A polynomial double reversal minimizational algorithm for deterministic finite automata

Manuel Vázquez de Parga, Pedro García, Damián López *
Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Spain

A R T I C L E   I N F O
Article history:
Received 3 May 2012
Accepted 5 March 2013
Communicated by J. Karhumaki
Keywords:
DFA minimization
Atomic automata
Polynomial double reversal algorithm

A B S T R A C T
We here propose a polynomial-time deterministic finite automaton minimization algorithm directly derived from Brzozowski's double reversal algorithm. To do so, we take into account the framework by Brzozowski and Tamm, to propose an atomization algorithm that allows us to achieve polynomial time complexity.

© 2013 Elsevier B.V. All rights reserved.

1. Introduction

The problem of automata minimization is a classic issue in Computer Science, with applications in many fields such as text processing, image analysis, and linguistics. Nowadays, this problem still arouses interest due to its importance in the implementation of efficient solutions.

Among the different approaches to automata minimization, the algorithm proposed by Brzozowski [1] is of special interest. This algorithm is based on two well-known automata constructions. The input to the algorithm is not restricted to a deterministic finite automaton (DFA), and its implementation is very straightforward. In essence, the algorithm alternates two reverse and determinization operations, and it is usually referred to as the double reversal minimizational algorithm.

This very concise and elegant algorithm has been usually set aside from the rest and considered unrelated to other approaches [2,3]. Despite its worst-case exponential time complexity, the algorithm has recently aroused interest, and some papers study its relationship with other classical algorithms. In this way, the papers by Champarnaud et al. [2] and García et al. [4] propose methods that substitute reverse and determinization operations by a sequence of split operations.

It is also important to cite here the paper by Brzozowski and Tamm [5], where the authors introduce the átomaton (in a similar way Sengoku [6] present the standard form automaton), and atomic automata, where the right language of the states is a union of some atoms. As the authors prove, the class of atomic automata is in fact a generalization of the átomaton, the DFA, and residual automata [7], and includes as well the universal automata [8,9]. Brzozowski and Tamm also prove in their paper that the first determinization of the double reversal algorithm can be substituted by any algorithm that returns an atomic automaton. As a consequence of this, the original double reversal algorithm [1] becomes an implementation within this framework.

In this paper we show that, whenever the input automaton \( A \) is deterministic, the atomization operation can be implemented with polynomial time complexity, which leads to a polynomial double reversal minimizational algorithm. The paper is structured as follows. Section 2 summarizes the notation and essential definitions needed in the paper. Then,
Section 3 recalls the main results related to the Brzozowski algorithm. We propose our polynomial atomization algorithm in Section 4. The correctness and polynomial time behaviour of the algorithm are also proved in that section.

2. Notation and definitions

Let $\Sigma$ be a finite alphabet, and let $\Sigma^*$ be the free monoid generated by $\Sigma$ with concatenation as the internal operation and the empty string $\lambda$ as neutral element. For any given $x \in \Sigma^*$, we will denote by $x^r$ the reverse of $x$. Let us denote the size of a set $Q$ by $|Q|$. Let us also denote by $2^Q$ the power set of $Q$.

A (non-deterministic) finite automaton (NFA) is a 5-tuple $A = (Q, \Sigma, \delta, I, F)$, where $Q$ is a finite set of states, $\Sigma$ is an alphabet, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Sigma \rightarrow 2^Q$ 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 $\Sigma^*$ and as well as to $2^Q$. Given an NFA $A$, we say that it is accessible if, for each $q \in Q$, there exists a string $x$ such that $q \in \delta(p, x)$ for some $p \in I$. The right language of a state $q$ of an NFA $A$ is defined as $I_q^r = \{x \in \Sigma^* : \delta(q, x) \cap F \neq \emptyset\}$. The language accepted by the NFA, which we will denote $L(A)$, is the union of the right languages of the initial states.

An automaton $A = (Q, \Sigma, \delta, \{q_0\}, F)$ is called deterministic (i.e., it is a DFA) if, for every state $q$ and every symbol $a$, the number of transitions is at most one, and $q_0$ is the only initial state. In the following, we will consider only accessible DFAs.

