The Full Quotient and Its Closure Property for Regular Languages

Teodor KNAPIK, Étienne PAYET

Université de la Réunion, BP 7151, 97715 Saint Denis Messag. Cedex 9, France
e-mail : {knapik, epayet}@univ-reunion.fr

Abstract

In this article, we introduce a new operation on languages. Due to its similarity to usual quotient, we name it *full quotient*. Its interest lies in the two following facts. First of all, the full quotient seems to be an elementary operation, in the sense that we think it can not be expressed as a finite combination of familiar operations. Secondly, the full quotient of two regular languages is also regular.

The main subject of this article is the demonstration of this latter fact. For this purpose, we provide an algorithm, the input of which is two regular languages given as finite automata, and the output of which is an automaton recognizing the full quotient of the initial languages. Its time complexity is proportional to the product of the number of states of the initial automata.

Key words: Formal languages

Introduction

This article is devoted to the presentation of a new operation on languages that is called the *full quotient*. This operation is derived from the *quotient* defined e.g. in [2], also called *residual* in [3] or *derivative* in [1]. Roughly speaking, the full quotient is defined as a subset of the usual quotient, each element of which satisfies a kind of minimality condition. Similarly to the quotient, full quotient has two forms: left and right.

Our interest in this operation is motivated by its relationship to the topic we are currently investigating, namely, the string rewriting of regular sets of words using rightmost and leftmost reduction strategy. Nevertheless, the full quotient seems general enough to be introduced separately in the present paper.

The closure property of the family of regular languages under full quotient is proved in this article. The reader is provided with an algorithm constructing a finite automaton recognizing the left full quotient of two regular languages. The time complexity of this algorithm is also given.

1 Basic Definitions

The reader is expected to have a smattering of finite automata and formal languages. Nevertheless, few basic definitions from these topics are recalled in the sequel.

From now on, suppose that every word or language considered is built over a finite alphabet A. Given a word w, the set of its non-empty prefixes (resp. suffixes), written $\mathsf{pref}_+(w)$ (resp. $\mathsf{suff}_+(w)$), is defined as follows:

$$\begin{aligned} \mathsf{pref}_+(w) &= \{u \in A^+ \mid \exists\, v \in A^* \;.\; uv = w\} \\ (\text{resp. } \mathsf{suff}_+(w) &= \{v \in A^+ \mid \exists\, u \in A^* \;.\; uv = w\})\;. \end{aligned}$$

The set $\mathsf{pref}_+(w) \cup \{\varepsilon\}$ of all prefixes of w, including the empty word ε, is written $\mathsf{pref}(w)$.

Let M and N be two languages. The *left quotient* (resp. *right quotient*) of N by M, written $M\backslash N$ (resp. N/M), is defined as follows:

$$\begin{aligned} M\backslash N &= \{u \in A^* \mid \exists\, w \in M \;.\; wu \in N\} \\ (\text{resp. } N/M &= \{u \in A^* \mid \exists\, w \in M \;.\; uw \in N\})\;. \end{aligned}$$

A *finite deterministic automaton* on A is given by a finite set Q of *states*, an element q_0 of Q called the *initial* state, a subset F of Q called the set of *final* states and a *transition* function δ from $Q \times A$ to Q. A finite deterministic automaton is said to be *complete* if its transition function is total. A finite deterministic automaton is noted as a quadruple (Q, δ, q_0, F).

A *path* in the finite deterministic automaton $\mathcal{A} = (Q_\mathcal{A}, \delta_\mathcal{A}, q_{0_\mathcal{A}}, F_\mathcal{A})$ is a word $t_1 \ldots t_n$ in $(Q_\mathcal{A} \times A \times Q_\mathcal{A})^*$ such that for all $i \in \{1, \ldots, n\}$, $t_i = (q_i, a_i, q_{i+1})$ and $q_{i+1} = \delta_\mathcal{A}(q_i, a_i)$. The word $a_1 a_2 \ldots a_n$ is the *label* of the path, the state q_1 its *origin* and the state q_{n+1} its *end*. A word is said to be *recognized* by $\mathcal{A}$ if it is the label of a path in $\mathcal{A}$, the origin of which is $q_{0_\mathcal{A}}$ and the end of which belongs to $F_\mathcal{A}$. The *language* $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of all the words recognized by $\mathcal{A}$. A state of $\mathcal{A}$ is said to be *reachable* if it is the end of a path from $q_{0_\mathcal{A}}$ in $\mathcal{A}$. The set of all the reachable states of $\mathcal{A}$ is written $R_\mathcal{A}$.

