A Study of a Simple Class of Modifiers: Product Modifiers

Pascal Caron, Edwin Hamel-de-le-court(✉), and Jean-Gabriel Luque

LITIS, Université de Rouen, Avenue de l'Université,
76801 Saint-Étienne du Rouvray Cedex, France
{Pascal.Caron,Jean-Gabriel.Luque}@univ-rouen.fr,
Edwin.Hamel-de-le-court@etu.univ-rouen.fr

Abstract. A modifier is a k-ary operator acting on DFAs and producing a DFA. Modifiers are involved in the theory of state complexity. We define and study a class of simple modifiers, called product modifiers, and we link closely the regular operations they encode to boolean operations.

1 Introduction

State complexity is a measure of complexity defined on regular operations. It allows to write the size of the minimal automaton recognizing the output as a function of the sizes of the minimal automata recognizing the inputs. The topic dates back to the 70s, from the seminal paper of Maslov [14] describing, explicitly but without any proof, the state complexities of several operations. Since the 90s, this area of research became very active and the state complexity of numerous operations has been computed. See, for example, [6,11–13,15] and [8] for a survey of the subject.

However, a few general methods are commonly used in order to compute state complexities. The most common method consists in providing a witness, which is a specific example reaching what is proven to be an upper bound. The witness itself is, in general, found by trial and error, sometimes using a witness that worked for a number of other operations and modifying it to fit the specific needs of the operation considered. In many cases, for example [1,7] or [4], the witness is constructed by considering, explicitly or implicitly, the whole monoid of the transformations acting on the states of the minimal automata recognizing the input languages. This method has been theorized in two independently written papers [2,5]. More precisely, the approach consists, on the one hand, in describing states as combinatorial objects and finding upper bounds using combinatorial tools, and, on the other hand, in building a huge witness, called a *monster*, chosen in a set of automata having as many transition functions as possible. This method can be applied to obtain the state complexity to the wide range of 1-*uniform* operations that are associated to operators, called *modifiers*, that act on automata to produce an automaton in a certain restrictive way. In this paper, we examine the regular operations described by the class of some very simple modifiers called *product modifiers*. These modifiers are characterized

© Springer Nature Switzerland AG 2020
N. Jonoska and D. Savchuk (Eds.): DLT 2020, LNCS 12086, pp. 110–121, 2020.
https://doi.org/10.1007/978-3-030-48516-0_9

by the fact that they build the Cartesian product automaton with the transitions took from the input automata. We investigate many properties of this class and in particular we completely describe the set of the regular operations that can be encoded by product modifiers. The paper is organized as follows. Section 2 gives definitions and notations about automata. In Sect. 3, we partially recall the monster approach. Finally, in Sect. 4, we define product modifiers and characterize the regular operations they encode in Sect. 5.

2 Preliminaries

2.1 Operations over Sets

The *set of subsets* of E is denoted by 2^E and the *set of mappings* of E into itself is denoted by E^E. The *symmetric difference* of two sets E_1 and E_2 is denoted by $\oplus$ and defined by $E_1 \oplus E_2 = (E_1 \cup E_2) \setminus (E_1 \cap E_2)$.

Let $(E_1, \ldots, E_k)$ be a k-tuple of finite sets, and let $(\delta_1, \ldots, \delta_k)$ be a k-tuple such that δ_i is a function from E_i to E_i for every $i \in \{1, \ldots, k\}$. For any k-tuple $(e_1, \ldots, e_k)$ such that $e_i \in E_i$ for all $i \in \{1, \ldots, k\}$, we denote by $(\delta_1, \ldots, \delta_k)(e_1, \ldots, e_k)$ the k-tuple $(\delta_1(e_1), \ldots, \delta_k(e_k))$.

Let E be a set, $f : E^j \to E$ and $g : E^k \to E$ for some $j, k \in \mathbb{N} \setminus \{0\}$. A *composition* is a function $f \circ_p g : E^{j+k-1} \to E$ defined for some $1 \leq p \leq j$ by

$$f \circ_p g(e_1, \ldots, e_{j+k-1}) = f(e_1, \ldots, e_{p-1}, g(e_p, \ldots, e_{p+k-1}), e_{p+k}, \ldots, e_{j+k-1}),$$

for any $e_1, \ldots, e_{j+k-1} \in E$.

2.2 Languages and Automata

Let Σ be a finite alphabet. A *word* w over Σ is a finite sequence of symbols of Σ. The set of all finite words over Σ is denoted by Σ^*. A *language* over Σ is a subset of Σ^*. We define the *complement* of a language $L \subseteq \Sigma^*$ by $L^c = \Sigma^* \setminus L$.

A *complete and deterministic finite automaton* (DFA) is a 5-tuple $A = (\Sigma, Q, i, F, \delta)$ where Σ is the input alphabet, Q is a finite set of states, $i \in Q$ is the initial state, $F \subset Q$ is the set of final states and δ is the transition function from $Q \times \Sigma$ to Q that is defined for every $q \in Q$ and every $a \in \Sigma$. We can extend transition functions in a natural way to functions from $Q \times \Sigma^*$ to Q, and again to functions from $2^Q \times \Sigma^*$ to Q. For any word w, we denote by δ^w the function $q \to \delta(q, w)$.

Let $A = (\Sigma, Q, i, F, \delta)$ be a DFA. A word $w \in \Sigma^*$ is *recognized* by the DFA A if $\delta(i, w) \in F$. The *language recognized* by a DFA A is the set $\mathrm{L}(A)$ of words recognized by A. By Kleene's theorem, a language is regular if and only if it is recognized by a DFA. It is well known that for any DFA, there exists a unique minimal one (up to isomorphism) among all DFAs recognizing the same language ([10]).

