic DFAs

An acyclic deterministic finite automaton (ADFA) is a DFA $\mathcal{A} = (Q \cup \{\Omega\}, \Sigma, \delta, q_0, F)$, where $F \subseteq Q$ and $q_0 \neq \Omega$ such that for all $\sigma \in \Sigma$, $\delta(\Omega, \sigma) = \Omega$ and there are no other cycles. The state Ω is called the *dead* or *sink* state, and is the only cyclic state of $\mathcal{A}$. It is easy to see that the language recognized by an ADFA is finite. In the rest of this paper we only consider accessible, complete and trim ADFAs whose only non-useful state is Ω.

Due to the fact that the language recognized by an ADFA is finite we can define the rank of a state $q \in Q$, denoted by $\mathrm{rk}(q)$ to be the length of the longest word accepted from q, that is, $\mathrm{rk}(q) = \max\{|w| \mid w \in \overrightarrow{\mathcal{L}}_{\mathcal{A}}(q)\}$. The rank of and ADFA $\mathcal{A}$ is $\mathrm{rk}(\mathcal{A}) = \max\{\mathrm{rk}(q) \mid q \in Q\}$. It's easy to see that $\mathrm{rk}(\mathcal{A}) = \mathrm{rk}(q_0)$. The rank function can be computed in linear time from

a depth-first search. Note that the transition function is strictly decreasing with respect to rank and that two equivalent states must have the same rank.

Lemma 2.1 ([Lot05]). *Let $\mathcal{A} = \langle Q \cup \{\Omega\}, \Sigma, \delta, q_0, F\rangle$ be an ADFA. Let $q \in Q$ be a state of $\mathcal{A}$ with $\mathrm{rk}(q) > 0$. Then $\mathrm{rk}(q) > \mathrm{rk}(\delta(q,\sigma))$, for all $\sigma \in \Sigma$. In particular, there is a symbol $\sigma \in \Sigma$ such that $\mathrm{rk}(\delta(q,\sigma)) = \mathrm{rk}(q) - 1$.*

Lemma 2.2 ([Lot05]). *Let $\mathcal{A} = \langle Q \cup \{\Omega\}, \Sigma, \delta, q_0, F\rangle$ be an ADFA. Let $p, q \in Q$ be two states of $\mathcal{A}$. If $\overrightarrow{\mathcal{L}}_{\mathcal{A}}(p) = \overrightarrow{\mathcal{L}}_{\mathcal{A}}(q)$ then $\mathrm{rk}(p) = \mathrm{rk}(q)$.*

Since ADFAs can be stratified into ranks, the minimality condition can be relaxed from equivalence to equality of states and can be computed in linear time using Revuz algorithm [Rev92].

Theorem 2.2 ([Rev92; Lot05]). *Let $\mathcal{A} = \langle Q \cup \{\Omega\}, \Sigma, \delta, q_0, F\rangle$ be an ADFA. The ADFA $\mathcal{A}$ is minimal if and only if for all distinct $p, q \in Q \cup \{\Omega\}$:*

$$(p \in F \mathbin{\dot\vee} q \in F) \vee (\exists \sigma \in \Sigma)\ \delta(p,\sigma) \neq \delta(q,\sigma)$$

3 Head Normal Form

In this section, we introduce the notion of *head normal form* for regular expressions representing finite languages. Since we restrict our attention to finite languages, we will only consider the fragment of regular expressions in which the Kleene star does not appear, to which we refer to as the *finite regular expressions*. A finite regular expression is said to be in *head normal form* if it is of the form

$$f + \sum_{\sigma \in \Sigma} \sigma \cdot \alpha_\sigma,$$

where $f \in \{\emptyset, \varepsilon\}$ and each α_σ is a finite regular expression in head normal form. Note that the expressions $\emptyset$, ε, and σ, for each $\sigma \in \Sigma$, are degenerate instances of the head normal form.

The fact that we call this form a normal form follows from the fact that any finite regular expression admits an equivalent expression in head normal form that can be recursively computed as follows.

Definition 3.1. The function $\mathrm{hnf} : \mathrm{RE}_\Sigma \to \mathrm{RE}_\Sigma$ is defined as

$$\mathrm{hnf}(\alpha) = \begin{cases} \emptyset, & \text{if } \alpha = \emptyset; \\ \varepsilon, & \text{if } \alpha = \varepsilon; \\ \sigma, & \text{if } \alpha = \sigma \in \Sigma; \\ \varepsilon(\alpha) + \sum_{\sigma \in \Sigma} \sigma \cdot \mathrm{hnf}(\mathrm{d}_\sigma(\alpha)), & \text{othewise.} \end{cases}$$

We establish the correctness of this definition through the following lemmata.