Given a language $L$ and an NFA $A = (Q, \Sigma, \delta, \{q_0\}, F)$ such that $L = L(A)$, the reverse automaton for $A$ is defined as the automaton $R(A) = (Q, \Sigma, \delta^r, F, I)$, where $q \in \delta^r(p, a)$ if and only if $p \in \delta(q, a)$. Given any language $L$, we will denote the reverse and the complementary languages by $L^r$ and $L^c$, respectively. For any NFA $A = (Q, \Sigma, \delta, \{q_0\}, F)$, it is known that the automaton $A' = (2^Q, \Sigma, \delta^r, \{I\}, F')$, where $F' = \{p \in 2^Q : p \cap F \neq \emptyset\}$ and $\delta^r(p, a) = \bigcup_{p \in \delta(q, a)} \delta(p, a)$, is a DFA equivalent to $A$. Let us denote the accessible version of $A'$ by $A^a$.

Given a regular language $L$ over $\Sigma$, the (left) quotient of $L$ by a word $u$ is defined as the language $u^{-1}L = \{v \in \Sigma^* : uv \in L\}$. It is well known that the set of quotients of a regular language is finite; let us denote this set by $\{L_1, L_2, \ldots, L_n\}$. This set permits us to define an atom $A$ of $L$ as any of the nonempty languages in the form of $L_1 \cap L_2 \cap \cdots \cap L_n$, where $L_i$ (which we will refer to as a literal) is either $L_i$ or $L_i^r$, and such that not all the quotients are complemented.

Brzozowski and Tamm define in [5] the atomaton of a given language $L$ as the automaton such that the atoms of $L$ determine the set of states. For any DFA $A$, the atomaton 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.

A partition of a set $Q$ is any set $\{P_1, P_2, \ldots, P_k\}$ of pairwise disjoint nonempty subsets of $Q$ such that $Q = \bigcup_{1 \leq i \leq k} P_i$. When necessary, we will refer to those subsets as blocks. In an analogous way to that of Berstel et al. in [10], we will denote by $S/P$ the partition of $P$ composed of the nonempty sets among the following: $S \cap P$ and $P - S$.

Let $P$ and $P'$ be two partitions of a given set $Q$; we will also denote by $P \land P'$ the coarsest partition which refines both $P$ and $P'$. Note that the blocks of $P \land P'$ are the nonempty sets $P \cap P'$ with $P \in P$ and $P' \in P'$.

3. Brzozowski’s algorithm

The algorithm proposed by Brzozowski [1] computes the minimal DFA equivalent to any non-deterministic automaton $A$. The process is based on the following result.

**Proposition 1** (Brzozowski). Given a DFA $A = (Q, \Sigma, \delta, \{q_0\}, F)$, then $D(R(A))$ is the minimal DFA which accepts the language $L(A)^r$.

This result implies that the minimal DFA of any automaton is equivalent up to isomorphism to the automaton $D(R(D(R(A))))$.

Recently, Brzozowski and Tamm [5] have proposed an extension based on the following theorem.

**Theorem 2** (Brzozowski and Tamm [5]). Given any accessible automaton $A$, then $D(A)$ is minimal if and only if $R(A)$ is atomic.

This result allows to the authors to propose an extension of the double reversal minimization algorithm. This extension consists of the computation of the automaton $D(R(At(R(A))))$, where the function $At(A)$ is any function that outputs an atomic automaton equivalent to its input. Note that the original double reversal algorithm can be seen as an instance in this framework, because any determinstic automaton is atomic.

In their work, Brzozowski and Tamm do not give an algorithm to compute the atomization of an automaton. In the next section, we propose a polynomial-time algorithm that, for any DFA $A$, and given $B = R(A)$, outputs an atomic automaton equivalent to $B$. This algorithm leads to a polynomial double reversal minimization algorithm for DFAs.

4. A polynomial minimization algorithm according to the Brzozowski–Tamm framework

**Definition 3** describes a construction of a replica of any given NFA $A$. **Proposition 4** proves that any replica of $A$ accepts the same language, which will be helpful to prove the correctness of the atomization algorithm we propose later. **Example 5** illustrates how this construction works.
**Definition 3.** Let \( A = (Q, \Sigma, \delta, I, F) \) be an automaton. A replica of \( A \) is defined as any accessible automaton \( A' = (Q', \Sigma, \delta', I', F') \) such that the following hold.