2.3 State Complexity

A *unary regular operation* is a function from regular languages of Σ into regular languages of Σ. A *k-ary regular operation* is a function from the set of k-tuples of regular languages over Σ into regular languages over Σ.

The state complexity of a regular language L denoted by $\mathrm{sc}(L)$ is the number of states of its minimal DFA. This notion extends to regular operations: the state complexity of a unary regular operation $\otimes$ is the function $\mathrm{sc}_\otimes$ such that, for all $n \in \mathbb{N}$, $\mathrm{sc}_\otimes(n)$ is the maximum of all the state complexities of $\otimes(L)$ when L is of state complexity n, *i.e.*

$$\mathrm{sc}_\otimes(n) = \max\{\mathrm{sc}(\otimes(L)) | \mathrm{sc}(L) = n\}.$$

This can be generalized, and the state complexity of a k-ary operation $\otimes$ is the k-ary function $\mathrm{sc}_\otimes$ such that, for all $(n_1, \ldots, n_k) \in (\mathbb{N})^k$,

$$\mathrm{sc}_\otimes(n_1, \ldots, n_k) = \max\{\mathrm{sc}(\otimes(L_1, \ldots, L_k)) \mid \text{ for all } i \in \{1, \ldots, k\}, \mathrm{sc}(L_i) = n_i\}.$$

Then, a witness for $\otimes$ is a way to assign to each $(n_1, \ldots, n_k)$, where each n_i is assumed sufficiently big, a k-tuple of languages $(L_1, \ldots, L_k)$ with $\mathrm{sc}(L_i) = n_i$, for all $i \in \{1, \ldots, k\}$, satisfying $\mathrm{sc}_\otimes(n_1, \ldots, n_k) = \mathrm{sc}(\otimes(L_1, \ldots, L_k))$.

3 Modifiers and 1-uniform Operations

We describe a class of regular operations, called 1-uniform which are interesting for the study of state complexity [3,5]. We then define operations on DFA called modifiers, and describe a subset of these operations that correspond to the set of 1-uniform regular operations.

3.1 Definition and First Properties

Definition 1. *Let Σ and Γ be two alphabets. A* morphism *is a function ϕ from Σ^* to Γ^* such that, for all $w, v \in \Sigma^*$, $\phi(wv) = \phi(w)\phi(v)$. Notice that ϕ is completely defined by its value on letters. A morphism ϕ is* 1-uniform *if the image by ϕ of any letter is a letter.*

The preimage $\phi^{-1}(L)$ of a regular language L by a morphism ϕ is regular, see, *e.g.*, [9]. This allows us to introduce the notion of 1-uniform regular operation.

Definition 2. *A k-ary regular operation $\otimes$ is 1-uniform if, for any k-tuple of regular languages $(L_1, \ldots, L_k)$, for any 1-uniform morphism ϕ, we have $\otimes(\phi^{-1}(L_1), \ldots, \phi^{-1}(L_k)) = \phi^{-1}(\otimes(L_1, \ldots, L_k))$.*

Obviously, 1-uniformity is stable by composition. Many well-known regular operations are 1-uniform. See [5] for a non-exhaustive list of examples like the complement, the Kleene star, the reverse, the cyclic shift, and the mirror, all boolean operations and catenation among others.

Each 1-uniform regular k-ary operation corresponds to a construction over DFAs, which is handy when we need to compute the state complexity of its elements. Such a construction on DFAs has some constraints that are described in the following definitions.

Definition 3. *The* state configuration *of a DFA $A = (\Sigma, Q, i, F, \delta)$ is the triplet (Q, i, F).*

Definition 4. *A k-*modifier *is a k-ary operation acting on a k-tuple of DFAs $(A_1, \ldots, A_k)$, on the same alphabet Σ, and producing a DFA $\mathfrak{m}(A_1, ..., A_k)$ such that*

– *its alphabet is Σ,*
– *its state configuration depends only on the state configurations of the DFAs $A_1, \ldots, A_k$,*
– *for any letter $a \in \Sigma$, the transition function of a in $\mathfrak{m}(A_1, \ldots, A_k)$ depends only on the state configurations of the DFAs $A_1, \ldots, A_k$ and on the transition functions of a in each of the DFAs $A_1, ..., A_k$.*

Example 1. For any DFA $A = (\Sigma, Q, i, F, \delta)$, define $\mathfrak{Star}(A) = (\Sigma, 2^Q, \emptyset, \{E | E \cap F \neq \emptyset\} \cup \{\emptyset\}, \delta_1)$, where for any $a \in \Sigma$, $\delta_1^a(\emptyset) = \delta^a(i)$ if $\delta^a(i) \notin F$ and $\delta_1^a(\emptyset) = \delta^a(i)$ otherwise, and, for all $E \neq \emptyset$, $\delta_1^a(E) = \delta^a(E)$ if $\delta^a(E) \cap F = \emptyset$ and $\delta_1^a(E) = \delta^a(E) \cup \{i\}$ otherwise. The modifier $\mathfrak{Star}$ describes a construcion on DFA associated to the Star operation on languages, *i.e.* for all DFA A, $\mathrm{L}(A)^* = \mathrm{L}(\mathfrak{Star}(A))$.

