Document downloaded from:

http://hdl.handle.net/10251/64262

This paper must be cited as:

García Gómez, P.; López Rodríguez, D.; Vázquez-De-Parga Andrade, M. (2015). DFA minimization: Double reversal versus split minimization algorithms. Theoretical Computer Science. 583:78-85. doi:10.1016/j.tcs.2015.04.002.

The final publication is available at

http://dx.doi.org/10.1016/j.tcs.2015.04.002

Copyright Elsevier

Additional Information

“NOTICE: this is the author’s version of a work that was accepted for publication in Theoretical Computer Science. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication. A definitive version was subsequently published inTheoretical Computer Science, [Volume 583, 7 June 2015, Pages 78–85] DOI 10.1016/j.tcs.2015.04.002

DFA minimization: double reversal versus split minimization algorithms

Pedro García, Damián López and Manuel Vázquez de Parga
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
{pgarcia,dlopez,mvazquez}@dsic.upv.es

Abstract

In this paper, we show the relationship between the two most widely used approaches for the minimization of deterministic finite automata: minimization by split of partitions and minimization by double reversal. Even though the double reversal approach has usually been considered to be unconventional with respect to the more common split approach, we show that any double reversal minimization algorithm can be related to a split minimization algorithm and vice versa.

Keywords: DFA minimization; atomic automaton; double reversal algorithms; split minimization algorithms

1 Introduction

The minimization of deterministic automata is a classic issue in Computer Science that still plays an important role in obtaining efficient solutions for certain problems in fields such as text processing, image analysis, and linguistics.

The minimization of deterministic finite automata (DFA) is based on the computation of Nerode's equivalence relation for the language that is accepted by the automaton to be minimized. In order to do this, two main approaches have been used. The first approach includes methods that iteratively refine an initial partition that distinguishes final and non-final states [1, 2, 3]. Among these algorithms, the algorithm by Hopcroft is of special interest because it is the algorithm that offers the best time complexity ($\mathcal{O}(n \log n)$, where n stands for the number of states of the input automaton). Recently, Berstel et al. [4] named the operation on which the minimization is based, the *split* operation. This operation formalizes the set (the *splitter*) that is used to refine a given partition. Their work in [4] describe the most important of these methods. Note that, for obvious reasons, the splitters are usually obtained taking into account the transition function of the automaton to be minimized. Nevertheless, in order to consider any potential algorithm that uses this approach, we will consider general splitters regardless of how they are obtained.

The second approach is based on Brzozowski's double reversal algorithm [5], which receives any automaton (regardless of whether or not it is deterministic) and outputs the minimal DFA for the language. Despite the fact that the algorithm has an exponential time complexity in the worst case (even when the input is a DFA), the algorithm has always sparked interest. The implementation of the algorithm is very straightforward and, in essence, alternates reverse operation and determinization operation twice. In other words, for any automaton A, the process can be described as $D(R(D(R(A))))$, where D and R stand for the determinization and reverse operations, respectively. The correctness proof is based on the fact that, for any DFA A, the automaton $D(R(A))$ is the minimal DFA for the reverse language of $L(A)$ [5].

Brzozowski's algorithm can be viewed as a special case within the framework that was recently proposed by Brzozowski and Tamm [6]. In their framework, the first determinization of the double reversal algorithm is substituted by any algorithm that returns an atomic automaton that accepts the reverse of the language (instead of the computation of the DFA for the reverse language). Thus, the framework can be summarized as $D(R(Atom(R(A))))$, where $Atom$ stands for any process that outputs an atomic automaton. In [6], Brzozowski and Tamm do not propose any algorithm to compute the atomization of an automaton. Recently in [7], we proposed a polynomial-time algorithm that, for any DFA A, outputs an atomic automaton that accepts the reverse language of $L(A)$. This implies that, when the problem is restricted to DFA, it is possible to implement the double-reversal minimization of the input automaton in polynomial time.

It is relevant to mention the difficulties that the community has had to connect both approaches to automata minimization. The paper by Champarnaud et al. [8] can be seen as a first attempt to relate Brzozowski's algorithm with split minimization methods. In their paper, the authors propose an algorithm that computes the first step of Brzozowski's algorithm (the computation of $D(R(A))$) and takes into account the set of states of the resulting automaton to sequentially split the initial partition of the set of states of A. The correctness proof of this algorithm considers that p and q are two equivalent states in A if and only if, for any state P of $D(R(A))$, it holds that $p \in P$ if and only if $q \in P$. Therefore, any split that is carried out using the states of $D(R(A))$ will keep the states p and q as equivalent. Despite its exponential behavior in the worst case, this algorithm is interesting because basically it rewrites Brzozowski's algorithm within the split paradigm.

