Split and join for minimizing : Brzozowski’s algorithm

J.-M. Champarnaud¹, A. Khorsi², T. Paranthoën¹

¹ LIFAR, University of Rouen, France
{champarnaud,paranthoen}@dir.univ-rouen.fr

² CSD, University of Djilali Liabes, Sidi-Bel-Abbes, Algeria
ahmed_khorsi@lycos.com

Résumé

Brzozowski’s minimization algorithm is based on two successive determinization operations. There is a paradox between its (worst case) exponential complexity and its exceptionally good performance in practice. Our aim is to analyze the way the twofold determinization performs the minimization of a deterministic automaton. We give a characterization of the equivalence classes of \( \mathcal{A} \) w.r.t. the set of states of the automaton computed by the first determinization. The second determinization is expected to compute these equivalence classes. We show that it can be replaced by a specific procedure based on the classes characterization, which leads to a more efficient variant of Brzozowski’s algorithm.

Keywords : Finite automata, DFA minimization, Brzozowski’s algorithm.

1 Introduction

It is well known that given a regular language \( L \) over an alphabet \( \Sigma \) there exists a canonical deterministic automaton which recognizes \( L \), namely the minimal (deterministic) automaton of \( L \), whose states are the left quotients of \( L \) w.r.t. the words of \( \Sigma^* \). This automaton, denoted by \( \mathcal{A}_L \), is unique (up to an isomorphism) and it has a minimal number of states [13]. Moreover, it can be computed from any deterministic automaton recognizing \( L \) by merging states which have identical right languages. There exist numerous algorithms to minimize a deterministic automaton. Watson published a taxonomy on this topic [18].
Among the various possible constructions, Brzozowski’s minimization algorithm [3] is of a specific interest, regarding to several criteria which are discussed below. Let us first recall how it works. Let $\mathcal{A}$ be a (non necessarily deterministic) automaton. $d(\mathcal{A})$ be the subset automaton of $\mathcal{A}$ and $r(\mathcal{A})$ be the reverse automaton of $\mathcal{A}$. Brzozowski’s algorithm is based on the following theorem:

$$\mathcal{A}_L = d(r(d(r(\mathcal{A}))))$$

This is a deep result since it relates DFA minimization to a basic operation, the determinization one. Let us mention that it has been generalized by Mohri to the case of bideterminizable transducers defined on the tropical semiring [12]. Brzozowski’s theorem is also a fundamental tool for the computation of the nondeterministic minimal automata of a regular language. Let us cite the implementation [6] of the canonical automaton $\mathcal{C}_L$ defined by Carrez [4, 1] and the construction of the fundamental automaton $\mathcal{F}_L$ by Matz and Potthoff [11].

We are here especially interested by algorithmic and complexity features. Watson used the fact that Brzozowski’s algorithm can take a nondeterministic automaton as input to design an algorithm which directly constructs a minimal deterministic automaton from a regular expression [19]. Since our aim is to study the way Brzozowski’s algorithm performs a minimization, we will essentially consider the case when the initial automaton is a deterministic one. The paradox is the following: since Brzozowski’s algorithm performs two determinizations, its (worst case) complexity is exponential w.r.t. the number of states of the initial automaton; nevertheless, as reported by Watson [18], Brzozowski’s algorithm has proved to be exceptionally good in practice, usually out-performing Hopcroft’s algorithm [7] significantly. Let us add that the average complexity of the algorithm has been proved to be exponential for group automata, although they likely are a favourable case since they are both deterministic and codeterministic [14].

Our contribution is the following. Let $\mathcal{A}$ be a deterministic automaton. We give a characterization of the equivalence classes of $\mathcal{A}$ w.r.t. the set of states of $d r(\mathcal{A})$, that is after the first determinization. The second determinization is expected to compute these equivalence classes. We show it can be replaced by a specific procedure based on the classes characterization, which leads to a more efficient variant of Brzozowski’s algorithm.