Example 2. For any DFAs $A = (\Sigma, Q_1, i_1, F_1, \delta_1)$ and $B = (\Sigma, Q_2, i_2, F_2, \delta_2)$, let $\mathfrak{Xor}(A, B) = (\Sigma, Q_1 \times Q_2, (i_1, i_2), (F_1 \times (Q_2 \setminus F_2) \cup (Q_1 \setminus F_1) \times F_2), (\delta_1, \delta_2))$. The modifier $\mathfrak{Xor}$ describes the classical construction associated to the symmetrical difference, *i.e* for all DFAs A and B, $\mathrm{L}(A) \oplus \mathrm{L}(B) = \mathrm{L}(\mathfrak{Xor}(A, B))$.

Definition 5. *A k-modifier $\mathfrak{m}$ is* 1-uniform *if, for every pair of k-tuples of DFAs $(A_1, \ldots, A_k)$ and $(B_1, \ldots, B_k)$ such that $\mathrm{L}(A_j) = \mathrm{L}(B_j)$ for all $j \in \{1, \ldots, k\}$, we have $\mathrm{L}(\mathfrak{m}(A_1, \ldots, A_k)) = \mathrm{L}(\mathfrak{m}(B_1, \ldots, B_k))$. In that case, there exists a regular operation $\otimes_{\mathfrak{m}}$ such that, for all k-tuples $(A_1, \ldots, A_k)$ of DFAs, we have $\otimes_{\mathfrak{m}}(\mathrm{L}(A_1), \ldots, \mathrm{L}(A_k)) = \mathrm{L}(\mathfrak{m}(A_1, \ldots, A_k))$. We say that $\mathfrak{m}$* describes *the operation $\otimes_{\mathfrak{m}}$.*

We easily check that, for modifiers, the 1-uniformity is stable by composition.

Claim. Let $\mathfrak{m}_1$ and $\mathfrak{m}_2$ be respectively a j-modifier and a k-modifier describing, respectively, operations $\otimes_1$ and $\otimes_2$. The modifier $\mathfrak{m}_1 \circ_p \mathfrak{m}_2$ describes $\otimes_1 \circ_p \otimes_2$.

The correspondence between 1-uniform modifiers and 1-uniform operations is stated in the following Theorem proved in [3].

Theorem 1. *A k-ary operation $\otimes$ is 1-uniform if and only if there exists a k-modifier $\mathfrak{m}$ such that $\otimes = \otimes_{\mathfrak{m}}$.*

Modifiers have been defined, for the first time, in [2] as a tool to compute state complexity of 1-uniform operations.

3.2 Functional Notations

When there is no ambiguity, for any character $\mathtt{X}$ and any integer k given by the context, we write $\underline{\mathtt{X}}$ for $(\mathtt{X}_1, \cdots, \mathtt{X}_k)$. The number k will often be the arity of the regular operation or of the modifier we are considering.

From Definition 4, any k-modifier $\mathfrak{m}$ can be seen as a 4-tuple of mappings $(\mathfrak{Q}, \mathfrak{i}, \mathfrak{f}, \mathfrak{d})$ acting on k DFAs $\underline{A}$ with $A_j = (\Sigma, Q_j, i_j, F_j, \delta_j)$ to build a DFA $\mathfrak{m}\underline{A} = (\Sigma, Q, i, F, \delta)$, where $Q = \mathfrak{Q}(\underline{Q}, \underline{i}, \underline{F})$, $i = \mathfrak{i}(\underline{Q}, \underline{i}, \underline{F})$, $F = \mathfrak{f}(\underline{Q}, \underline{i}, \underline{F})$ and $\forall a \in \Sigma$, $\delta^a = \mathfrak{d}(\underline{i}, \underline{F}, \underline{\delta^a})$. For the sake of clarity, we do not write explicitly the domains of the 4-tuple of mappings but the reader can derive them easily from the above equalities. Notice that we do not need to point out explicitly the dependency of $\mathfrak{d}$ on $\underline{Q}$ because the information is already contained in $\underline{\delta^a}$. We identify modifiers and such 4-tuples of mappings with each other. Below we revisit the definition of $\mathfrak{Xor}$ according to this formalism.

Example 3. $\mathfrak{Xor} = (\mathfrak{Q}, \mathfrak{i}, \mathfrak{f}, \mathfrak{d})$ where

$$\mathfrak{Q}((Q_1, Q_2), (i_1, i_2), (F_1, F_2)) = Q_1 \times Q_2, \ \mathfrak{i}((Q_1, Q_2), (i_1, i_2), (F_1, F_2)) = (i_1, i_2),$$
$$\mathfrak{f}((Q_1, Q_2), (i_1, i_2), (F_1, F_2)) = F_1 \times (Q_2 \setminus F_2) \cup (Q_1 \setminus F_1) \times F_2,$$
$$\mathfrak{d}((i_1, i_2), (F_1, F_2), (\delta_1, \delta_2)) = (\delta_1, \delta_2).$$

4 Product Modifiers

In this section, we study a kind of simple modifier called *product modifiers* and show that they are closely linked to boolean operations.

Definition 6. *A k-modifier $\mathfrak{m} = (\mathfrak{Q}, \mathfrak{i}, \mathfrak{f}, \mathfrak{d})$ is a* product modifier *if, for any k-tuple of finite sets $\underline{Q}$, for any k-tuple of finite sets $\underline{F}$ such that $F_j \subseteq Q_j$ for all j, and for any $\underline{i} \in Q_1 \times \cdots \times Q_k$*

1. $\mathfrak{Q}(\underline{Q}, \underline{i}, \underline{F}) = Q_1 \times \cdots \times Q_k$.
2. $\forall a \in \Sigma$, $\mathfrak{d}(\underline{i}, \underline{F}, \underline{\delta^a}) = \underline{\delta}^a$, *with* $\underline{\delta}^a(\underline{q}) = (\delta_1^a(q_1), \delta_2^a(q_2), ..., \delta_k^a(q_k))$.