In this paper, we show the relationship between the two approaches. To do this, we first consider any possible split minimization algorithm and take into account the sequence of partitions obtained during the minimization process of any input DFA A. We prove that this information suffices to obtain an atomic automaton that accepts the reverse of the language $L(A)$. Second, we note that the double reversal approach is determined by the atomic automaton used. Thus, we consider any possible atomic automaton B, and we prove that we can obtain a set of splitters (and therefore a sequence of partitions) that allows any DFA A that accepts the reverse of the language $L(B)$ to be minimized.

2 Notation and definitions

Let Σ be a finite alphabet and let Σ^* be the free monoid generated by Σ with concatenation as the internal operation and the empty string λ as neutral element. A *finite automaton* is a 5-tuple $A = (Q, \Sigma, \delta, I, F)$, where Q is a finite set of states, Σ is an alphabet, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states and $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is the transition function, which can also be seen as $\delta \subseteq (Q \times \Sigma \times Q)$. The transition function can be extended in a natural way to Σ^* and $\mathcal{P}(Q)$. We will denote the language accepted by an automaton $A = (Q, \Sigma, \delta, I, F)$ with $L(A)$, which is defined as the set of $x \in \Sigma^*$ such that $\delta(I, x) \cap F \neq \emptyset$.

Given an automaton $A = (Q, \Sigma, \delta, I, F)$ and a state $q \in Q$, we define the *right language* of q (denoted by $\overrightarrow{L}_q^A$) as the language $\{x \in \Sigma^* \ : \ \delta(q, x) \cap F \neq \emptyset\}$. In a similar way, we define the *left language* of a state q (denoted by $\overleftarrow{L}_q^A$) as the language $\{x \in \Sigma^* \ : \ q \in \delta(I, x)\}$. When necessary, we will refer to the strings in $\overleftarrow{L}_q^A$ as *representatives* of the state q. We also will consider the natural extension of these languages to $\mathcal{P}(Q)$.

Given an automaton A, we say that it is accessible if, for each $q \in Q$, there exists a string x such that $q \in \delta(I, x)$. An automaton is called *deterministic* (DFA) if, for every state q and every symbol a, the number of transitions $\delta(q, a)$ is at most one, and it has only one initial state. A DFA is said to be complete whenever the size of the set $\delta(q, a)$ is always one. In the following, we consider only complete and accessible DFA. The minimal DFA for any given regular language L is the DFA with the minimum number of states. We recall that each state of the minimal DFA relates to one class of Nerode's relation for the language.

Given an automaton $A = (Q, \Sigma, \delta, I, F)$ that accepts a language L, the reverse automaton is defined as the automaton $R(A) = (Q, \Sigma, \delta^{-1}, F, I)$, where $q \in \delta^{-1}(p, a)$ if and only if $p \in \delta(q, a)$. It is known that $L(R(A)) = L^r$, where L^r denotes the reverse language of L. For any automaton $A = (Q, \Sigma, \delta, I, F)$, it is known that the automaton $A' = (2^Q, \Sigma, \delta', I, F')$, where $F' = \{P \in 2^Q \ : \ P \cap F \neq \emptyset\}$, and $\delta'(P, a) = \cup_{p \in P} \delta(p, a)$, is a DFA that is equivalent to A. Let us denote the accessible version of A' by $D(A)$.

Given a regular language L over Σ, the (left) quotient of L by a string u is defined as the language $u^{-1}L = \{v \in \Sigma^* \ : \ uv \in L\}$. It is well known that, by Nerode's theorem, the set of quotients of a regular language is finite; let us denote this set with $\{L_1, L_2, \dots, L_n\}$. In [6], Brzozowski and Tamm consider this set to define an *atom* A of L as any of the nonempty languages of the form $\widetilde{L_1} \cap \widetilde{L_2} \cap \dots \cap \widetilde{L_n}$, where $\widetilde{L_i}$ (which we will refer to as a *literal*) is either L_i or $\overline{L_i}$.

In the same work, Brzozowski and Tamm define the *átomaton* of a given language L as the automaton in which each state represents an atom of L. For any DFA A the átomaton for $L(A)$ can be obtained by computing $R(D(R(A)))$. In the same way, an NFA is *atomic* if the right language of each of its states is a union of atoms. Let us also recall that in [6], Brzozowski and Tamm also prove that an automaton A is atomic if and only if $D(R(A))$ is the minimal DFA for the language $L(A)^r$. Note that this implies that any DFA is also atomic because $D(R(D(R(A))))$ is the minimal DFA for L(A) according to Brzozowski's algorithm. Also note that any DFA A is atomic because, for any state q of A, the language $\overrightarrow{L}_q^A$ is a quotient and can therefore be obtained by union of the atoms that include the corresponding quotient uncomplemented. This will be important in proving our results, especially the correctness of Algorithm 3.1.