Lemma 3.1. *For every finite regular expression $\alpha \in \mathrm{RE}_\Sigma$, the previous definition of* hnf *is sound.*

Proof. We proceed by cases on the definition of hnf. In the base cases, α is already in head normal form, so the algorithm is sound. Otherwise, the sum has a finite number of summands, since the alphabet is finite, so it suffices to show that each inductive call is sound.

Consider the following well-founded measure defined on the finite regular expressions:

$$\ell(\emptyset) = 0, \qquad \ell(\varepsilon) = 0, \qquad \ell(\sigma) = 1,$$
$$\ell(\alpha + \beta) = \max\{\ell(\alpha), \ell(\beta)\}, \qquad \ell(\alpha \cdot \beta) = \ell(\alpha) + \ell(\beta).$$

It is easy to check that this syntactic measure semantically corresponds to the length of the longest word in the language denoted by α, i.e., $\ell(\alpha) = \max\{|w| \mid w \in \mathcal{L}(\alpha)\}$. Since the derivative of a regular expression with respect to a symbol semantically corresponds to a left quotient of the language it denotes, whenever $\mathrm{d}_\sigma(\alpha) \neq \emptyset$, the value of the measure decrease, i.e., $\ell(\mathrm{d}_\sigma(\alpha)) < \ell(\alpha)$. When $\mathrm{d}_\sigma(\alpha) = \emptyset$, the inductive call reaches a base case. Thus, by induction on the value of ℓ, each inductive call is sound. Hence, the definition is sound. □

Lemma 3.2. *For every finite regular expression $\alpha \in \mathrm{RE}_\Sigma$, the expression* $\mathrm{hnf}(\alpha)$ *is in head normal form.*

Proof. We proceed by induction on the structure of α. For the base cases, the resulting expression is trivially in head normal form. For the inductive case, by the inductive hypothesis, $\mathrm{hnf}(\mathrm{d}_\sigma(\alpha))$ is in head normal form for every $\sigma \in \Sigma$. Since $\varepsilon(\alpha) \in \{\emptyset, \varepsilon\}$, the resulting expression is in head normal form. □

Lemma 3.3. *For every finite regular expression $\alpha \in \mathrm{RE}_\Sigma$, $\mathcal{L}(\mathrm{hnf}(\alpha)) = \mathcal{L}(\alpha)$.*

Proof. We proceed by induction on the structure of α. For the base cases, α is already in head normal form, and thus $\mathcal{L}(\mathrm{hnf}(\alpha)) = \mathcal{L}(\alpha)$ trivially follows. For the inductive case, by the inductive hypothesis, $\mathcal{L}(\mathrm{hnf}(\mathrm{d}_\sigma(\alpha))) = \mathcal{L}(\mathrm{d}_\sigma(\alpha))$ for every $\sigma \in \Sigma$. By Theorem 2.1, the decomposition of an expression is language preserving. Hence, $\mathcal{L}(\mathrm{hnf}(\alpha)) = \mathcal{L}(\alpha)$. □

An important property of head normal form is that it is preserved under the derivative operator. In particular this implies that, without loss of generality, we can take the representatives of the equivalence classes that constitute the states of the Brzozowski automaton to be regular expressions in head normal form.

Lemma 3.4. *The Brzozowski derivative of a regular expression in head normal form is in head normal form.*

Proof. Let $\alpha = f + \sum_{\sigma\in\Sigma} \sigma\alpha_\sigma$ be a finite regular expression in head normal form. By linearity of the derivative operator, for all $\sigma' \in \Sigma$:

$$\mathrm{d}_{\sigma'}(\alpha) = \mathrm{d}_{\sigma'}(f) + \sum_{\sigma\in\Sigma} \mathrm{d}_{\sigma'}(\sigma\alpha_\sigma) = \sum_{\sigma\in\Sigma} [\sigma = \sigma']\alpha_\sigma = \alpha_{\sigma'}.$$

By definition, $\alpha_{\sigma'}$ is in head normal form. □

Unlike the general case, where two ACI-inequivalent regular expressions may still agree on their acceptance of the empty word and produce ACI-equivalent derivatives, the finite regular expressions in head normal form can always be separated either by their acceptance of the empty word or by one of their derivatives. We make this precise in the following lemma.