Next section recalls some useful notations and definitions of automata theory. Section 3 is especially devoted to determinization and minimization operations. Section 4 presents Brzozowski’s minimization algorithm and its proof. Section 5 provides an original analysis of the algorithm and the variant it leads to.
2 Preliminaries

Let us first review basic notions and terminology concerning finite automata and regular languages. For further details, classical books [2, 8] or handbooks [20] are excellent references.

Let \( \Sigma \) be a non-empty finite set of symbols, called the alphabet. Symbols are denoted by \( x_1, x_2, \ldots, x_m \). A word \( u \) over \( \Sigma \) is a finite sequence \( (y_1, y_2, \ldots, y_n) \) of symbols, usually written \( y_1 y_2 \ldots y_n \). The length of a word \( u \), denoted \( |u| \) is the number of symbols in \( u \). The empty word denoted by \( \varepsilon \) has a zero length. If \( u = y_1 y_2 \ldots y_n \) and \( v = z_1 z_2 \ldots z_p \) are two words over \( \Sigma \), their concatenation \( u \cdot v \), usually written \( uv \), is the word \( y_1 y_2 \ldots y_n z_1 z_2 \ldots z_p \). The set of all the words over \( \Sigma \) is denoted \( \Sigma^* \). A language over \( \Sigma \) is a subset of \( \Sigma^* \). The operations of union, concatenation and star over the subsets of \( \Sigma^* \) are called regular operations. The regular languages over \( \Sigma \) are the languages obtained from the finite subsets of \( \Sigma^* \) by using a finite number of regular operations.

A (finite) automaton is a 5-tuple \( \mathcal{M} = (Q, \Sigma, \delta, I, F) \) where \( Q \) is a (finite) set of states, \( \Sigma \) is a finite alphabet, \( I \subseteq Q \) is the set of initial states, \( F \subseteq Q \) is the set of final states, and \( \delta \) is the transition function. The automaton \( \mathcal{M} \) is deterministic (\( \mathcal{M} \) is a DFA) if and only if \( |I| = 1 \) and \( \delta \) is a mapping from \( Q \times \Sigma \) to \( Q \). Otherwise \( \mathcal{M} \) is a NFA and \( \delta \) is a mapping from \( Q \times \Sigma \) to \( 2^Q \). The automaton \( \mathcal{M} \) is complete if and only if \( \delta \) is a full mapping. A path of \( \mathcal{M} \) is a sequence \( (q_i, a_i, q_{i+1}), i = 1, \ldots, n \), of consecutive edges. Its label is the word \( w = a_1 a_2 \ldots a_n \). A word \( w = a_1 a_2 \ldots a_n \) is recognized by the automaton \( \mathcal{M} \) if there is a path with label \( w \) such that \( q_1 \in I \) and \( q_n+1 \in F \). The language \( L(\mathcal{M}) \) recognized by the automaton \( \mathcal{M} \) is the set of words which it recognizes. Two automata \( \mathcal{M} \) and \( \mathcal{M}' \) are equivalent if and only if they recognize the same language. A state is accessible (resp. coaccessible) if and only if there is a path from an initial state to this state (resp. from this state to a final state). An automaton is trim if and only if all its states are both accessible and coaccessible.

Kleene’s theorem [10] states that a language is regular if and only if it is recognized by a finite automaton.