A *partition* π of a set Q is a set $\{P_1, P_2, \dots, P_k\}$ of pairwise disjoint non-empty subsets of Q such that $Q = \cup_{1 \leq i \leq k} P_i$. We refer to those subsets as *blocks*, and we denote the block of π that contains p by $B(p, \pi)$. A partition π is refined by π' (π is coarser than π') if each block in π' is contained in some block in π. We denote this as $\pi \geq \pi'$.

Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ and a partition π over Q, we define the *quotient automaton* $A/\pi = (\pi, \Sigma, \delta', B(q_0, \pi), F')$, where $F' = \{P_i \in \pi \ : \ P \cap F \neq \emptyset\}$ and where $P_j \in \delta'(P_i, a)$ if there exist two states $p \in P_i$, $q \in P_j$ such that $\delta(p, a) = q$.

Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$, we are interested in the sequences of partitions $\langle \pi_1 = \{F, Q - F\}, \pi_2, \dots, \pi_k \rangle$ of Q that fulfill that A/π_k is the minimal DFA for $L(A)$ and such that $\pi_i \geq \pi_{i+1}$ for each $1 \leq i < k$. We will call such a sequence a *convergent sequence of partitions* (for the automaton A).

Let us note that, given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ and a convergent sequence of partitions $\mathscr{P} = \langle \pi_1, \pi_2, \dots, \pi_k \rangle$ for A, it holds that each block P in the sequence of partitions $\mathscr{P}$ is the union of some blocks of π_k (if it were not, then the sequence of partitions $\mathscr{P}$ would not be convergent).

Let π_1 and π_2 be two partitions of Q, and let us denote the coarsest partition that refines both π_1 and π_2 by $\pi_1 \wedge \pi_2$. The blocks of this partition are the non-empty sets in $P_1 \cap P_2$, where $P_1 \in \pi_1$ and $P_2 \in \pi_2$.

Note that the algorithms that solve the minimization of an automaton by the successive refinement of its set of states, in essence, take into account the transition function in order to obtain a set (of states) that allows such a refinement. In order to unify the notation, in [4], the authors propose the notion of *splitter*. Thus, for any given DFA $A = (Q, \Sigma, \delta, q_0, F)$, any $P, R \subset Q$, and $a \in \Sigma$, the authors define a splitter as the set $\delta^{-1}(P, a)$, which allows the *split* of the set R into the sets $R' = \delta^{-1}(P, a) \cap R$ and $R'' = R - R'$. The result of the split is denoted $(P, a)|R$. Whenever $\delta^{-1}(P, a) \cap R = \emptyset$ or $\delta^{-1}(P, a) \cap R = R$, it is said that the splitter $\delta^{-1}(P, a)$ does not split R which is denoted with $(P, a)|R = R$.

We extend this notation in order to consider splitters that may be obtained by any other processes. Thus, for any given sets such that $P, R \subset Q$, we will denote the split of the set R into the sets $P \cap R$ and $R - P$ as $(P)|R$, and we will say that the splitter (P) splits the set R whenever both $P \cap R$ and $R - P$ are non-empty.

3 From split to double reversal minimization algorithms

Given a regular language L and any DFA A such that $L(A) = L$, we consider any minimization algorithm that refines the initial partition of states of the automaton by successive split (for instance, [2, 1]). We do not restrict the procedures that any algorithm may use to split any given partition.

In order to prove the relationship between the two minimization approaches, we first propose Algorithm 3.1. This algorithm, for any given DFA A and using any convergent sequence of partitions obtained by any split minimization algorithm, outputs an atomic automaton B such that $L(B) = L(A)^r$. The algorithm builds an automaton that initially considers each block P in the convergent sequence of partitions provided as a state (note that $P \subseteq Q_A$). Then, the algorithm iterates over the sequence of partitions and considers any state P in the current partition such that, for some $a \in \Sigma$, $\delta_B(P, a)$ is undefined. In those cases, if $\delta_A^{-1}(P, a)$ can be covered using the blocks in the next partition, then a transition is added from P into B using the symbol a to each one of these blocks. The final partition is used to define any undefined transition that may remain. Example 1 illustrates the behavior of the proposed algorithm.

Example 1 *Let us consider the DFA in Figure 1 and the following convergent sequence of partitions:*

$$\pi_1 = \{\{1, 2, 3, 4, 7\}, \{5, 6\}\}$$
$$\pi_2 = \{\{1, 2, 7\}, \{3, 4\}, \{5, 6\}\}$$
$$\pi_3 = \{\{1, 2\}, \{3, 4\}, \{5, 6\}, \{7\}\}$$
$$\pi_4 = \{\{1, 2\}, \{3\}, \{4\}, \{5, 6\}, \{7\}\}$$

Initially, the algorithm obtains the set of states as well as the sets of initial and final states. These sets are shown below.