In other words, if $\mathfrak{m}$ is a product modifier, then $\mathfrak{m}\underline{A}$ is the product automaton of the A_j, but with final states $\mathfrak{f}(\underline{Q}, \underline{i}, \underline{F})$ and initial state $\mathfrak{i}(\underline{Q}, \underline{i}, \underline{F})$. Intuitively, product modifiers do not change the transition functions of the automata they act on, but seek only to change their final and initial states. We can easily check that the class of product modifiers is stable by composition.

For the sake of simplicity, in this section, $\mathfrak{m}$ denotes any k-ary product (but not necessarily 1-uniform) modifier and $\underline{A} = (A_1, \ldots, A_k)$ any sequence of k DFAs, with $A_j = (\Sigma, Q_j, i_j, F_j, \delta_j)$. Recall that $\underline{i} = (i_1, \ldots, i_k)$, $\underline{Q} = (Q_1, \ldots, Q_k)$ and $\underline{F} = (F_1, \ldots, F_k)$. We also denote $\mathfrak{m}\underline{A} = (\Sigma, Q', \underline{i}', F', \underline{\delta})$.

We define the complementary product to get an easier access to the intersection of languages and their complement.

Definition 7. *For any k-tuple $\underline{P}$ of finite sets, for any k-tuple $\underline{G}$ of finite sets such that $G_j \subseteq P_j$ for all j, and for any $d \subseteq \{1, 2, ..., k\}$, we define* $\mathrm{cp}(d, \underline{G}, \underline{P}) =$ $X_1 \times \cdots \times X_k$*, where $X_i = P_i \backslash G_i$ if $i \in d$ and $X_i = G_i$ otherwise.*

Example 4. $\mathrm{cp}(\{1,3\},(\{1\},\{2,3\},\{2\}),(\{1,2\},\{1,2,3,4\},\{1,2,3\})) = \{2\} \times \{2,3\} \times \{1,3\}$.

Lemma 1. *The set* $\{\mathrm{cp}(d,\underline{F},\underline{Q}) \mid d \subseteq \{1,\ldots,k\}\}$ *is a partition of* Q'.

Proof. Let $d \neq d'$ and suppose that there exists $j \in d \setminus d'$. For any element $\underline{q} \in \mathrm{cp}(d,\underline{F},\underline{Q})$, we have $q_j \notin F_j$ and, for any element $\underline{q'} \in \mathrm{cp}(d',\underline{F},\underline{Q})$, we have $q'_j \in F_j$. It follows that $\mathrm{cp}(d,\underline{F},\underline{Q}) \cap \mathrm{cp}(d',\underline{F},\underline{Q}) = \emptyset$.

Furthermore, consider an element $\underline{q} \in Q'$ and set $d = \{j \mid q_j \notin F_j\}$. Obviously, $\underline{q} \in \mathrm{cp}(d,\underline{F},\underline{Q})$. This proves our result. □

The following lemma sets a restriction on the form of $\mathfrak{f}$ on each of its entries, given that $\mathfrak{i}$ does not change the initial states in its entries.

Lemma 2. *Assume* $\underline{i'} = \underline{i}$. *If* $\mathfrak{m}$ *is* 1*-uniform then there exists* $E \subseteq 2^{\{1,2,\ldots,k\}}$ *such that* $F' = \bigcup_{d\in E} \mathrm{cp}(d,\underline{F},\underline{Q})$.

Proof. Let us prove the contrapositive statement and assume that there is no set $E \subseteq 2^{\{1,2,\ldots,k\}}$ such that $F' = \bigcup_{d\in E} \mathrm{cp}(d,\underline{F},\underline{Q})$. From Lemma 1, there exists $d \subseteq \{1,2,...,k\}$ such that $F' \cap \mathrm{cp}(d,\underline{F},\underline{Q}) \notin \{\emptyset, \mathrm{cp}(d,\underline{F},\underline{Q})\}$. Let d be such a set and let $G = \mathrm{cp}(d,\underline{F},\underline{Q})$. The idea of the proof is to construct, with the states in G, two k-tuple of automata $\underline{B}$ and $\underline{C}$ that recognize the same languages, and such that $\mathrm{L}(\mathfrak{m}\underline{B})$ and $\mathrm{L}(\mathfrak{m}\underline{C})$ are different.

We distinguish two cases :