Let \( q \) be a state of \( \mathcal{A} = (Q, \Sigma, \delta, i, F) \). The right language of \( q \) is the language \( L^R_q(q) \) (written \( L^R_q(q) \) if not ambiguous) recognized by the automaton \( \mathcal{A}_d(q) = (Q, \Sigma, \delta, q, F) \) obtained from \( \mathcal{A} \) by making \( q \) the unique initial state. The left language of \( q \) is the language \( L^L_q(q) \) (written \( L^L_q(q) \) if not ambiguous) recognized by the automaton \( \mathcal{A}_l(q) = (Q, \Sigma, \delta, i, q) \) obtained from \( \mathcal{A} \) by making \( q \) the unique final state. We will use the following proposition:

**Proposition 1** An automaton is deterministic if and only if the left languages of its states are pairwise disjoint.
The reverse $r(u)$ of the word $u$ is defined as follows: $r(\varepsilon) = \varepsilon$ and, if $u = u_1u_2\ldots u_p$, then $r(u) = v_1v_2\ldots v_p$, with $v_i = u_{p-i+1}$, for all $i$ from 1 to $p$. The reverse of the language $L$ is the language $r(L) = \{u \mid r(u) \in L\}$. The reverse of the automaton $A = (Q, \Sigma, \delta, I, F)$ is the automaton $r(A) = (Q, \Sigma, r(\delta), F, I)$, obtained from $A$ by swapping the role of initial and final states and by reversing the transitions.

We will use the following propositions, where $A$ is a trim automaton:

**Proposition 2** If $A$ recognizes the language $L$ then $r(A)$ recognizes the language $r(L)$.

**Proposition 3** If the left (resp. right) language of the state $q$ in $A$ is $L_l(q)$ (resp. $L_r(q)$), then its left (resp. right) language in $r(A)$ is $L_d(q)$ (resp. $L_g(q)$).

### 3 Determinization and minimization operations

#### 3.1 Determinization

**Definition 1** Let $A = (Q, \Sigma, \delta, I, F)$ be a NFA. The subset-automaton of $A$ is the automaton $d(A) = (Q', \Sigma, \delta', \{q'\}, F')$ defined as follows [8, 20]:

- **Set of states**: A deterministic state is a set of nondeterministic states; for all $q'$ in $Q'$, we have $q' \subseteq Q$.
- **Initial state**: The initial state in $d(A)$ is the set $I$ of initial states in $A$.
- **Set of transitions**: Let $q'$ be a deterministic state and $a$ be a symbol in $\Sigma$. If the transition from $q'$ on symbol $a$ is defined, then, by construction, its target is the state $\delta'(q', a)$ such that:

$$\delta'(q', a) = \bigcup_{q \in q'} \delta(q, a).$$  \hfill (1)

- **Set of final states**: A deterministic state is final if and only if it contains at least one final nondeterministic state: $q' \in F' \iff q' \cap F \neq \emptyset$.

We will use the following proposition:

**Proposition 4** The right language of a state $q'$ of $d(A)$ is equal to the union of the right languages of the states $q$ of $A$ belonging to the subset $q'$.

Let $n$ (resp. $n'$) be the number of states in $A$ (resp. in $d(A)$). As stated by Rabin and Scott [16], the upper bound $n' \leq 2^n - 1$ can be reached. Moreover, the automaton $d(A)$ can be computed with the following complexity [15, 5]: $O(\sqrt{n}2^{2n})$ when using lists, and $O(n^2(\log n)2^n)$ when using balanced search trees.
3.2 Minimization

The (left) quotient of a regular language \( L \) w.r.t. a word \( u \) of \( \Sigma^* \) is the language \( u^{-1}L = \{ v \in \Sigma^* \mid uv \in L \} \). The minimal automaton \( \mathcal{A}_L \) of a regular language \( L \) is defined as follows:

- the set of states is the set of quotients of \( L \),
- the initial state is \( L \),
- the final states are the quotients which contain the empty word,
- the transition function is such that \( \delta(u^{-1}L, x) = (ux)^{-1}L \).

The automaton \( \mathcal{A}_L \) is unique up to an isomorphism and it has a minimal number of states [13]. We will use the following proposition:

**Proposition 5** A (deterministic, complete, accessible) automaton is minimal if and only if the right languages of its states are all different.