Algorithm 3.1 Algorithm to obtain an atomic automaton from a convergent sequence of partitions.

```
Input: A DFA $A = (Q_A, \Sigma, \delta_A, q_0, F_A)$
Input: A convergent sequence of partitions $\mathscr{P} = \langle \pi_1, \pi_2, \ldots, \pi_n \rangle$ of $Q_A$
Output: An atomic automaton $B = (Q_B, \Sigma, \delta_B, I, F_B)$ that accepts $L(A)^r$
 1: Method
 2: $Q_B = \bigcup_{i=1}^{n} \pi_i$        // the set of the different blocks in any partition of $\mathscr{P}$
 3: $I = \{F_A\}$
 4: $F_B = \{P \in Q_B \ : \ q_0 \in P\}$
 5: $\delta_B = \emptyset$
 6: for $i = 1$ to $n - 1$ do
 7:    for each $a \in \Sigma$ and every block $P \in \pi_i$ do
 8:       if $\delta_B(P, a)$ is undefined and $\delta_A^{-1}(P, a)$ can be covered using the
          blocks in $\pi_{i+1}$ then
 9:          Let $\mathcal{P}$ be the cover of $\delta_A^{-1}(P, a)$ using the blocks in $\pi_{i+1}$
10:          for each block $P' \in \mathcal{P}$ do
11:             Add the transition $(P, a, P')$ to $\delta_B$
12:          end for
13:       end if
14:    end for
15: end for
16: for each $P \in Q_B$ and $a \in \Sigma$ such that $\delta_B(P, a)$ is undefined do
17:    Let $\mathcal{P}$ be the cover of $\delta_A^{-1}(P, a)$ taking into account the blocks in $\pi_n$
18:    for each $P' \in \mathcal{P}$ do
19:       Add the transition $(P, a, P')$ to $\delta_B$
20:    end for
21: end for
22: return the accessible version of the automaton $(Q_B, \Sigma, \delta_B, I, F_B)$
23: End Method.
```

$$Q = \{\{1,2,3,4,7\}, \{5,6\}, \{1,2,7\}, \{3,4\}, \{1,2\}, \{7\}, \{3\}, \{4\}\}$$
$$F = \{\{1,2,3,4,7\}, \{1,2,7\}, \{1,2\}\}$$
$$I = \{\{5,6\}\}$$

The first iteration of the loop at line 6 considers the partition π_1. Note that: $\delta_A^{-1}(\{5,6\}, a) = \emptyset$ can be discarded; $\delta_A^{-1}(\{5,6\}, b) = \{3,4,7\}$, which cannot be covered using the blocks in π_2; $\delta_A^{-1}(\{1,2,3,4,7\}, a) = \{1,2,3,4,5,6,7\}$, which can be covered using the blocks $\{1,2,7\}$, $\{3,4\}$, and $\{5,6\}$; and, finally, $\delta_A^{-1}(\{1,2,3,4,7\}, b) = \{1,2,5,6\}$, which cannot be covered using the blocks in π_2. When the iteration ends, the following transitions have been added:

$$\delta_B(\{1,2,3,4,7\}, a) = \{\{1,2,7\}, \{3,4\}, \{5,6\}\}$$

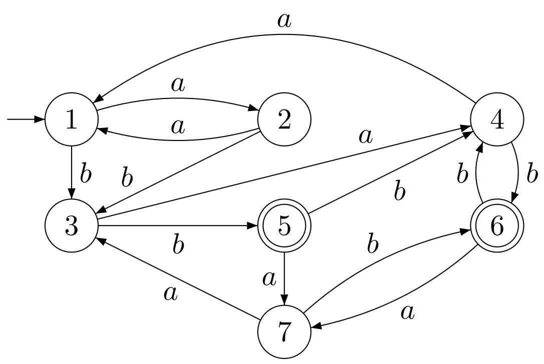

Figure 1: DFA example.

The second iteration of the loop considers the partition π_2 and processes each block of it. Thus, note that:

- $\delta_A^{-1}(\{1,2,7\},a) = \{1,2,4,5,6\}$, *which cannot be covered using the blocks in* π_3.
- $\delta_A^{-1}(\{1,2,7\},b) = \emptyset$ *and it is discarded.*
- $\delta_A^{-1}(\{3,4\},a) = \{3,7\}$ *and it is not possible to cover using the blocks in* π_3.
- $\delta_A^{-1}(\{3,4\},b) = \{1,2,5,6\}$, *which can be covered using the blocks* $\{1,2\}$ *and* $\{5,6\}$ *in* π_3. *Therefore, the transitions* $\delta_B(\{3,4\},b) = \{\{1,2\}, \{5,6\}\}$ *are added.*
- $\delta_A^{-1}(\{5,6\},a) = \emptyset$.
- $\delta_A^{-1}(\{5,6\},b) = \{3,4,7\}$, *which is covered using the blocks* $\{3,4\}$ *and* $\{7\}$ *in* π_3. *Therefore, the algorithm adds the transitions* $\delta_B(\{5,6\},b) = \{\{3,4\},\{7\}\}$.