Lemma 3.5. *Let $\alpha, \beta \in \mathrm{RE}_\Sigma$ be two finite regular expressions in head normal form. Then*

$$\alpha \not\equiv_{\mathrm{ACI}} \beta \implies (\varepsilon(\alpha) \dot{\vee} \varepsilon(\beta)) \vee (\exists\sigma \in \Sigma)\ \mathrm{d}_\sigma(\alpha) \not\equiv_{\mathrm{ACI}} \mathrm{d}_\sigma(\beta)$$

Proof. Let $\alpha = f_\alpha + \sum_{\sigma\in\Sigma} \sigma\alpha_\sigma$ and $\beta = f_\beta + \sum_{\sigma\in\Sigma} \sigma\beta_\sigma$, with $f_\alpha, f_\beta \in \{\emptyset, \varepsilon\}$. Since $\alpha \not\equiv_{ACI} \beta$, they must differ in at least one of their summands. If they differ in their nullability summand, i.e., $f_\alpha \neq f_\beta$, then $\varepsilon(\alpha) \dot{\vee} \varepsilon(\beta)$ and the lemma holds. Otherwise, they must differ in one of their composite summands, i.e., for some $\sigma \in \Sigma$, $\sigma\alpha_\sigma \not\equiv_{\mathrm{ACI}} \sigma\beta_\sigma$. Since if $\alpha_\sigma \equiv_{ACI} \beta_\sigma$ held, then by congruence of $\equiv_{ACI}$ we would have $\sigma\alpha_\sigma \equiv_{ACI} \sigma\beta_\sigma$, a contradiction, we conclude $\alpha_\sigma \not\equiv_{\mathrm{ACI}} \beta_\sigma$. Finally, by definition of the derivative on head normal forms, $\mathrm{d}_\sigma(\alpha) \equiv_{ACI} \alpha_\sigma$ and $\mathrm{d}_\sigma(\beta) \equiv_{ACI} \beta_\sigma$, and therefore $\mathrm{d}_\sigma(\alpha) \not\equiv_{\mathrm{ACI}} \mathrm{d}_\sigma(\beta)$. □

It follows from Lemma 3.5 is that, since by Theorem 2.2, in the ADFA setting, it suffices to show that two states can be distinguished either by their finality or their transitions, the automata obtained by applying the Brzozowski derivative construction on finite regular expressions in head normal form are always minimal.

Theorem 3.2. *Let $\alpha \in \mathrm{RE}_\Sigma$ be a finite regular expression in head normal form. Then the automaton $\mathrm{Brz}(\alpha)$ is minimal.*

Proof. By the definition of the Brzozowski automaton, its states correspond to the equivalence classes modulo ACI of the derivatives of α. By Lemma 3.4, we know that, without loss of generality, we can take the representatives of those equivalence classes to be finite regular expressions in head normal form. Lemma 3.5 ensures that two different representatives are distinguishable either by their finality or their derivatives and thus, by Theorem 2.2, the automaton is minimal. □

Remark 3.3. *Since finite regular expressions in head normal form are deterministic, their sets of partial derivatives [Ant96] are either the empty set or singletons and thus can be identified with the Brzozowski derivatives. Consequently, for any finite regular expression α in head normal form, the partial derivative automaton $\mathrm{Pd}(\alpha)$ is deterministic and isomorphic to the Brzozowski automaton $\mathrm{Brz}(\alpha)$.*

Remark 3.4. *It is important to remark that our head normal form for finite regular expressions constitutes a particular case of the ACIAL normal form of Maia et al. [MMR14]. In their paper, they showed that the Partial derivate automaton of a finite regular expression in ACIAL normal form is isomorphic to the bissimilar automaton of its Glushkov automaton. In the nondeterministic setting, although it does not imply a minimal number of states, bissimulation implies that no two distinct states have the same right languages. Unfortunately, this result does not translate well to the deterministic setting as is shown by the following counter example:*

Let $\alpha = a(aab + a(ab + c)) + ca(ab + c)$. Note that this expression is in ACIAL normal form, but not in head normal form. Figure 1 shows the ADFA obtained by applying the Brzozowski derivative construction to α. It is easy to see that the automaton is not minimal since the states q_1 and q_2 are equivalent, i.e., $\overrightarrow{\mathcal{L}}(q_1) = \overrightarrow{\mathcal{L}}(q_2)$, and, thus, can be collapsed.