2 Full Quotient and Its Closure Property for Regular Languages

Assume that one is interested in the following problem: given a language M and a word x, find a word u such that $wu = x$ and w is the maximal element of $M \cap \mathsf{pref}(x)$. Then, u may be seen as a result of a partial operation "$\backslash\backslash$" involving M and x, that is, $u = M\backslash\backslash x$. Such an operation may be naturally extended to languages by setting $M\backslash\backslash N = \bigcup_{x\in N} M\backslash\backslash x$ for $N \subseteq A^*$. The operation "$\backslash\backslash$" is precisely defined as follows.

Definition The *left full quotient* (resp. *right full quotient*) of $N \subseteq A^*$ by $M \subseteq A^*$, written $M\backslash\backslash N$ (resp. $N/\!/M$), is defined as follows:

$$M\backslash\backslash N = \{u \in A^* \mid \exists\, w \in M \;.\; wu \in N \wedge \forall\, v \in \mathsf{pref}_+(u) \;.\; wv \notin M\}$$
$$(\text{resp. } N/\!/M = \{u \in A^* \mid \exists\, w \in M \;.\; uw \in N \wedge \forall\, v \in \mathsf{suff}_+(u) \;.\; vw \notin M\})\;.$$

The following example illustrates the concepts introduced above.

Example *Let $A = \{a, b\}$ be an alphabet and $M = a(ba)^*$ and $N = a(a+b)^*$ be two languages over A.*

First, notice that any word u in A^* and any word w in M are such that $wu \in N$. However, u belongs to $M\backslash\backslash N$ only if it satisfies the condition on prefixes, that is, only if it is not in $(ba)^+(a+b)^*$. Therefore, u must be an element of $\varepsilon + b + (a+bb)(a+b)^*$. Since every element of the latter set is in $M\backslash\backslash N$, we have $M\backslash\backslash N = \varepsilon + b + (a+bb)(a+b)^*$.

Similar reasoning yields $N/\!/M = \varepsilon + a(a+b)^*(a+bb)$.

One common mistake about the left full quotient is to believe that $M\backslash\backslash N = \mathsf{Min}_{\mathsf{suff}}(M\backslash N)$ or $M\backslash\backslash N = \mathsf{Max}_{\mathsf{pref}}(M)\backslash N$ where $\mathsf{Min}_{\mathsf{suff}}$ (resp. $\mathsf{Max}_{\mathsf{pref}}$) stands for the restriction to the minimal elements with respect to the suffix ordering (resp. maximal elements with respect to the prefix ordering). It is not too difficult to show that both $\mathsf{Min}_{\mathsf{suff}}(M\backslash N)$ and $\mathsf{Max}_{\mathsf{pref}}(M)\backslash N$ are included in $M\backslash\backslash N$, but these inclusions may be proper. The latter assertion may be checked by taking $M = \{a, b, ab\}$ and $N = \{aaa, aba, abb, bba\}$. In fact, considering the subset of $M\backslash N$ of "non-left full quotients", that is

$$M\not\!\backslash\!\!\!\not\backslash N = \{u \in A^* \mid \exists\, w \in M \;.\; wu \in N \wedge \exists\, v \in \mathsf{pref}_+(u)\,.\, wv \in M\},$$

the relation between $M\backslash N$ and $M\backslash\backslash N$ can be written:

$$M\backslash N = M\backslash\backslash N \cup M\not\!\backslash\!\!\!\not\backslash N\;.$$

Notice that $M\backslash\backslash N$ and $M\mathbin{\diagup\!\!\!\!\diagdown}N$ are not always disjoint sets. As a proof, consider languages $M = \{a, b, ba\}$ and $N = \{aab, bab\}$: we have $ab \in M\backslash\backslash N \cap M\mathbin{\diagup\!\!\!\!\diagdown}N$.