Figure 2 depicts the automaton once the loop at line 6 ends.

Now, the loop at line 16 traverses the undefined transitions of the automaton, taking into account the blocks in the last partition to cover the remaining transitions. We stress here that the automaton may have some non-accessible states (for instance, state $\{1,2,7\}$ *in this example). Once the automaton is complete, the algorithm outputs the accessible version of the automaton. Figure 3 shows the output for this example.*

Let us now recall a remark in [7].

Remark 2 (Remark 7 in [7]) *Let* M_1 *and* M_2 *denote unions of atoms. Let us stress that atoms are disjoint sets. Then* $M_1 \cap M_2$ *and* $M_1 \cap M_2^c$ *(where* M_2^c *denotes the complementary language of* M_2*) are unions of atoms.*

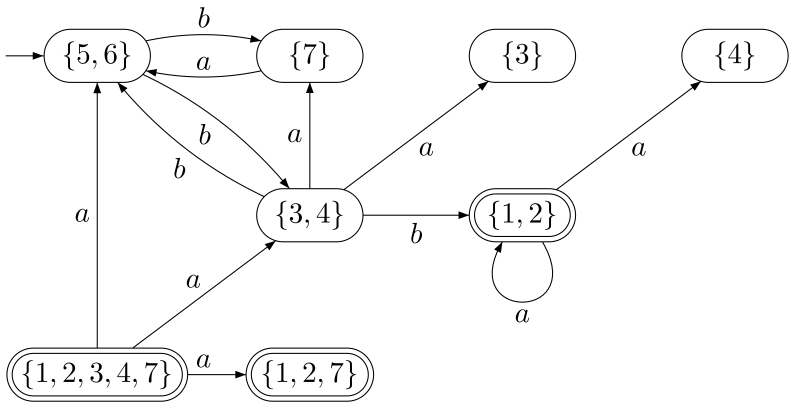

Figure 2: The automaton obtained once the sequence of partitions has been considered.

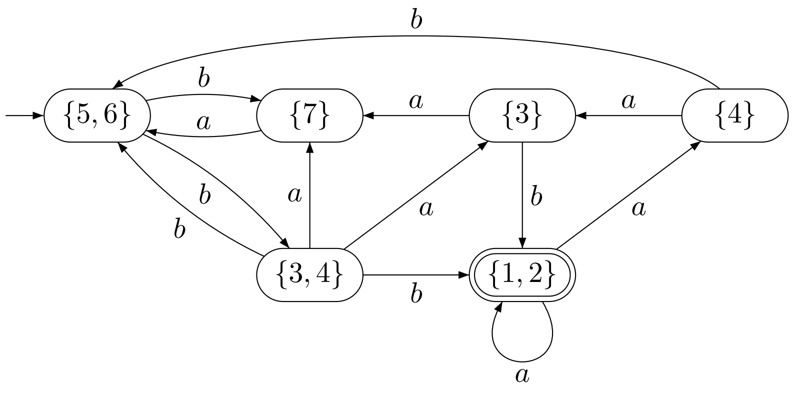

Figure 3: The final automaton output by the algorithm.

Note that, if M_1 and M_2 denote unions of atoms, then, trivially, the union of M_1 and M_2 is also a union of atoms, and the following result follows:

Remark 3 *Let $\mathscr{M} = \{M_1, M_2, \ldots, M_n\}$ be a set where each M_i is a union of atoms, then, the elements of the boolean closure of $\mathscr{M}$ are also unions of atoms.*

In Proposition 5, both Remark 3 and Proposition 4 are used to prove that the output of Algorithm 3.1 is an atomic automaton that accepts the reverse language of the input automaton.

Proposition 4 *Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let $\mathscr{P} = \langle \pi_1, \pi_2, \ldots,$ $\pi_n \rangle$ be a convergent sequence of partitions for A. Also let $Q' = \{Q_1, Q_2, \ldots,$*

$Q_r\}$ be the set of states of the automaton $D(R(A))$. It holds that every block P in $\mathscr{P}$ belongs to the boolean closure of Q'.

Proof. *We first recall that, in [8], Champarnaud et al. base their minimization algorithm on the fact that π_n can be obtained by considering the states in Q' as splitters. More formally:*

$$\pi_n = \bigwedge_{\forall Q_i \in Q'} (Q_i)|Q.$$

In other words, in any partition π_k in the sequence of partitions, the blocks in π_k can be obtained by performing a finite number of intersection and complementation operations over the elements in Q'. Therefore, the blocks in π_k belong to the boolean closure of Q'.