4 From Acyclic Automata to Expressions

The problem of converting a DFA into an equivalent regular expression is an interesting and historically important problem in automata theory [Sak06; GH08] since the existence of such an algorithm is an essential component of the proof of Kleene's Theorem [Kle56]. Since Kleene's original proof, several conceptually simpler and more efficient methods have been proposed,

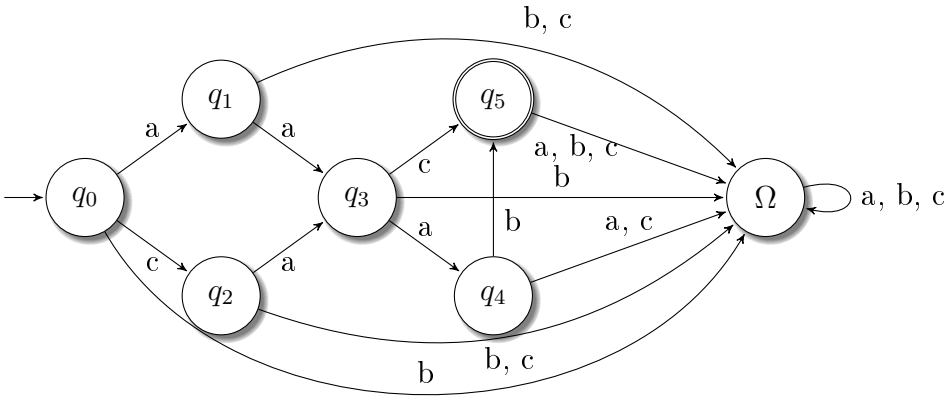


Figure 1: Non-minimal Brzozowski automaton $\mathrm{Brz}(\alpha)$ of a regular expression α in ACIAL normal form.

Algorithm 1: Converting an ADFA to a regular expression via rank ordering

Input: An ADFA $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$
Output: A regular expression $r_{\mathcal{A}}$ such that $\mathcal{L}(r_{\mathcal{A}}) = \mathcal{L}(\mathcal{A})$

```
Q ← SortByRank(Q);
r_Ω ← ∅;
for q ∈ Q do
    r_q ← [q ∈ F];
    for σ ∈ Σ do
        r_q ← r_q + σ · r_{δ(q,σ)};
return r_{q_0};
```

namely the State Elimination method of Brzozowski and McCluskey [BM63], and the method of solving a system of equations by Gaussian elimination using Arden's Lemma [Koz94; Ard61]. In this section we present a simplified linear time version of these algorithms that will turn out to be useful for the minimization algorithm in the next section.

The algorithm works by assigning to each state in increasing rank order a regular expression constructed as a combination of the regular expressions assigned to those states in lower ranks. The state Ω is assigned the expression $\emptyset$ while every other state $q \in Q$ is assigned an expression r_q of the form

$$r_q = [q \in F] + \sum_{\sigma \in \Sigma} \sigma r_{\delta(q,\sigma)}.$$

The fact that expressions are constructed in rank order paired with Lemma 2.1 guarantees that all subexpression have already been computed when they are needed. Additionally, the acyclical nature of the ADFA guarantees that the equations can be solved without resorting to Arden's Lemma. Thus, we have the following theorem:

Theorem 4.1. *Let $\mathcal{A} = \langle Q \cup \{\Omega\}, \Sigma, \delta, q_0, F\rangle$ be an ADFA. Let $r_{\mathcal{A}}$ be the regular expression obtained by the application of Algorithm 1 to $\mathcal{A}$. Then $\mathcal{L}(\mathcal{A}) = \mathcal{L}(r_{\mathcal{A}})$.*

Theorem 4.2. *Let $\mathcal{A}$ be an n-state ADFA over an alphabet with k symbols. Then Algorithm 1 runs in $\mathcal{O}(nk)$ time.*

Proof. The SortByRank step can be implemented in $\mathcal{O}(n + nk) = \mathcal{O}(nk)$ time using a topological sorting algorithm. Since all operations within the loops' bodies have unit cost, the loop body contributes another $\mathcal{O}(nk)$. The overall complexity is thus $\mathcal{O}(nk)$. □

An important consequence of the fact that the application of Arden's Lemma is not needed to solve the resulting system of equations is that the expressions assigned to each state by the algorithm are finite regular expressions in head normal form.