Analogous comments relevant to the right full quotient are omitted.

If w is a word, w^R is the word w written backward. This notation is extended to sets of words: $M^R = \bigcup\limits_{w \in M}(\{w^R\})$. Considering the definitions above, the following result is rather obvious.

Proposition 1 *Let $M \subseteq A^*$ and $N \subseteq A^*$. Then $(M/\!/N)^R = N^R\backslash\backslash M^R$.*

The main result of this article is the following.

Theorem *The family of regular languages is closed under full quotient.*

PROOF. In Sect. 3, a finite automaton that recognizes the left full quotient of two regular languages is constructed. As far as the right full quotient is concerned, the proof comes from this construction and Proposition 1. □

Notice that the family of regular languages is not closed under full quotient with arbitrary languages, unlike the usual quotient. It comes from the next example that the left full quotient of a regular language by a deterministic context free language is not necessarily regular.

Example *Consider alphabet $A = \{a, b\}$ and languages $R = a^+b^+a^+b^+$ and $L = \{a^n b^n a^m b^m \mid n \in \mathbb{N} \setminus \{0\},\ m \in \mathbb{N}\}$. Then*

$$L\backslash\backslash R = \{\varepsilon\} \cup b^+ \cup b^+a^+b^+ \cup \{a^n b^m \mid n > m > 0\}\ .$$

Indeed any word $w = a^n b^n$ in L can be completed with a word $u = b^i a^j b^k$ (where $j \neq 0$ and $k \neq 0$) to get a word of R. However, in order to satisfy the prefix-based condition, i, j and k must verify $i \neq 0$ or $(i = 0) \wedge (j > k)$ (i.e. every word $a^j b^k$ with $j \leq k$ is not in $L\backslash\backslash R$ because its prefix $a^j b^j$ is such that $a^n b^n a^j b^j$ is in L). Moreover, notice that any word $w = a^n b^n a^m b^m$ in L can be completed with a word $u = b^i$ with $i \neq 0$.

3 Construction of a Finite Automaton Recognizing Left Full Quotient of Two Regular Languages

In this section, a construction of a finite automaton $\mathcal{C}$ recognizing the left full quotient of a regular language N by a regular language M is described. In order to show the correction of this construction, two automata, $\mathcal{A}$ and $\mathcal{B}$, are built. Automaton $\mathcal{A}$, in which M-prefixed words of N are easily distinguishable, is constructed first. Then $\mathcal{B}$ is built from $\mathcal{A}$; it recognizes all words obtained when removing, from M-prefixed words of N, the prefixes in M. Finally, $\mathcal{C}$ is built from $\mathcal{B}$. A straightforward algorithm producing an automaton recognizing $M \backslash\backslash N$ is provided in Sect. 4. It relies on the constructions of $\mathcal{A}$ and $\mathcal{B}$.

Automata $\mathcal{M}$ and $\mathcal{N}$. Let $\mathcal{M} = (Q_\mathcal{M}, \delta_\mathcal{M}, q_{0_\mathcal{M}}, F_\mathcal{M})$ and $\mathcal{N} = (Q_\mathcal{N}, \delta_\mathcal{N}, q_{0_\mathcal{N}}, F_\mathcal{N})$ be two finite deterministic complete automata recognizing M and N respectively.