The automaton \( \mathcal{A}_L \) can be computed from any deterministic automaton recognizing \( L \) by merging states which are equivalent w.r.t. Nerode equivalence:

\[
s \equiv t \iff [s \cdot u \in F \leftrightarrow t \cdot u \in F, \forall u \in \Sigma^*]
\]

Computing Nerode equivalence can be realized with a \( O(n^2) \) complexity [13]. Using the notion of coarsest partition leads to a complexity of \( O(n \log(n)) \) [7].

4 Brzozowski’s minimization algorithm

Let \( \mathcal{A} \) be an automaton. Let \( d(\mathcal{A}) \) (resp. \( r(\mathcal{A}) \)) be the subset automaton (resp. the reverse automaton) of \( \mathcal{A} \). We will write \( dr(\mathcal{A}) \) for \( d(r(\mathcal{A})) \), \( rdr(\mathcal{A}) \) for \( r(d(r(\mathcal{A}))) \) and \( ddrdr(\mathcal{A}) \) for \( d(r(d(r(\mathcal{A})))) \).

Brzozowski’s algorithm is based on the following theorem [3] :

**Theorem 1** (Brzozowski, 1962) Given a (non necessarily deterministic) automaton \( \mathcal{A} \) recognizing a regular language \( L \), the minimal deterministic automaton \( \mathcal{A}_L \) of \( L \) can be computed by the formula:

\[
\mathcal{A}_L = ddrdr(\mathcal{A})
\]

**Proof.** The proof is based on Propositions (1)-(5). By construction, the automaton \( ddrdr(\mathcal{A}) \) is deterministic, complete and accessible. From Proposition (2) it recognizes the language \( L \). Let us show that the right languages of \( ddrdr(\mathcal{A}) \) are all distinct. From Proposition (1) the left languages of \( dr(\mathcal{A}) \) are pairwise disjoint. From Proposition (3) the right languages of \( rdr(\mathcal{A}) \) are the left languages of \( dr(\mathcal{A}) \). Therefore they are pairwise disjoint. From Proposition (4) a right language of \( ddrdr(\mathcal{A}) \) is a union of right languages of \( rdr(\mathcal{A}) \). Since the right languages of
are pairwise disjoint, the right languages of \( \text{drdr} \) are all distinct. Thus, by Proposition (5) the automaton \( \text{drdr} \) is minimal.

\[ \]

5 Analysis of Brzozowski’s algorithm

5.1 Split and join for minimizing

Let \( \mathcal{A} \) be an automaton which recognizes a regular language \( L \). We study the transformation of the sequence \( S_d = (L_d^A(q))_{q \in Q} \) of the right languages of the states of \( \mathcal{A} \), when the twofold determinization is performed:

\[ S_d \rightarrow_{\text{dr}} S_d^1 \rightarrow_{\text{dr}} S_d^2 \]

Notice that since the languages of \( S_d^1 \) are pairwise disjoint and the languages of \( S_d^2 \) are all distinct, \( S_d^1 \) and \( S_d^2 \) are sets. Let us remind that the right language of a state is a (left) quotient of \( L \) if \( \mathcal{A} \) is deterministic and a subset of the intersection of some (left) quotients of \( L \) if \( \mathcal{A} \) is nondeterministic. The first determinization splits the right languages of \( \mathcal{A} \) into disjoint pieces, whereas the second one joins the pieces in order to recombine the set of (left) quotients of \( L \). The effect of the twofold determinization is illustrated by the Example 1. This example is intentionally simple: the initial automaton is deterministic and even minimal.