- First, suppose that $\underline{i} \in G$. If $\underline{i} \in F'$ then we choose $\underline{j} \in G \setminus F'$, otherwise we choose $\underline{j} \in G \cap F'$. Consider the two k-tuples of DFAs $\underline{B}$ and $\underline{C}$ such that $B_l = (\{a\}, Q_l, i_l, F_l, \beta_l)$ and $C_l = (\{a\}, Q_l, i_l, F_l, \gamma_l)$, where, for all positive integer $l \leq k$, $\beta_l^a(i_l) = j_l$ if $x = i_l$, $\beta_l^a(x) = x$ if $x \in Q_{i_l} \setminus \{i_l\}$, and $\gamma_l^a(x) = x$, for any $x \in Q_{i_l}$. Let us remark that, as $\underline{i}, \underline{j} \in G = \mathrm{cp}(d,\underline{F},\underline{Q})$, i_l and j_l are either both in F_l (if $l \notin d$), or both not in F_l (if $l \in d$) by definition of cp. Therefore, i_l and j_l have the same finality in B_l, which is also their finality in C_l, and either B_l and C_l recognize a^*, or B_l and C_l recognize $\emptyset$.
  As described in Fig. 1, the transition functions $\underline{\beta}$ of $\mathfrak{m}\underline{B}$ and $\underline{\gamma}$ of $\mathfrak{m}\underline{C}$ satisfy $\underline{\beta}^a(\underline{i}) = \underline{j}$ and $\underline{\gamma}^a(\underline{i}) = \underline{i}$.
  The finality of $\underline{i}$ is the same in $\mathfrak{m}\underline{B}$ and $\mathfrak{m}\underline{C}$. However, it is not the same finality as $\underline{j}$ in $\mathfrak{m}\underline{B}$ and $\mathfrak{m}\underline{C}$. Therefore, we have $(a \in \mathrm{L}(\mathfrak{m}\underline{B}) \wedge a \notin \mathrm{L}(\mathfrak{m}\underline{C}))$ or $(a \notin \mathrm{L}(\mathfrak{m}\underline{B}) \wedge a \in \mathrm{L}(\mathfrak{m}\underline{C}))$. As a consequence, $\mathrm{L}(\mathfrak{m}\underline{B}) \neq \mathrm{L}(\mathfrak{m}\underline{C})$ and this implies that $\mathfrak{m}$ is not 1-uniform.
- Suppose now that $\underline{i} \notin G$. Let $\underline{j} \in G \setminus F'$, and let $\underline{j'} \in G \cap F'$. Consider the two k-tuple of DFAs $\underline{B}$ and $\underline{C}$ such that $B_l = (\{a,b\}, Q_l, i_l, F_l, \beta_l)$ and $C_l = (\{a,b\}, Q_l, i_l, F_l, \gamma_l)$, where, for all letters $u \in \{a,b\}$, for all positive integer $l \leq k$ and all $x \in Q_l$,

$$\beta_l^u(x) = \begin{cases} j_l \text{ if } x = i_l \wedge u = a \\ j'_l \text{ if } x = j_l \wedge u = b \\ x \text{ otherwise.} \end{cases} \text{ and } \gamma_l^u(x) = \begin{cases} j'_l \text{ if } x = i_l \wedge u = a \\ j_l \text{ if } x = j'_l \wedge u = b \\ x \text{ otherwise.} \end{cases}$$

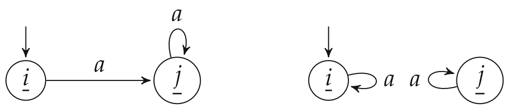

Fig. 1. Part of $\mathfrak{m}\underline{B}$ and $\mathfrak{m}\underline{C}$.

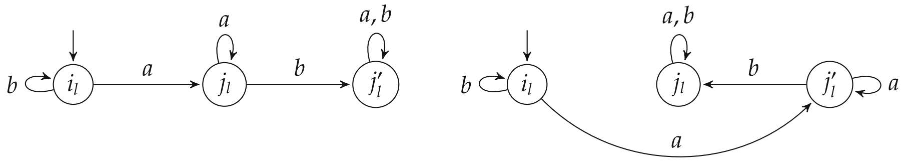

Fig. 2. Parts of B_l and C_l.

For any positive integer $l \leq k$, B_l and C_l recognize the same language. Indeed, from Fig. 2, as $\underline{j}, \underline{j}' \in G = \mathrm{cp}(d, \underline{F}, \underline{Q})$, j_l and j'_l have the same finality in B_l and C_l by definition of cp, we distinguish the cases :

- $i_l \in F_l$ and $j_l \in F_l$. $\mathrm{L}(B_l) = \mathrm{L}(C_l) = (a+b)^*$
- $i_l \in F_l$ and $j_l \notin F_l$. $\mathrm{L}(B_l) = \mathrm{L}(C_l) = b^*$
- $i_l \notin F_l$ and $j_l \in F_l$. $\mathrm{L}(B_l) = \mathrm{L}(C_l) = b^* a(a+b)^*$
- $i_l \notin F_l$ and $j_l \notin F_l$. $\mathrm{L}(B_l) = \mathrm{L}(C_l) = \emptyset$

As $\mathfrak{m}\underline{B}$ and $\mathfrak{m}\underline{C}$ are cartesian products of the B_l and the C_l respectively, if we call $\underline{\beta}$ the transition function of $\mathfrak{m}\underline{B}$ and $\underline{\gamma}$ the transition function of $\mathfrak{m}\underline{C}$, we have $\underline{\beta}^a(\underline{i}) = \underline{j}$, $\underline{\beta}^b(\underline{j}) = \underline{j}'$, $\gamma^a(\underline{i}) = \underline{j}'$, and $\gamma^b(\underline{j}') = \underline{j}$.

The finality of $\underline{j}$ is the same in $\mathfrak{m}\underline{B}$ and $\mathfrak{m}\underline{C}$. However, it is different from the finality of $\underline{j}'$ in $\mathfrak{m}\underline{B}$ and $\mathfrak{m}\underline{C}$. Therefore, we have $(ab \in \mathrm{L}(\mathfrak{m}\underline{B}) \wedge ab \notin \mathrm{L}(\mathfrak{m}\underline{C}))$ or $(ab \notin \mathrm{L}(\mathfrak{m}\underline{B}) \wedge ab \in \mathrm{L}(\mathfrak{m}\underline{C}))$. As a consequence, $\mathrm{L}(\mathfrak{m}\underline{B}) \neq \mathrm{L}(\mathfrak{m}\underline{C})$ which implies that $\mathfrak{m}$ is not 1-uniform. □

The following two lemmas state that, for product modifiers, we can set $\underline{i}' = \underline{i}$ without changing the regular operation associated to $\mathfrak{m}$.