Lemma 4.1. *Let $\mathcal{A}$ be an ADFA. For all $q \in Q$, the regular expressions r_q obtained during the application of Algorithm 1 to $\mathcal{A}$ are finite and in head normal form.*

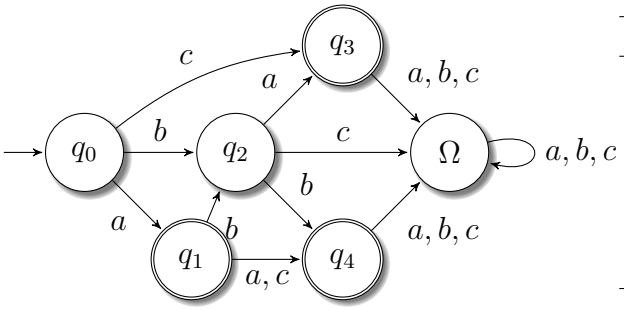
Proof. We proceed by induction on the rank of states.

Base Case: $q = \Omega$.
The algorithm assigns $r_\Omega = \emptyset$, which is trivially in head as it corresponds to the case in which the set of summands is empty.

Base Case: $\mathrm{rk}(q) = 0$.
The only state to which a state q with rank 0 can have transitions to is Ω. Since it has rank 0, it must accept a word of length 0 (i.e., the empty word ε) and so must be final. Thus, the assigned expression is $r_q = \varepsilon$ which is in head normal form.

Inductive Case: $\mathrm{rk}(q) = n$.
By induction hypothesis, let $r_{q'}$ be in head normal form for all $q' \in Q$ with $\mathrm{rk}(q') < n$. The algorithm constructs the following expression:

$$r_q = [q \in F] + \sum_{\sigma \in \Sigma} \sigma r_{\delta(q,\sigma)},$$



State	Expression
Ω	$\emptyset$
q_4	ε
q_3	ε
q_2	$a + b$
q_1	$\varepsilon + a + r_{q_2} + c$
q_0	$a \cdot r_{q_1} + b \cdot r_{q_2} + c$

Figure 2: Application of Algorithm 1 to an ADFA $\mathcal{A}$, yielding the regular expression $r_{\mathcal{A}} = a(\varepsilon + a + b(a + b) + c) + b(a + c) + c$.

It is easy to see that the first summand is either the empty set or consists only of ε, and the second summand is a sum of distinct, ordered symbols followed by $r_{\delta(q,\sigma)}$. Since by Lemma 2.1, $\mathrm{rk}(\delta(q, \sigma)) < n$ for all $\sigma \in \Sigma$, the induction hypothesis guarantees that each $r_{\delta(q,\sigma)}$ is in head normal form. Hence, r_q is in head normal form, completing the proof. □

In particular, since the algorithm returns the expression that was assigned to state q_0, we get the following corollary that guarantees that the returned a finite regular expression is in head normal form:

Corollary 4.1. *Let $\mathcal{A}$ be an ADFA. The regular expressions $r_{\mathcal{A}}$ obtained by the application of Algorithm 1 to $\mathcal{A}$ is finite and in head normal form.*

Example 4.3. *Let $\mathcal{A}$ be the ADFA represented in Fig. 2. We start by ordering the states by their ranks as follows: $\Omega < q_4 \leq q_3 < q_2 < q_1 < q_0$. Notice that since $\mathrm{rk}(q_3) = \mathrm{rk}(q_4)$ their elimination order does not affect the result of the algorithm. Then, we compute the regular expressions associated with each state in rank order, performing simplifications along the way, as shown in Fig. 2. The algorithm outputs $r_{\mathcal{A}} = r_{q_0} = a(\varepsilon+a+b(a+b)+c)+b(a+c)+c$.*

5 A New Minimization Algorithm

Within the formal language and automata theory community, Brzozowski is perhaps best known not for his definition of regular expression derivatives, but for his ingenious minimization algorithm [Brz62]. Unlike more standard approaches to automata minimization, such as Moore's and Hopcroft's [Moo56; Hop71], which compute the Nerode equivalence classes of the automaton's states and then form the corresponding quotient automaton, Brzozowski's minimization algorithm takes an operationally simpler route: reverse

the automaton, determinize it, reverse it again, and, finally, determinize it. The fact that this procedure yields a minimal automaton is a surprising result with very elegant proofs such as the one by Bonchi et al [Bon+12]. Interestingly, although the determinization step can incur in an exponential time complexity in the worst case, empirical studies have shown Brzozowski's minimization algorithm to be faster than Nerode-based approaches for a large set of automata.