Automaton $\mathcal{A}$ is built according to the standard construction of the automaton recognizing $\mathcal{L}(\mathcal{M}) \cap \mathcal{L}(\mathcal{N})$, except that the final states of $\mathcal{A}$ are $Q_\mathcal{M} \times F_\mathcal{N}$. Formally,

$$\mathcal{A} = (Q_\mathcal{A}, \delta_\mathcal{A}, q_{0_\mathcal{A}}, F_\mathcal{A}) = (Q_\mathcal{M} \times Q_\mathcal{N}, \delta_{\mathcal{M}\|\mathcal{N}}, (q_{0_\mathcal{M}}, q_{0_\mathcal{N}}), Q_\mathcal{M} \times F_\mathcal{N})$$

where $\delta_{\mathcal{M}\|\mathcal{N}}$, that simulates the synchronized working of $\mathcal{M}$ and $\mathcal{N}$, is defined as follows:

$$\begin{aligned} \delta_{\mathcal{M}\|\mathcal{N}} : (Q_\mathcal{M} \times Q_\mathcal{N}) \times A &\rightarrow Q_\mathcal{M} \times Q_\mathcal{N} \\ ((q, q'), a) \quad\quad &\mapsto (\delta_\mathcal{M}(q, a), \delta_\mathcal{N}(q', a)) \ . \end{aligned}$$

It is easy to see that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{N})$.

Note 1 Obviously, changing the final states of $\mathcal{A}$ into $F_\mathcal{M} \times Q_\mathcal{N}$, one gets an automaton recognizing $\mathcal{L}(\mathcal{M})$. Consequently, every word of N that is M-prefixed is recognized along a path of $\mathcal{A}$ passing through a state $f \in F_\mathcal{M} \times Q_\mathcal{N}$ (and that the prefix in M is recognized at f).

Automaton $\mathcal{B}$. Careful readers will notice that automaton $\mathcal{B}$ defined below is not deterministic, unlike automata $\mathcal{M}$, $\mathcal{N}$ and $\mathcal{A}$. Indeed, although $\delta_\mathcal{B}$ is a function, $\mathcal{B}$ is provided with a *set* of initial states. Consequently, a word u is recognized by $\mathcal{B}$ if it is the label of a path from *an* initial state to a final state of $\mathcal{B}$.

Now the interest lies in the recognition of the words obtained when removing prefixes belonging to M from every word of N. In view of Note 1, these words are the labels of all the paths in $\mathcal{A}$ from $(F_\mathcal{M} \times Q_\mathcal{N}) \cap R_\mathcal{A}$ to $F_\mathcal{A}$. It is important to only consider the elements of $F_\mathcal{M} \times Q_\mathcal{N}$ that belong to $R_\mathcal{A}$; if not, some words u such that there is no w in A^* satisfying $wu \in N$ could be taken into account.

Let $\mathcal{B}$ be the automaton $(Q_{\mathcal{B}}, \delta_{\mathcal{B}}, Q_{0_{\mathcal{B}}}, F_{\mathcal{B}})$ where $Q_{\mathcal{B}} = Q_{\mathcal{M}} \times Q_{\mathcal{N}}$, $\delta_{\mathcal{B}} = \delta_{\mathcal{A}}$, $Q_{0_{\mathcal{B}}} = (F_{\mathcal{M}} \times Q_{\mathcal{N}}) \cap R_{\mathcal{A}}$ and $F_{\mathcal{B}} = F_{\mathcal{A}}$.

It is not too difficult to realize that

$$\mathcal{L}(\mathcal{B}) = \{u \in A^* \mid \exists\, w \in \mathcal{L}(\mathcal{M}) \,.\, wu \in \mathcal{L}(\mathcal{N})\} = M \backslash N \,. \tag{1}$$

Consequently, we have $\mathcal{L}(\mathcal{M}) \backslash\backslash \mathcal{L}(\mathcal{N}) \subseteq \mathcal{L}(\mathcal{B})$. The next definition helps characterizing the subset of $\mathcal{L}(\mathcal{B})$ that is exactly $\mathcal{L}(\mathcal{M}) \backslash\backslash \mathcal{L}(\mathcal{N})$.

Definition A path in $\mathcal{B}$ has a *suffix-$\mathcal{M}$ prefix* if it has a non-empty prefix, the end of which is in $F_{\mathcal{M}} \times Q_{\mathcal{N}}$.

The next Lemma comes from the above construction of $\mathcal{B}$ from $\mathcal{A}$ and from the definition of "$\backslash\backslash$".