Lemma 3. *If $\mathfrak{m}$ is 1-uniform then $\underline{i}'$ and $\underline{i}$ have the same finality.*

Proof. Let us prove the contrapositive of our statement. Assume that $\underline{i}'$ and $\underline{i}$ do not have the same finality, *i.e.* $(\underline{i} \notin F' \wedge \underline{i}' \in F')$ or $(\underline{i} \in F' \wedge \underline{i}' \notin F')$. Consider the two k-tuples of DFAs $\underline{B}$ and $\underline{C}$ such that $B_l = (\{a\}, Q_l, i_l, F_l, \beta_l)$ and $C_l = (\{a\}, Q_l, i_l, F_l, \gamma_l)$, where, for any $l \in \{1, \ldots, k\}$, $\beta_l^a(i'_l) = i_l$, $\beta_l^a(q) = q$ when $q \neq i'_l$ and $\gamma_l^a(q) = q$. Let us remark that B_l and C_l recognize $\{a\}^*$ if $i_l \in F_l$, and $\emptyset$ otherwise. In any case, they recognize the same language.

If we denote by $\underline{\beta}$ the transition function of $\mathfrak{m}\underline{B}$ and by $\underline{\gamma}$ the transition function of $\mathfrak{m}\underline{C}$, we have $\underline{\beta}^a(\underline{i}') = \underline{i}$ and $\underline{\gamma}^a(\underline{i}') = \underline{i}'$. Recall that $\underline{i}'$ is the initial state of $\mathfrak{m}\underline{B}$ and $\mathfrak{m}\underline{C}$. Since $\underline{i}$ and $\underline{i}'$ do not have the same finality, the word a belongs to one of the languages $\mathrm{L}(\mathfrak{m}\underline{B})$ or $\mathrm{L}(\mathfrak{m}\underline{C})$ but not both (see Fig. 3). Hence the two automata do not recognize the same language and, as a consequence, $\mathfrak{m}$ is not 1-uniform. □

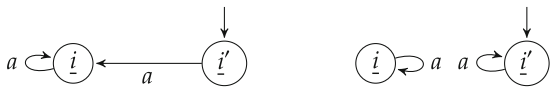

Fig. 3. Part of $\mathfrak{m}\underline{B}$ and $\mathfrak{m}\underline{C}$.

We define an equivalence relation on states of the output of product modifiers whose relationship with the finality of states is examined in Lemma 4.

Definition 8. *Let $\underline{j}$ and $\underline{j}'$ be two k-tuples. We define the equivalence relation $\sim_{\underline{j},\underline{j}'}$ on k-tuples by $(x_1,\ldots,x_k) \sim_{\underline{j},\underline{j}'} (y_1,\ldots,y_k)$ if and only if for all $l \in \{1,\ldots,k\}$, $j_l = j'_l$ implies $x_l = y_l$.*

Example 5. We have $(3,3,2,5,1) \sim_{(1,4,3,2,3),(2,4,2,2,6)} (1,3,5,5,2)$.
We do not have $(3,3,2,5,1) \sim_{(1,4,3,2,3),(2,4,2,2,6)} (1,3,5,1,2)$.

Lemma 4. *If $\mathfrak{m}$ is 1-uniform then* $\mathrm{L}(\mathfrak{m}\underline{A}) = \mathrm{L}((\Sigma, Q', \underline{i}, F', \underline{\delta}))$.

Proof. One has to investigate the two complementary cases:

- *There exists two states $\underline{q} \in F', \underline{q}' \in Q' \setminus F'$ such that $\underline{q} \sim_{\underline{i},\underline{i}'} \underline{q}'$.*
  In this case we prove that $\underline{i} = \underline{i}'$, in other words $\mathfrak{m}A = (\Sigma, Q', \underline{i}, F', \delta)$. Let us show the contrapositive of the property. Suppose $\underline{i} \neq \underline{i}'$. We have to show that $\mathfrak{m}$ is not 1-uniform. By Lemma 3, $\underline{i} \in F' \wedge \underline{i}' \in F'$ or $\underline{i} \notin F' \wedge \underline{i}' \notin F'$. Consider the two k-tuples of DFAs $\underline{B}$ and $\underline{C}$ such that $B_l = (\{a\}, Q_l, i_l, F_l, \beta_l)$ and $C_l = (\{a\}, Q_l, i_l, F_l, \gamma_l)$, where for all $l \in \{1,\ldots,k\}$ and all $q \in Q_l$,

$$\beta_l^a(q) = \begin{cases} q_l \text{ if } q = i'_l \\ q \text{ otherwise} \end{cases} \text{ and } \gamma_l^a(q) = \begin{cases} q'_l \text{ if } q = i'_l \\ q \text{ otherwise.} \end{cases}$$

  Let us remark that either $i'_l = i_l$, which implies $q_l = q'_l$, and $B_l = C_l$, or $i'_l \neq i_l$, and B_l and C_l recognize $\{a\}^*$ if $i_l \in F_l$ and $\emptyset$ otherwise. In any case, they recognize the same language. Recall that $\underline{\beta}$ is the transition function of $\mathfrak{m}\underline{B}$ and $\underline{\gamma}$ is the transition function of $\mathfrak{m}\underline{C}$. We have $\underline{\beta}^a(\underline{i}') = \underline{q}$ and $\underline{\gamma}^a(\underline{i}') = \underline{q}'$. Thus we have $a \in \mathrm{L}(\mathfrak{m}\underline{B})$ and $a \notin \mathrm{L}(\mathfrak{m}\underline{C})$. Therefore, $\mathrm{L}(\mathfrak{m}\underline{B}) \neq \mathrm{L}(\mathfrak{m}\underline{C})$ and $\mathfrak{m}$ is not 1-uniform.