Example 1

Let \( q_1 \) and \( q_2 \) be two states of \( \mathcal{A} \). We suppose that there exist three distinct words, \( u, v \) and \( w \) such that:

\[ L_d^A(q_1) = \{u, v\}, \quad L_d^A(q_2) = \{v, w\}, \quad \{q \mid u \in L_d^A(q_1)\} = \{q_1\}, \quad \{q \mid w \in L_d^A(q_2)\} = \{q_2\} \]

We suppose that there exist two distinct words, \( s \) and \( t \) such that:

\[ L_d^A(q_2) = \{t\}, \quad \{q \mid s \in L_d^A(q)\} = \{q_1\} \]

The determinization of \( \text{r} \) produces the three states \( q_1^1 \), \( q_2^1 \) and \( q_3^1 \) of \( \text{dr} \) such that:

\[ q_1^1 = \{q_1\}, \quad q_2^1 = \{q_2\} \]

The right languages of \( q_1^1 \), \( q_2^1 \) and \( q_3^1 \) in \( \text{drd} \) are pairwise disjoint (they are respectively equal to \( \{u\}, \{w\} \) and \( \{v\} \)).

The effect of the first determinization is that the two right languages \( \{u, v\} \) and \( \{v, w\} \) of \( \mathcal{A} \) have been split into three right languages in \( \text{drd} \): \( \{u\}, \{w\} \) and \( \{v\} \).

Notice that the left languages of \( q_1^1 \), \( q_2^1 \) and \( q_3^1 \) in \( \text{rd} \) are respectively equal to \( \{s\}, \{t\} \) and \( \{s, t\} \) and thus all distinct. This is due to the fact that \( \mathcal{A} \) is deterministic (see Proposition (6)).
The determinization of \( \text{rdr}(A) \) produces the two states \( q''_1 \) and \( q''_2 \) of \( \text{rdr}(A) \) such that: \( q''_1 = \{ q'_1, q'_2 \} \) and \( q''_2 = \{ q'_3, q'_4 \} \). The right languages of \( q''_1 \) and \( q''_2 \) in \( \text{rdr}(A) \) are distinct (they are respectively equal to \( \{ u, v \} \) and \( \{ v, w \} \)).

The effect of the second determinization is that the three right languages \{u\}, \{w\} and \{v\} of \( \text{rdr}(A) \) have been joined into two right languages in \( \text{rdr}(A) \) : \{u, v\} and \{v, w\}.

5.2 The deterministic case

Brzozowski’s algorithm can be applied to a nondeterministic automaton. Here we focus on the case when \( A \) is deterministic. Proposition (6) is due to Brzozowski [3]. Proposition (8) and Corollary (1) are very likely not original. These propositions are gathered in this section for sake of completeness.

**Proposition 6** If \( A \) is deterministic, then \( \text{dr}(A) \) is the minimal automaton of \( r(L) \).

**Proof.** Since \( A \) is deterministic, its left languages are pairwise disjoint, and so are the right languages of \( r(A) \). The right languages of \( \text{dr}(A) \), which are unions of right languages of \( r(A) \), are therefore all distinct.

**Proposition 7** If \( A \) is deterministic, then a state of \( \text{rdr}(A) \) is a union of Nerode equivalence classes of the automaton \( A \).

**Proof.** The transition function of \( r(A) \) is denoted by \( \delta_r \). Let \( q_1 \) and \( q_2 \) be two states of \( A = (Q, \Sigma, \delta, i, F) \). We have:

\[
q_1 = q_2 \iff [L^A_{\delta_r}(q_1) = L^A_{\delta_r}(q_2) \iff F^r_r[A](q_1) = F^r_r(A)(q_2)]
\]

Let \( q' \) be a state of \( \text{dr}(A) \). By construction, there exists a word \( u \) of \( \Sigma^* \) such that \( q' = \delta_r(F, u) \). We have: \( q \in \delta_r(F, u) \iff u \in L^r_{\delta_r}(q) \). Therefore, \( q_1 \) and \( q_2 \) are equivalent if and only if they are such that: \( q_1 \in \delta_r(F, u) \iff q_2 \in \delta_r(F, u) \). Thus, a state of \( \text{rdr}(A) \) is a union of equivalence classes of states in \( A \).