- \( Q' \subseteq 2^Q \).
- \( I = \bigcup_{P \in I'} P \).
- \( F' = \{ P \in Q' : P \cap F \neq \emptyset \} \).
- For any \( P \in Q' \) and \( a \in \Sigma \), \( \delta'(P, a) = \{ P_1, \ldots, P_k \} \), where \( P_i \in Q' \), implies that
  \[ \delta(P, a) = \bigcup_{i=1}^{k} P_i. \]

**Proposition 4.** Given any automaton \( A = (Q, \Sigma, \delta, I, F) \), any replica \( A' = (Q', \Sigma, \delta', I', F') \) of \( A \) accepts \( L(A) \).

**Proof.** Let \( P \subseteq Q \) and \( P_1, \ldots, P_k \in Q' \) such that
\[ P = \bigcup_{i=1}^{k} P_i. \]

Note that, by definition of a replica, the following holds:
\[ \delta(P, a) = \bigcup_{i=1}^{k} \delta'(P_i, a). \]

Note also that, for any \( x \in \Sigma^* \), it is possible to prove by induction on the length of \( x \), that the following equality is fulfilled:
\[ \delta(P, x) = \bigcup_{i=1}^{k} \delta'(P_i, x), \]
and, in particular,
\[ \delta(I, x) = \bigcup_{P \in I'} \delta'(P, x), \]
and, therefore, we can conclude that \( L(A) = L(A') \).

**Example 5.** Let us consider the automaton in Fig. 1. The following table summarizes the construction of a replica according to **Definition 3**.

<table>
<thead>
<tr>
<th></th>
<th>( a )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rightarrow {1} )</td>
<td>{1, 2}</td>
<td>{1}</td>
</tr>
<tr>
<td>( \rightarrow {2} )</td>
<td>{3, 4}</td>
<td>{3, 4}</td>
</tr>
<tr>
<td>( {1, 2} )</td>
<td>{1, 2}, {3, 4}</td>
<td>{1}, {3, 4}</td>
</tr>
<tr>
<td>( {3, 4} )</td>
<td>{5}</td>
<td>{5}</td>
</tr>
<tr>
<td>( \leftarrow {5} )</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>

Note that one more replica of the automaton is the automaton represented in the following table:

<table>
<thead>
<tr>
<th></th>
<th>( a )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rightarrow {1, 2} )</td>
<td>{1, 2}, {3, 4}</td>
<td>{1}, {3, 4}</td>
</tr>
<tr>
<td>( {3, 4} )</td>
<td>{5}</td>
<td>{5}</td>
</tr>
<tr>
<td>( {1} )</td>
<td>{1, 2}</td>
<td>{1}</td>
</tr>
<tr>
<td>( \leftarrow {5} )</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>

Finally, we note that both \( A \) and \( D(A) \) are replicas of \( A \).

We now propose **Algorithm 4.1**, which outputs a replica of the input automaton. **Example 6** shows an example of a run.
Algorithm 4.1 Algorithm to obtain a replica of the input automaton

Require: An automaton $A = (Q, \Sigma, \delta, I, F)$.
Ensure: A replica of $A$

1: Method
2: $I = \{I\}$
3: $Q = I$
4: $\delta' = \emptyset$
5: $P = \{I\}$ if $I \cap F \neq \emptyset$
6: $\mathcal{S} = I$
7: while $\mathcal{S} \neq \{\}$ do
8: Choose and delete $P$ from $\mathcal{S}$
9: for $a \in \Sigma$ do
10: $P = \bigcup_{S \in P} \delta(P, a)$
11: $\mathcal{S} = \text{Append}(\mathcal{S}, P - Q)$
12: $Q = Q \cup P$
13: Add to $\delta'$ the transitions $(P, a, S')$, where $S' \in P$
14: $F = F \cup \{S \in P : S \cap F \neq \emptyset\}$
15: end for
16: end while
17: Return $(A' = (Q, \Sigma, \delta', I, F))$
18: End Method.

Example 6. Let us consider the automaton shown in Fig. 1. The algorithm initially considers the initial state $I = \{1, 2\}$ (which is non-final). Please note that the algorithm is not deterministic and that the result depends on the implementation of line 8. In this example, we will follow a first-in–first-out scheme.

In the main loop, $P$ is set to $\{1, 2\}$, and thus $\delta(P, a) = \{1, 2, 3, 4\}$ is partitioned into $\{\{1, 2\}, \{3, 4\}\}$. The state $\{3, 4\}$ is added to $Q$ and included in $\mathcal{S}$. The function $\delta'$ is also updated with the transitions $\{(1, 2), a, \{1, 2\}\}$ and $\{(1, 2), a, \{3, 4\}\}$.