- *For any two states $\underline{q}, \underline{q}' \in Q'$, $\underline{q} \sim_{\underline{i},\underline{i}'} \underline{q}'$ implies that q and q' have the same finality.*

First, for any letter $a \in \Sigma$, any two states $\underline{q}, \underline{q}' \in Q'$, the equivalence $\underline{q} \sim_{\underline{i},\underline{i}'} \underline{q}'$ implies

$$\underline{\delta}^a(\underline{q}) = (\delta_1^a(q_1), \delta_2^a(q_2), \ldots, \delta_k^a(q_k)) \sim_{\underline{i},\underline{i}'} (\delta_1^a(q'_1), \delta_2^a(q'_2), \ldots, \delta_k^a(q'_k)) = \underline{\delta}^a(\underline{q}').$$

This property extends inductively to any word $w \in \Sigma^*$, *i.e.* $\underline{q} \sim_{\underline{i},\underline{i}'} \underline{q}'$ implies $\underline{\delta}^w(\underline{q}) \sim_{\underline{i},\underline{i}'} \underline{\delta}^w(\underline{q}')$. In particular, applying this to $\underline{q} = \underline{i}$ and $\underline{q}' = \underline{i}'$, we have $\delta^w(\underline{i}') \in F'$ if and only if $\delta^w(\underline{i}) \in F'$. As a direct consequence, the languages recognized by the two automata are the same. $\square$

From Lemma 4, one can assume without loss of generality that $\underline{i} = \underline{i}'$. Hence, applying Lemma 2, we obtain

Corollary 1. *If $\mathfrak{m}$ is 1-uniform then there exists $E \subseteq 2^{\{1,2,\ldots,k\}}$ such that $F' = \bigcup_{d \in E} \mathrm{cp}(d, \underline{F}, \underline{Q})$.*

5 Quasi-boolean Operations

Before stating our main result, we need to clarify what is meant by a *boolean operation.* A boolean operation is an operation associated to an expression involving only the operators union, intersection and complement. It is well known that such an expression is equivalent to one written as a union of intersection of languages or their complement. More formally,

Definition 9. *A k-ary boolean operation $\otimes$ over regular languages $L_1, \ldots, L_k$ is defined as*

$$\otimes \underline{L} = \bigcup_{d \in E} \left(\bigcap_{i \in d} L_i \cap \bigcap_{i \notin d} {L_i}^c \right),$$

for some $E \subseteq 2^{\{1,\ldots,k\}}$. Notice that there is a one-to-one correspondence between the boolean k-ary operations and the sets $E \subseteq 2^{\{1,\ldots,k\}}$. So we denote $E_\otimes = E$.

Example 6. The classical boolean operation union can be written this way: for any two regular languages L_1 and L_2,

$$L_1 \cup L_2 = (L_1 \cap {L_2}^c) \cup (L_1 \cap L_2) \cup ({L_1}^c \cap L_2) = \bigcup_{d \in E} \left(\bigcap_{i \in d} L_i \cap \bigcap_{i \notin d} {L_i}^c \right),$$

with $E = \{\{1\}, \{2\}, \{1, 2\}\}$.

We easily check that boolean operations are 1-uniform and can be associated to some product modifiers. More formally,

Lemma 5. *Assume that $\otimes$ is a k-ary boolean operation. Then $\otimes = \otimes_{\mathfrak{m}}$, where $\mathfrak{m} = (\mathfrak{Q}, \mathfrak{i}, \mathfrak{f}, \mathfrak{d})$ is a product modifier such that $\mathfrak{i}(\underline{Q}, \underline{i}, \underline{F}) = \underline{i}$ and*

$$\mathfrak{f}(\underline{Q}, \underline{i}, \underline{F}) = \bigcup_{d \in E_\otimes} \mathrm{cp}(d, \underline{F}, \underline{Q}).$$

From Definition 9, we construct a wider class of operators that we prove to be in correspondence with product modifiers.

Definition 10. *For any k-ary regular operation $\otimes$, for any $\underline{v} \in \{0,1\}^k$, we denote by $\otimes^{\underline{v}}$ the restriction of $\otimes$ to the set*

$$\mathcal{L}^{\underline{v}} = \{(L_1, \ldots, L_k) \mid \forall i \in \{1, \ldots, k\}, L_i \text{ is regular and } v_i = 0 \Leftrightarrow \epsilon \in L_i\}.$$

We say that $\otimes$ is a k-ary quasi-boolean *operation if for all $\underline{v} \in \{0,1\}^k$, $\otimes^{\underline{v}}$ is a boolean operation,* i.e. *for any $\underline{v}$, there exists a boolean operation $\otimes_1$ such that for any $\underline{L} \in \mathcal{L}^{\underline{v}}$ we have $\otimes_1 \underline{L} = \otimes^{\underline{v}} \underline{L}$.*

Example 7. Consider the unary operator defined by $\otimes L = L$ if $\epsilon \in L$ and L^c otherwise. This operation is clearly not boolean. Nevertheless, since for each $L \in \mathcal{L}^{(0)}$ we have $\otimes L = L$ and for each $L \in \mathcal{L}^{(1)}$ we have $\otimes L = L^c$, the operation $\otimes$ is quasi-boolean.