**Corollary 1** Let \( A \) be a deterministic automaton recognizing a regular language \( L \). Let \( n \) be the number of states of \( A \). Let \( r \) be the number of (left) quotients of \( L \). Then the deterministic complexity of \( r(A) \) is \( 2^r \leq 2^n \).

The following proposition leads to a characterization of the equivalence classes of \( A \). It says that two states \( p \) and \( q \) of \( A \) are equivalent if and only if they belong to the same states of \( \text{dr}(A) \). This property can be seen as a corollary of Proposition (8).
Proposition 8  Let $p$ and $q$ be two states of $\mathcal{A}$. It holds:

$$ p \equiv q \iff [p \in P \iff q \in P, \forall P \in Q_{dr}(\mathcal{A})] $$

Proof. We have: $p \equiv q \iff [u \in L_d(p) \iff u \in L_d(q), \forall u \in \Sigma^*]$. Moreover $L_d(p) = \bigcup_{P \in P} r(L_g(P))$, with $P \in Q_{dr}(\mathcal{A})$. Hence the result.

6  A variant of Brzozowski’s minimization algorithm

We still assume that $\mathcal{A}$ is deterministic. We show that the Proposition (8) leads to an original computation of the equivalence classes of the states of $\mathcal{A}$ after the determinization of $r(\mathcal{A})$ is achieved. On the one hand this result allows us to have a better understanding of how Brzozowski’s algorithm performs the minimization: the second determinization actually is a state-equivalence-based procedure. On the other hand it yields a variant of Brzozowski’s minimization algorithm, where the second determinization is replaced by a more efficient computation of the equivalence classes.

The Algorithm 1 computes the equivalence classes of $\mathcal{A}$. The partition of $Q$ initially contains two sets: $Q - F$ and $F$. At each step of the algorithm, a set $Y$ of the current partition contains possibly equivalent states, in the sense that so far they belong to the same states of $dr(\mathcal{A})$. Every time a new state $X$ of $dr(\mathcal{A})$ is processed, it is checked w.r.t. every set of the partition in order to detect sets containing non-equivalent states of $\mathcal{A}$. 
1. Begin
2. Partition $\leftarrow \{Q - F, F\}$
3. Waiting $\leftarrow \{F\}$
4. While Waiting $\neq \emptyset$ do begin
5. $X \leftarrow \text{First}(\text{Waiting})$
6. Waiting $\leftarrow \text{Waiting} - \{X\}$
7. Processed $\leftarrow \text{Processed} \cup \{X\}$
8. for all $a \in \Sigma$ do begin
9. $Z \leftarrow \delta_r(X, a)$; if $Z \not\in \text{Processed}$ then Waiting $\leftarrow \text{Waiting} \cup Z$
10. end
11. for all $Y \in \text{Partition}$ do begin
12. $K \leftarrow X \cap Y$
13. if $K \neq \emptyset$ then Partition $\leftarrow \text{Partition} \cup K$
14. if $X \not\subseteq Y$ then Partition $\leftarrow \text{Partition} \cup (X - K)$
15. if $Y \not\subseteq X$ then Partition $\leftarrow \text{Partition} \cup (Y - K)$
16. end
17. end
18. end

Algorithm 1: Algorithm to extract equivalence classes of $\mathcal{A}$.

The complexity of the Algorithm 1 is exponential since it contains the determinization of $r(\mathcal{A})$. However it is likely more efficient to extract equivalence classes on the fly than performing a second determinization.

7 Conclusion

Brzozowski’s minimization algorithm is both simple and mysterious. It is based on two basic and easily understandable operations. However the behaviour of the algorithm is not so obvious. Its average complexity and experimental performance are still unknown or unexplained. This short analysis is intended to contribute to a better understanding of how this algorithm performs the minimization. In particular it shows that the place of Brzozowski’s algorithm, in a taxonomy such as Watson’s one, is among minimization algorithms based on the computation of a state equivalence.
Références