The same process is carried out taking into account $\delta(P, b) = \{1, 3, 4\}$, which is partitioned into $\{\{1\}, \{3, 4\}\}$, and that leads to the update of $Q$ and $\mathcal{S}$ with the addition of $\{1\}$. The function $\delta'$ is also updated taking into account the new transitions $\{(1, 2), a, \{1\}\}$ and $\{(1, 2), b, \{1\}\}$.

The second iteration considers $P = \{3, 4\}$. Note that $\delta(P, a) = \delta(P, b) = \{5\}$. This set is not split; thus, the function $\delta'$ is also updated with the transitions $\{(3, 4), a, \{5\}\}$ and $\{(3, 4), b, \{5\}\}$. Both the set of states $Q$ and $\mathcal{S}$ are modified as well with the addition of $\{5\}$.

The state to be considered in the next iteration is $P = \{1\}$, and the algorithm obtains $\delta(P, b) = \{1\}$, which is not partitioned. The transition function is therefore updated with $\{(1), b, \{1\}\}$. Neither the set of states $Q$ nor $\mathcal{S}$ is modified.

Note that $\delta(P, a) = \{1, 2\}$, which is partitioned in line 10 to obtain $\{(1), \{2\}\}$. The transition function is therefore updated with $\{(1), a, \{1\}\}$ and $\{(1), a, \{2\}\}$. This also leads to the update of $Q$ and $\mathcal{S}$ with the addition of $\{2\}$. No more states are added.

The process continues and, when the algorithm ends, it returns the automaton shown in Fig. 2.

A referee noted that the algorithm can be modified in order not to carry out the partition of those states $\delta(P, a)$ which are already in the state set $Q$ (line 10). Thus, the resulting automaton could have fewer states (for instance, in Example 6, state $\{2\}$ would not appear).

In order for Algorithm 4.1 to be useful in a polynomial DFA minimization process, we prove in Proposition 10 that, for any NFA $A$ such that $R(A)$ is deterministic, the algorithm outputs an atomic replica of the input. In order to enlighten the proof of this, a remark and a previous lemma are presented. Proposition 11 proves that the algorithm runs in polynomial time.

Remark 7. Let $M_1$ and $M_2$ denote union of atoms. 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.
Proof. Note that, given a language, the set of its unions of atoms is a Boolean algebra with operations union, intersection, and complementation over the set of quotients of the language. Therefore, both $M_1 \cap M_2$ and $M_1 \cap M_2^c$ can be expressed, using set theory properties, as a union of intersections of sets that represent quotients of the language (or their complements).

Remark 8. Let $A = (Q, \Sigma, \delta, I, \{q_0\})$ be a finite automaton such that $R(A)$ is a DFA. Given $P, P' \subseteq Q$ such that $L^A_P$ and $L^A_{P'}$ are union of atoms of $L(A)$, it is fulfilled that the unions of atoms $L^A_P \cap L^A_{P'}$ and $L^A_P - L^A_{P'}$ are represented unambiguously by the sets $P \cap P'$ and $P - P'$, respectively.

Proof. Note that, for every $p, q \in Q$, if $p \neq q$, then $L^A_p \cap L^A_q = \emptyset$, because $R(A)$ is deterministic. Therefore, for any $P, P' \subseteq Q$, it is fulfilled that $L^A_P \cap L^A_{P'} = L^A_{P \cap P'}$. □

Remark 9. Let $A = (Q, \Sigma, \delta, I, \{q_0\})$ be a finite automaton such that $R(A)$ is a DFA. Given $P \subseteq Q$ such that $L^A_P$ is a union of atoms of $L(A)$, then, for any $a \in \Sigma$, the set $\delta(P, a)$ is also a union of atoms of $L(A)$.

Proof. From the fact that $P$ is such that $L^A_P$ is a union of atoms, it follows that the quotient $a^{-1}L^A_P$ is also a union of atoms, because $a^{-1}(L \cup L') = a^{-1}L \cup a^{-1}L'$, $a^{-1}(L \cap L') = a^{-1}L \cap a^{-1}L'$, and $a^{-1}L^c = \Sigma^* - a^{-1}L = (a^{-1}L)^c$. Therefore, by Remark 8, the set of atoms of the language $a^{-1}L^A_P$ is represented by the set $\delta(P, a)$. □

Proposition 10. Let $A = (Q, \Sigma, \delta, I, \{q_0\})$ be a finite automaton such that $R(A)$ is an accessible DFA. Algorithm 4.1 outputs an atomic replica of $A$.