In this section we present a new minimization algorithm from ADFAs that culminates from the results of the previous sections. It is in a lot of ways similar to Brzozowski's minimization algorithm. First, it does not directly compute the Nerode equivalence classes and rather relies on (co)algebraic properties of automata and regular expressions. Secondly, it is conceptually simple as it stems from the composition of two independently well understood operations. Lastly, it relies on the conversion between to another system of representing regular languages: NFAs in the case of Brzozowski and regular expressions in ours.

The algorithm's description is simple: given an ADFA, we convert it into a finite regular expression in head normal form using the algorithm of Section 4, and then we construct its Brzozowski automaton. The intuition behind it is that by running Algorithm 1 in a reverse topological order based on the rank of the states we are able to convert distinct but equivalent states into ACI-equivalent regular expressions that are then collapsed into the same state by Brzozowski's automata construction.

Theorem 5.1. *Let $\mathcal{A}$ be an ADFA. The automaton* $\mathrm{Brz}(r_{\mathcal{A}})$ *is the smallest ADFA recognizing* $\mathcal{L}(\mathcal{A})$.

Proof. We split the proof in two parts

First, we prove language preservation. By Theorem 4.1, Algorithm 1 computes an expression $r_{\mathcal{A}}$ such that $\mathcal{L}(r_{\mathcal{A}}) = \mathcal{L}(\mathcal{A})$. Since the Brzozowski's automata construction is language preserving, it follows that $\mathcal{L}(\mathrm{Brz}(r_{\mathcal{A}})) = \mathcal{L}(r_{\mathcal{A}}) = \mathcal{L}(\mathcal{A})$.

Secondly, we prove minimality of the resulting automaton. By Corollary 4.1, the expression $r_{\mathcal{A}}$ produced by Algorithm 1 is in head normal form. By Theorem 3.2, the Brzozowski automaton of an expression in head normal form is minimal. Hence $\mathrm{Brz}(r_{\mathcal{A}})$ is minimal.

Combining these two facts, $\mathrm{Brz}(r_{\mathcal{A}})$ is the minimal ADFA recognizing $\mathcal{L}(\mathcal{A})$. □

6 Conclusion

In this paper we presented a new minimization algorithm for ADFAs that, unlike classical approaches, avoids the explicit computation of the Nerode equivalence classes. Instead, it achieves minimality as a byproduct of the composition of two independent transformations between models of the regular languages. Additionally, our work contributes to a deeper understanding of the effect that certain structural properties of regular expressions have on the structure of their resulting Brzozowski automaton.

Several questions remain open. Most importantly is if the minimality result for finite regular expression in head normal form can be extended to other classes of regular languages. More broadly, a classification of all classes of regular expressions that yield a minimal Brzozowski automaton remains open. Lastly, the study of the average size of expressions in head normal form, and of the average state complexity of the resulting minimal ADFAs, remains an interesting direction for future work.

References

[Ant96] Valentin Antimirov. "Partial derivatives of regular expressions and finite automaton constructions". In: *Theoretical Computer Science* 155.2 (1996), pp. 291–319. DOI: 10.1016/0304-3975(95)00182-4.

[Ard61] Dean N. Arden. "Delayed-logic and finite-state machines". In: *2nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1961)*. 1961, pp. 133–151. DOI: 10.1109/FOCS.1961.13.

[Bon+12] Filippo Bonchi et al. "Brzozowski's algorithm (co)algebraically". In: *Logic and Program Semantics: Essays Dedicated to Dexter Kozen on the Occasion of His 60th Birthday*. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 12–23. ISBN: 9783642294846. DOI: 10.1007/978-3-642-29485-3_2.

[Brz62] J. A. Brzozowski. "Canonical regular expressions and minimal state graphs for definite events". In: *Mathematical theory of Automata*. Vol. 12. 6. 1962, pp. 529–561.

[BM63] J. A. Brzozowski and E. J. McCluskey. "Signal Flow Graph Techniques for Sequential Circuit State Diagrams". In: *IEEE Transactions on Electronic Computers* EC-12.2 (1963), pp. 67–76. DOI: 10.1109/PGEC.1963.263416.