As we stated in Section 2, if $\mathscr{P}$ is convergent, then each block P in the sequence of partitions $\mathscr{P}$ is the union of some blocks of π_k, and, therefore, P also belongs to the boolean closure of Q'. □

Proposition 5 *Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let $\mathscr{P} = \langle \pi_1, \pi_2, \ldots, \pi_n \rangle$ be a convergent sequence of partitions for A such that A/π_n is the minimal DFA for $L(A)$. Algorithm 3.1 outputs an atomic automaton that accepts $L(A)^r$.*

Proof. *We first recall that, as stated in the previous section, the automaton $D(R(A))$ is atomic because it is deterministic. Therefore, any state in $D(R(A))$ is a union of atoms.*

The states of the automaton output by Algorithm 3.1 are the blocks of the partitions in the convergent sequence $\mathscr{P}$. By Proposition 4, all these blocks belong to the boolean closure of the set of states of $D(R(A))$.

Note that the blocks in any partition of the sequence of partitions can be obtained by the union of some blocks in π_n (the final partition is the finer one in the sequence), and therefore, the blocks in the final partition are able to cover any undefined transition (in the loop on line 16 of the algorithm).

By Remark 3, each state of the output automaton denotes a union of atoms, and, therefore, it can be concluded that the algorithm obtains an atomic automaton.

□

Note that, given any DFA A, Algorithm 3.1 and Proposition 5 relate the computation carried out by any split minimization algorithm with the construction of an atomic automaton that accepts $L(A)^r$. Therefore, Algorithm 3.1 and Proposition 5 also relate this computation with any double reversal minimization algorithm. Please note that the minimization is granted by the following proposition, due to Brzozowski and Tamm.

Proposition 6 (Corollary 2 in [6]) *B is an atomic automaton if and only if $D(R(B))$ is the minimal DFA for $L(B)^r$.*

4 From double reversal to split minimization algorithms

In this section, we show that for any given DFA A, and using any atomic automaton for the language $L(A)^r$, it is possible to obtain a set of splitters that allow to minimize A.

Note that the splitters obtained allow convergent sequences of partitions to be built, where each one of the sequences does not necessarily have to be related to a known split minimization algorithm. Similarly to what we did in Section 3, in order to use the most general conditions possible, we assume that the input can be any atomic automaton of L^r and we propose a procedure that obtains a sequence of partitions that minimizes any input DFA that accepts L.

Given any DFA A and an atomic automaton B that accepts $L(A)^r$, Algorithm 4.1 proposes a method that obtains a sequence of splitters that allows A to be minimized. The algorithm chooses a string x in $\overleftarrow{L}_q^A$ for each state q in A. These strings are used to label each state in the atomic automaton with a subset of Q_A. These sets are considered to be as the splitters in order to obtain the minimal DFA for $L(A)$. Example 7 illustrates the behavior of the proposed algorithm.

Algorithm 4.1 Algorithm to obtain a set of splitters from an atomic automaton

```
Input: A DFA $A = (Q_A, \Sigma, \delta_A, q_0, F_A)$
Input: An atomic automaton $B = (Q_B, \Sigma, \delta_B, I, F_B)$ that accepts $L(A)^r$
Output: A set of splitters $\mathcal{S} = P_1, P_2, \ldots, P_n$ such that $\bigwedge_{\forall P_i \in \mathcal{S}} (P_i)|Q_A$ is the
    partition corresponding to the minimal DFA for $L(A)$
 1: Method
 2: $W = \emptyset$
 3: for each $p \in Q_A$ do
 4:    Append to $W$ the pair $(p, u)$ where $u$ is the first string in canonical
       order such that $\delta_A(q_0, u) = p$
 5: end for
 6: Let $\mathcal{S}$ be an empty array indexed by $Q_B$
 7: for each $(p, u) \in W$ do
 8:    for each $q \in \delta_B^{-1}(F_B, u)$ do
 9:       Append $p$ to $\mathcal{S}[q]$
10:    end for
11: end for
12: return $\mathcal{S}$
13: End Method.
```

Q_B	A	B	C	D	E	F
S	$\{5,6\}$	$\{3\}$	$\{7\}$	$\{3,4\}$	$\{1,2\}$	$\{1,2,4\}$

Table 1: Relations found by Algorithm 4.1 between the states of the automata in Figures 1 and 4.