Lemma 2 *Let u be a word in $\mathcal{L}(\mathcal{B})$. Then, u is a word of $\mathcal{L}(\mathcal{M}) \backslash\backslash \mathcal{L}(\mathcal{N})$ if and only if it is the label of a path from $Q_{0_{\mathcal{B}}}$ to $F_{\mathcal{B}}$ that has no suffix-$\mathcal{M}$ prefix.*

PROOF. Let u be a word in $\mathcal{L}(\mathcal{B})$. Then, u is the label of a path P_u in $\mathcal{B}$, the origin of which is in $Q_{0_{\mathcal{B}}}$ and the end of which is in $F_{\mathcal{B}}$.

First, suppose that u is a word in $\mathcal{L}(\mathcal{M}) \backslash\backslash \mathcal{L}(\mathcal{N})$. Then, there exists w in $\mathcal{L}(\mathcal{M})$ such that wu is in $\mathcal{L}(\mathcal{N})$ and that for all v in $\mathsf{pref}_+(u)$, wv is not in $\mathcal{L}(\mathcal{M})$ (a). If P_u had a suffix-$\mathcal{M}$ prefix P_v the label of which is v, we would have $v \in \mathsf{pref}_+(u)$ and $wv \in \mathcal{L}(\mathcal{M})$ (according to suffix-$\mathcal{M}$ prefix definition), which contradicts sentence (a) above. So, P_u has no suffix-$\mathcal{M}$ prefix.

Now, suppose that P_u has no suffix-$\mathcal{M}$ prefix. As u is in $\mathcal{L}(\mathcal{B}) = M \backslash N$, there exists w in $\mathcal{L}(\mathcal{M})$ such that wu is in $\mathcal{L}(\mathcal{N})$. But P_u has no non-empty prefix ending in $F_{\mathcal{M}} \times Q_{\mathcal{N}}$. Consequently, for all v in $\mathsf{pref}_+(u)$, wv does not belong to $\mathcal{L}(\mathcal{M})$. So, u is in $\mathcal{L}(\mathcal{M}) \backslash\backslash \mathcal{L}(\mathcal{N})$. $\square$

Therefore, removing from $\mathcal{B}$ all the suffix-$\mathcal{M}$ prefixes, one can get an automaton recognizing $\mathcal{L}(\mathcal{M}) \backslash\backslash \mathcal{L}(\mathcal{N})$. Construction of automaton $\mathcal{C}$ given below takes its inspiration from this note.

Automaton $\mathcal{C}$. Let $\mathcal{C}$ be the automaton $(Q_{\mathcal{C}}, \delta_{\mathcal{C}}, Q_{0_{\mathcal{C}}}, F_{\mathcal{C}})$ where $Q_{\mathcal{C}} = Q_{\mathcal{B}}$, $Q_{0_{\mathcal{C}}} = Q_{0_{\mathcal{B}}}$, $F_{\mathcal{C}} = F_{\mathcal{B}}$ and $\delta_{\mathcal{C}}$ is defined by

$$\delta_{\mathcal{C}}(q, a) = q' \Leftrightarrow (\delta_{\mathcal{A}}(q, a) = q' \;\text{ and }\; q' \notin F_{\mathcal{M}} \times Q_{\mathcal{N}}) \,.$$

Notice that, as automaton $\mathcal{B}$, $\mathcal{C}$ is non-deterministic with respect to the initial state. Due to Lemma 2, the next proposition holds.

Proposition 3 $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{M})\backslash\backslash\mathcal{L}(\mathcal{N})$.

PROOF. Let u be a word in $\mathcal{L}(\mathcal{M})\backslash\backslash\mathcal{L}(\mathcal{N})$. Then, because of (1), u is recognized by $\mathcal{B}$ along a path P_u from $Q_{0_\mathcal{B}}$ to $F_\mathcal{B}$. According to Lemma 2, P_u has no suffix-$\mathcal{M}$ prefix, that is, P_u is of the form $(q_1, a_1, q_2) \ldots (q_n, a_n, q_{n+1})$ with $q_{i+1} = \delta_\mathcal{B}(q_i, a_i)$ for all i in $\{1, \ldots, n\}$ and q_i is not in $F_\mathcal{M} \times Q_\mathcal{N}$ for all i in $\{2, \ldots, n+1\}$. So, for all i in $\{1, \ldots, n\}$, we have $q_{i+1} = \delta_\mathcal{C}(q_i, a_i)$ due to the definition of $\delta_\mathcal{C}$. Consequently, and by definition of $Q_{0_\mathcal{C}}$ and $F_\mathcal{C}$, P_u is a path in $\mathcal{C}$ from $Q_{0_\mathcal{C}}$ to $F_\mathcal{C}$, that is, u is a word of $\mathcal{L}(\mathcal{C})$.