[Brz64] Janusz A. Brzozowski. "Derivatives of Regular Expressions". In: *J. ACM* 11.4 (Oct. 1964), pp. 481–494. ISSN: 0004-5411. DOI: 10.1145/321239.321249.

[Con12] J.H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall mathematics series. Dover Publications, 2012. ISBN: 9780486485836.

[Glu61] Victor Mikhaylovich Glushkov. "The abstract theory of automata". In: *Russian Mathematical Surveys* 16.5 (1961), pp. 1–53. DOI: https://doi.org/10.1070/RM1961v016n05ABEH004112.

[GH08] Hermann Gruber and Markus Holzer. "Provably Shorter Regular Expressions from Deterministic Finite Automata". In: *Developments in Language Theory*. Ed. by Masami Ito and Masafumi Toyama. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 383–395. DOI: 10.1007/978-3-540-85780-8_30.

[Hop71] John Hopcroft. "An n log n algorithm for minimizing states in a finite automaton". In: *Theory of Machines and Computations*. Ed. by Zvi Kohavi and Azaria Paz. Academic Press, 1971, pp. 189–196. ISBN: 978-0-12-417750-5. DOI: 10.1016/B978-0-12-417750-5.50022-1.

[HU79] John Hopcroft and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation*. 1st ed. Reading, Massachusetts: Addison-Wesley, 1979.

[IY03] Lucian Ilie and Sheng Yu. "Reducing NFAs by invariant equivalences". In: *Theoretical Computer Science* 306.1 (2003), pp. 373–390. ISSN: 0304-3975. DOI: https://doi.org/10.1016/S0304-3975(03)00311-6.

[Kle56] Stephen Cole Kleene. "Representation of Events in Nerve Nets and Finite Automata". In: *Automata Studies*. 1956, pp. 3–42. DOI: 10.1515/9781400882618-002.

[Koz94] D. Kozen. "A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events". In: *Information and Computation* 110.2 (1994), pp. 366–390. ISSN: 0890-5401. DOI: 10.1006/inco.1994.1037.

[Lot05] M. Lothaire. "Algorithms on Words". In: *Applied Combinatorics on Words*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2005, pp. 1–105.

[MMR14] Eva Maia, Nelma Moreira, and Rogério Reis. "Partial Derivative and Position Bisimilarity Automata". In: *Implementation and Application of Automata*. Ed. by Markus Holzer and Martin Kutrib. Cham: Springer International Publishing, 2014, pp. 264–

277. ISBN: 978-3-319-08846-4. DOI: 10.1007/978-3-319-08846-4_20.

[Mir67] B. G. Mirkin. "A new algorithm for constructing a base in a language of regular expressions". In: *1967.* (1967), pp. 161–170.

[Moo56] Edward F. Moore. "Gedanken-Experiments on Sequential Machines". In: *Automata Studies*. Ed. by C. E. Shannon and J. McCarthy. Princeton: Princeton University Press, 1956, pp. 129–154. ISBN: 9781400882618. DOI: 10.1515/9781400882618-006.

[ORT09] Scott Owens, John Reppy, and Aaron Turon. "Regular-expression derivatives re-examined". In: *Journal of Functional Programming* 19.2 (2009), pp. 173–190. DOI: 10.1017/S0956796808007090.

[Rev92] Dominique Revuz. "Minimisation of acyclic deterministic automata in linear time". In: *Theoretical Computer Science* 92.1 (1992), pp. 181–189. ISSN: 0304-3975. DOI: 10.1016/0304-3975(92)90142-3.

[Sak06] Jacques Sakarovitch. "The Language, the Expression, and the (Small) Automaton". In: *Implementation and Application of Automata*. Ed. by Jacques Farré, Igor Litovsky, and Sylvain Schmitz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 15–30. DOI: 10.1007/11605157_2.

[Sal66] Arto Salomaa. "Two Complete Axiom Systems for the Algebra of Regular Events". In: *J. ACM* 13.1 (Jan. 1966), pp. 158–169. DOI: 10.1145/321312.321326.

[Tho68] Ken Thompson. "Programming techniques: Regular expression search algorithm". In: *Communications of the ACM* 11.6 (1968), pp. 419–422. DOI: 10.1145/363347.363387.

[Yu97] Sheng Yu. "Regular Languages". In: *Handbook of Formal Languages*. 1997, pp. 41–110. DOI: 10.1007/978-3-642-59136-5_2.