Example 7 *Let A be the DFA in Figure 1 and let B be the atomic automaton shown in Figure 4. Note that $L(B) = L(A)^r$.*

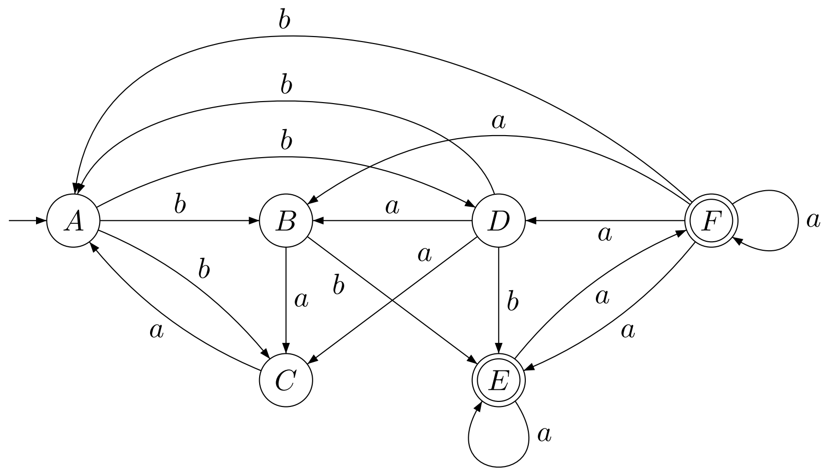

Figure 4: An atomic automaton that accepts the reverse language of the DFA shown in Figure 1.

The algorithm first stores pairs in W. Each of these pairs relates a state in A with the first string in canonical order that reach that state in A. These strings are shown below.

$$W = \{(1,\lambda),(2,a),(3,b),(4,ba),(5,bb),(6,bab),(7,bba)\}$$

The algorithm then processes the pairs in W and relates the state p to each state in $\delta_B^{-1}(F_B,u)$, where $(p,u) \in W$. For instance, note that $\delta_B^{-1}(F_B,bb) = \{A\}$; therefore, $S[A]$ will contain the state $5 \in Q_A$ (note that the pair $(5,bb) \in W$). Table 1 shows the whole set of relations.

Finally, the algorithm outputs the set S. Although it is not explicitly stated in the algorithm, let us show that the sets stored in S allow the partition induced by Nerode's equivalence relation to be obtained. Actually, note that the order in which the splitters are considered may lead to different *sequences of partitions, but always with the same result. For instance:*

$$\begin{aligned}
\pi_1 &= (\{1,2,4\})|Q_A = \{\{1,2,4\},\{3,5,6,7\}\} \\
\pi_2 &= \pi_1 \wedge (\{5,6\})|Q = \{\{1,2,4\},\{3,5,6,7\}\} \wedge \{\{1,2,3,4,7\},\{5,6\}\} \\
&= \{\{1,2,4\},\{3,7\},\{5,6\}\} \\
\pi_3 &= \pi_2 \wedge (\{7\})|Q = \{\{1,2,4\},\{3\},\{5,6\},\{7\}\} \\
\pi_4 &= \pi_3 \wedge (\{3,4\})|Q = \{\{1,2\},\{3\},\{4\},\{5,6\},\{7\}\}
\end{aligned}$$

Also note that the consideration of the remaining splitters does not modify the partition, which is the same that is induced by Nerode's equivalence relation.

The proof of correctness of the algorithm takes into account that the left language of every state of $R(B)$ is a union of classes of Nerode's equivalence for the language $L(B)^r = L(A)$. Proposition 8 and Corollary 9 are helpful in proving the correctness of the algorithm in Proposition 10.

Proposition 8 *Let $A = (Q, \Sigma, \delta, I, F)$ be an automaton. It holds that $D(A)$ is the minimal DFA for $L(A)$ if and only if, for every state q of A, the language $\overleftarrow{L}_q^A$ is a union of classes of Nerode's equivalence for the language $L(A)$.*

Proof. *On the one hand, let us suppose the contrary; thus, for some class $[x]$ of $L(A)$ there would be two strings u and v in the class $[x]$ and a state p in A such that $p \in \delta(I,u)$ and $p \notin \delta(I,v)$. Therefore, in $D(A)$, two strings in the same class would go to different states, which is a contradiction because $D(A)$ is minimal.*

On the other hand, note that if the left languages of the states of A are unions of classes of Nerode's equivalence, then, for any given class $[x]$, every string in the class $[x]$ reaches the same set of states, and, therefore, $D(A)$ is minimal. □

Corollary 9 *Let $B = (Q, \Sigma, \delta, I, F)$ be an atomic automaton. Then $R(B)$ is such that the left language of each state is a union of classes of Nerode's equivalence for the language $L(B)^r$.*

Proof. *Note that, taking into account that $D(R(B))$ is the minimal DFA for $L(B)^r$ (Proposition 6), it is a direct consequence of Proposition 8.* □

Proposition 10 *Let $A = (Q_A, \Sigma, \delta_A, q_0, F_A)$ be a DFA and let $B = (Q_B, \Sigma, \delta_B, I, F_B)$ be an atomic automaton such that $L(B) = L(A)^r$. Algorithm 4.1 outputs a set of splitters that allow the minimal DFA for $L(A)$ to be obtained.*

Proof. *Let us note that the left language of every state in A represents a fraction of a class of Nerode's equivalence. Let us also note that B is atomic and, by Corollary 9, the left language of each state in $R(B)$ is a union of classes of Nerode's equivalence relation on $L(A)$.*

We first note that, if p and q in Q_A are equivalent and x_p and x_q are representative strings of p and q, then x_p and x_q are in the same class of Nerode's equivalence for $L(A)$, and therefore $\delta_{R(B)}(I_{R(B)}, x_p)$ is equal to $\delta_{R(B)}(I_{R(B)}, x_q)$ because the left languages of the states in $R(B)$ are union of Nerode's equivalence classes for $L(A)$.