Now, if u is a word in $\mathcal{L}(\mathcal{C})$, then it is the label of a path P_u in $\mathcal{C}$ from $Q_{0_\mathcal{C}}$ to $F_\mathcal{C}$. According to $\delta_\mathcal{C}$ definition, P_u does not pass through a state of $F_\mathcal{M} \times Q_\mathcal{N}$, except for its origin which is in this latter set. Consequently, P_u has no suffix-$\mathcal{M}$ prefix. As P_u is also a path in $\mathcal{B}$ from $Q_{0_\mathcal{B}}$ to $F_\mathcal{B}$ (by definition of automaton $\mathcal{C}$), we can state, due to Lemma 2, that u is a word of $\mathcal{L}(\mathcal{M})\backslash\backslash\mathcal{L}(\mathcal{N})$. $\square$

4 A Straightforward Algorithm

A straightforward algorithm constructing an automaton recognizing the left full quotient of two regular languages M and N is now provided. The basis of the present work lies on the automata of the previous section. In view of the definition of automaton $\mathcal{C}$, the reachable states of $\mathcal{A}$ have to be considered. This leads to the following definition.

Definition The *reachable part* of $\mathcal{A}$ is the automaton

$$\mathcal{RP}(\mathcal{A}) = (R_\mathcal{A}, \delta_{\mathcal{RP}(\mathcal{A})}, q_{0_\mathcal{A}}, F_\mathcal{A} \cap R_\mathcal{A})$$

where $R_\mathcal{A}$ is the set of all the reachable states of $\mathcal{A}$ as defined in Sect. 1 and $\delta_{\mathcal{RP}(\mathcal{A})}$ is the part of $\delta_\mathcal{A}$ each element of which has its extremities in $R_\mathcal{A}$.

Suppose automaton $\mathcal{RP}(\mathcal{A})$ has been constructed. Then $Q_{0_\mathcal{B}}$ is easily obtained, and automaton $(R_\mathcal{A}, \delta_{\mathcal{RP}(\mathcal{A})}, Q_{0_\mathcal{B}}, F_\mathcal{A} \cap R_\mathcal{A})$ can be built. Let $\delta'_{\mathcal{RP}(\mathcal{A})}$ be the set of all the transitions of $\delta_{\mathcal{RP}(\mathcal{A})}$ that do not lead to a state of $F_\mathcal{M} \times Q_\mathcal{N}$. Automaton $\mathcal{D} = (R_\mathcal{A}, \delta'_{\mathcal{RP}(\mathcal{A})}, Q_{0_\mathcal{B}}, F_\mathcal{A} \cap R_\mathcal{A})$ clearly verifies $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{C})$ due to the definition of $\mathcal{C}$.

The algorithm. In the following, transition functions δ are viewed as sets of transitions (that is as relations); therefore, $(q, a, q') \in \delta$ stands for $\delta(q, a) = q'$.

The algorithm is given in Fig. 1. It constructs $\mathcal{D}$ in a "breadth-first" way from automata $\mathcal{M}$ and $\mathcal{N}$. It starts from the initial state $(q_{0_\mathcal{M}}, q_{0_\mathcal{N}})$ of $\mathcal{RP}(\mathcal{A})$: it looks for each pair $((q_{0_\mathcal{M}}, a, q), (q_{0_\mathcal{N}}, a, q')) \in \delta_\mathcal{M} \times \delta_\mathcal{N}$ and, if $q \notin F_\mathcal{M}$, it constructs the transition $((q_{0_\mathcal{M}}, q_{0_\mathcal{N}}), a, (q, q'))$ of $\delta'_{\mathcal{RP}(\mathcal{A})}$ and stores (q, q') in a queue. When every pair has been processed, the algorithm goes on with the state on the head of the queue.