Proof. First, it is easy to see that the algorithm outputs an automaton that holds the conditions in Definition 3, and therefore, by Proposition 4, accepts the same language. We prove now that it is atomic.

Let us here recall that the automaton $D(A)$ is the minimal DFA for the language $L(A)$; thus, the states in $D(A)$ are the different quotients of $L(A)$. Each quotient is a union of atoms [5].

When Algorithm 4.1 is run, the set $Q$ initially contains only the state $I$, which corresponds to the quotient $(\lambda^{-1}L(A))$ and therefore to a union of atoms.

Let us suppose that at a given iteration $\mathcal{E} \neq \emptyset$, and that $Q$ contains only states that represent unions of atoms. Let $P$ be a state extracted from $\mathcal{E}$ at the beginning of an iteration. Note that the partition $S|\delta(P, a)$ (where $\delta(P, a)$, by Lemma 9, is itself a union of atoms) returns either $\delta(P, a)$, or the sets $S \cap \delta(P, a)$ and $\delta(P, a) - S$, both of which, by Remark 7, represent unions of atoms.

Thus, $P = \bigcup_{S \in Q} S|\delta(P, a)$ is a partition of $\delta(P, a)$ where all its elements are unions of atoms. □

Proposition 11. Algorithm 4.1 runs with polynomial-time complexity.

Proof. To prove the proposition we will prove that the number of elements appended to the list $\mathcal{E}$ is bounded by a polynomial. First, let us note that, for any $P \in Q$, and according the algorithm (line 10), every $P'$ to be appended to the list fulfills that it is either included in $P$ or disjoint from $P$.

Given any set $Q$, we now prove, by induction on the size of the set, that the number of sets fulfilling the previous conditions is at most $2|Q| - 1$. First, it is trivial that, when $|Q| = 1$, the proposition holds.

Let us suppose that the conditions hold when the set has fewer than $k$ elements. Let us now consider the set $Q$ with $k$ elements. We note that the worst case occurs when twofold partitions are successively carried out. Let us consider, without loss of generality, this case. Thus, we consider that $Q$ is partitioned into two sets $P$ and $P'$ with size $n$ and $k - n$, respectively. By the induction hypothesis, the numbers of sets of $P$ and $P'$ that hold the conditions are $2n - 1$ and $2(k - n) - 1$, respectively. Note that the number of sets of $Q$, in this worst case, is $1$ plus the sum of both values, that is, $2k - 1$. □

Let us note that, whenever the input automaton is deterministic, it is straightforward to use our algorithm in the framework proposed by Brzozowski and Tamm. Thus, the algorithm carries out the minimization of any DFA $A$ by computing the automaton $D(R(\lambda(R(A))))$, where the function $\lambda$ is implemented by Algorithm 4.1. We finally note that the complexity of the whole process is bounded by a polynomial because both reverse operations can be carried out with linear complexity; Proposition 11 proves the polynomial bound of the atomization; and the determination is bounded by the size of the input automaton. Example 12 depicts a minimization process.

Example 12. Let $A$ be the DFA in Fig. 3. Note that $R(A)$ is the automaton shown in Example 6 (depicted in Fig. 1); thus, by Proposition 10, the output of the algorithm (shown in Fig. 2) is an atomic replica of $R(A)$. According to a double reversal approach, subsequent reverse and determinization steps result in the automaton shown in Fig. 4, which is the minimal DFA that accepts $L(A)$.
5. Conclusions

The Brzozowski algorithm [1] outputs the minimal DFA equivalent to any input NFA. The restriction of the input in order to consider only deterministic automata permits us to propose a polynomial minimization algorithm within the framework proposed by Brzozowski and Tamm in [5]. Thus, according this framework, the algorithm we propose substitutes the first determinization of Brzozowski’s algorithm by the output of the Algorithm 4.1.

It is known that residual automata, the universal automaton, and the átomaton of a language are atomic [5]. Therefore, it should be possible to implement the atomization function in the Brzozowski–Tamm framework in some other ways to consider an NFA as input. Nevertheless, according to classic complexity results, it is also well known that any double reversal algorithm with input an arbitrary NFA and output the minimal equivalent DFA could not run in polynomial time.

Acknowledgment

We thank the anonymous referee for comments on our results and also on the atomization algorithm.

References