Second, we note that, if $\delta_{R(B)}(I_{R(B)}, x_p) = \delta_{R(B)}(I_{R(B)}, x_q)$ then the quotients $x_p^{-1}L(R(B))$ and $x_q^{-1}L(R(B))$ are equal, and therefore x_p and x_q are in the same class of the Nerode's equivalence relation for $L(R(B)) = L(A)$, and therefore, due to A is a DFA, x_p and x_q are representative strings of equivalent states.

In other terms, the representatives of two states p and q in Q_A reach the same states in $R(B)$ if and only if $\overleftarrow{L}_p^A$ and $\overleftarrow{L}_q^A$ are fragments of the same class of Nerode's equivalence, and this is so if and only if p and q are equivalent.

Algorithm 4.1 relates a set of states $P \subseteq Q_A$ to each state $p \in Q_B$ (we use the notation of the algorithm and denote the whole set of subsets of Q_A related to states in B as $\mathcal{S}$). To do so, the loop at line 7 takes into account the first string in canonical order that reaches any state of A in order to assign to each state in Q_B a set of states of A that represents the mentioned above unions of classes of Nerode's equivalence relation.

Therefore, if p and q in Q_A are equivalent, then, for any $P \in \mathcal{S}$, it holds that $p \in P$ if and only if $q \in P$. If p and q are not equivalent, then there is a set $P \in \mathcal{S}$ such that $p \in \mathcal{S}$ and $q \notin \mathcal{S}$. This implies that $\mathcal{S}$ is a valid set of splitters.

□

In this section, for any given DFA A, Algorithm 4.1 and Proposition 10 relate the computation carried out by a double reversal minimization algorithm (regardless of how the atomic automaton is obtained) to the minimization of A using a split minimization algorithm. This result completes the result in Section 3 and shows the relationship between the two minimization approaches.

5 Conclusions

The minimization of DFA are usually tackled using two different approaches that are traditionally considered to be unrelated. The first approach includes methods that iteratively refine an initial partition that distinguishes final and non-final states. These methods have recently been described in a unified way by Berstel et al. [4]. The second approach is represented by the method first proposed by Brzozowski [5] and recently extended by Brzozowski and Tamm [6].

The paper by Champarnaud et al. [8] can be viewed as a first attempt to relate Brzozowski's algorithm to split minimization methods. In our work, we clarify the relationship between the two approaches. To do this, we consider both any convergent sequence of partitions (and therefore any potential split minimization algorithm) and any method that obtains an atomic automaton (and therefore any generic double reversal algorithm according to [6]). We provide methods to transform any set of splitters into an atomic automaton and vice versa, thus showing the relationship between the two approaches.

References

[1] E. F. Moore. Gedanken experiments on sequential machines. In C. E. Shannon and J. Mc-Carthy, editors, *Automata Studies.* Princeton Universty Press, 1956.

[2] J. E. Hopcroft. An $n \cdot \log n$ algorithm for minimizing states in a finite automaton. Technical report, Stanford University, Stanford, CA, USA, 1971.

[3] D. Wood. *Theory of Computation.* John Wiley & sons, 1987.

[4] J. Berstel, L. Boasson, O. Carton, and I. Fagnot. *Automata: from Mathematics to Applications*, chapter Minimization of automata. European Mathematical Society. (arXiv:1010.5318v3). To appear.

[5] J.A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. *Mathematical Theory of Automata*, pages 529–561, 1962. MRI Symposia Series, Polytecnic Press, Polytecnic Institute of Brooklyn.

[6] J. A. Brzozowski and H. Tamm. Theory of átomata. *Theoretical Computer Science*, 539:13–27, 2014.

[7] M. Vázquez de Parga, P. García, and D. López. A polynomial double reversal minimization algorithm for deterministic finite automata. *Theoretical Computer Science*, 487:17–22, 2013.

[8] J-M. Champarnaud, A. Khorsi, and T. Paranthoën. Split and join for minimizing : Brzozowski´s algorithm. Technical report, Czech Technical University of Prague, 2002. Proceedings of the Prague Stringology Conference 2002 (PSC'02).