The algorithm makes use of two particular functions, $\delta_\mathcal{M}(q)$ and $\delta_\mathcal{N}(q, a)$. The first one yields the set of transitions of $\mathcal{M}$, the starting state of which is q. The second one returns the transition of $\mathcal{N}$, the starting state and label of which is, respectively, q and a.

```
Left-full-quotient((Q_M, δ_M, q0_M, F_M), (Q_N, δ_N, q0_N, F_N))
   StatesQueue := Empty-queue;
   Enqueue((q0_M, q0_N), StatesQueue);
   Q := {};   δ := {};   Q_0 := {};   F := {};
   while [¬Empty?(StatesQueue)] do
      (q_M, q_N) := Head(StatesQueue);
      StatesQueue := Dequeue(StatesQueue);
      if [(q_M, q_N) ∉ Q] then          % (i)
         Q := Q ∪ {(q_M, q_N)};
         if [q_M ∈ F_M] then Q_0 := Q_0 ∪ {(q_M, q_N)} end if;
         if [q_N ∈ F_N] then F := F ∪ {(q_M, q_N)} end if;
         for each (q_M, a, q'_M) ∈ δ_M(q_M) do
            (q_N, a, q'_N) := δ_N(q_N, a);
            if [q'_M ∉ F_M] then
               δ := δ ∪ {( (q_M, q_N), a, (q'_M, q'_N) )};
               Enqueue((q'_M, q'_N), StatesQueue);
            end if;
         end for;
      end if;                           % (ii)
   end while;
   return (Q, δ, Q_0, F);
end Left-full-quotient;
```

Fig. 1. Algorithm Left-full-quotient

Time complexity of the algorithm. Every membership test, as well as computations of $\delta_\mathcal{M}(q)$ and $\delta_\mathcal{N}(q, a)$, are assumed to be basic operations.

The set of instructions from (i) to (ii) is executed at most $|Q_{\mathcal{M}}|.|Q_{\mathcal{N}}|$ times. Moreover, for one state (q, q') of the queue, the **for** loop is repeated exactly $|A|$ times because $\mathcal{M}$ is a deterministic complete automaton.

The maximum amount of states that get through the queue is now computed. This amount is obtained supposing that, each time a state (q, q') is dealt with, $|A|$ transitions are built for $\mathcal{A}$, that is, $|A|$ states are enqueued. This leads, on the whole, to a maximum amount of $|Q_{\mathcal{M}}|.|Q_{\mathcal{N}}|.|A|$ states. Thus, the **while** loop is repeated at most $|Q_{\mathcal{M}}|.|Q_{\mathcal{N}}|.|A|$ times. The following proposition has been established.

Proposition *The time complexity of the algorithm* Left-full-quotient *is in* $O(|Q_{\mathcal{M}}|.|Q_{\mathcal{N}}|)$.

Conclusion

A binary operation on languages, named full quotient due to its similarity to usual quotient, has been introduced. This similarity is not very deep, since, as shown in Sect. 2, the family of regular languages is not closed under full quotient with an arbitrary language. However, the main result of this paper establishes that the full quotient of two regular languages is regular. With this aim in view, a construction of an automaton recognizing the full quotient has been described. From this construction, an algorithm has been derived, the time complexity of which is proportional to the product of the number of states of the initial automata.

Closure properties of other families of languages under full quotient have not been investigated. Moreover, one should ask whether the full quotient may be expressed as a finite combination of some familiar operations on languages. A positive answer to that question is rather dubious. In other words, it can be believed that in a sense the full quotient is an elementary operation.

Acknowledgement

The authors wish to thank Marie-Catherine Daniel-Vatonne for helpful discussions.

References

[1] J. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, (11):481–494, 1964.

[2] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[3] D. Perrin. Finite Automata. In Jan van Leeuwen, editor, *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, pages 3–57. Elsevier, 1990.