These operations do not have a higher state complexity than boolean operations, as we show in the following statement.

Proposition 1. *For any quasi-boolean k-ary operation $\otimes$, we have*

$$\mathrm{sc}_{\otimes}(n_1, \ldots, n_k) \leq n_1 \cdots n_k.$$

Proof. Lemma 5 implies that $\mathrm{sc}_{\otimes}(n_1, \ldots, n_k) \leq n_1 \cdots n_k$ for any boolean operation $\otimes$. We we prove our statement, by remarking that, for any quasi-boolean operation $\otimes$, we have $\mathrm{sc}_{\otimes}(n_1, \ldots, n_k) \leq \max\{\mathrm{sc}_{\otimes^{\underline{v}}}(n_1, \ldots, n_k) \mid \underline{v} \in \{0,1\}^k\}$. $\square$

We now introduce our main result that characterizes the operations encoded by 1-uniform product modifiers.

Theorem 2. *An operation $\otimes$ is quasi-boolean if and only if there exists a 1-uniform product modifier $\mathfrak{m}$ such that $\otimes = \otimes_{\mathfrak{m}}$.*

Proof. Let $\otimes$ be a k-ary quasi-boolean operation. We construct a modifier $\mathfrak{m}$ such that $\otimes = \otimes_{\mathfrak{m}}$ as follows. We consider the product modifier $\mathfrak{m} = (\mathfrak{Q}, \mathfrak{i}, \mathfrak{f}, \mathfrak{d})$ such that $\mathfrak{i}(\underline{Q}, \underline{i}, \underline{F}) = \underline{i}$ and,

$$\mathfrak{f}(\underline{Q}, \underline{i}, \underline{F}) = \bigcup_{d \in E_{\otimes^{\underline{v}}}} \mathrm{cp}(d, \underline{F}, \underline{Q}),$$

where $\underline{v} \in \{0,1\}^k$ is such that $v_j = 0$ if and only if $i_j \in F_j$.

Let $\underline{L} \in \mathcal{L}^{\underline{v}}$ for some $\underline{v} \in \{0,1\}^k$. For any k-tuple of DFAs $\underline{A}$ such that $A_j = (\Sigma, Q_j, i_j, F_j, \delta_j)$ recognizes L_j, we have $i_j \in F_j$ if and only if $v_j = 0$. From Lemma 5, one has $\mathrm{L}(\mathfrak{m}\underline{A}) = \otimes^{\underline{v}}\underline{L}$. Hence, $\mathfrak{m}$ is $1-uniform$ and $\otimes = \otimes_{\mathfrak{m}}$.

Now, we prove the converse. Let $\otimes$ be a regular operation such that there exists a 1-uniform product modifier $\mathfrak{m}$ satisfying $\otimes_{\mathfrak{m}} = \otimes$. We use a *reductio ad absurdum* argument by assuming that $\otimes$ is not quasi-boolean. Let $\underline{v} \in \{0,1\}^k$ be such that $\otimes^{\underline{v}}$ is not a boolean operation. Let $\underline{A}$ be a k-tuple of DFAs with $A_l = (\Sigma, Q_l, i_l, F_l, \alpha_l)$ such that $(\mathrm{L}(A_1), \ldots, \mathrm{L}(A_k)) \in \mathcal{L}^{\underline{v}}$. Furthermore, we assume that for all $l \in \{1, \ldots, k\}$, $F_l \notin \{\emptyset, Q_l\}$. By Corollary 1, there exists $E \subseteq 2^{\{1,2,\ldots,k\}}$ such that $F = \mathfrak{f}(\underline{Q}, \underline{i}, \underline{F}) = \bigcup_{d \in E} \mathrm{cp}(d, \underline{F}, \underline{Q})$. Therefore, $\mathfrak{m}\underline{A} = \bigcup_{d \in E} \left(\bigcap_{l \in d} \mathrm{L}(A_l) \cap \bigcap_{l \in \{1,2,\ldots,k\} \setminus d} \mathrm{L}(A_l)^c \right)$ which is obviously a boolean operation applied to $(\mathrm{L}(A_1), \ldots, \mathrm{L}(A_k))$. Since $\otimes^{\underline{v}}$ is not a boolean operation, there exists $\underline{A}'$, with $A'_l = (\Sigma', Q'_l, i'_l, F'_l, \alpha'_l)$, a k-tuple of DFAs such that $(\mathrm{L}(A'_1), \ldots, \mathrm{L}(A'_k)) \in \mathcal{L}^{\underline{v}}$ and $\mathfrak{m}\underline{A}' \neq \bigcup_{d \in E} \left(\bigcap_{l \in d} \mathrm{L}(A'_l) \cap \bigcap_{l \in \{1,2,\ldots,k\} \setminus d} \mathrm{L}(A'_l)^c \right)$. We construct new k-tuples of DFAs $\underline{B}$ and $\underline{B}'$ such that $\mathrm{L}(B_l) = \mathrm{L}(B'_l)$ but such that $\mathrm{L}(\mathfrak{m}\underline{B}) \neq \mathrm{L}(\mathfrak{m}\underline{B}')$, contradicting the 1-uniformity of $\mathfrak{m}$. By Corollary 1, there exists $H \subseteq 2^{\{1,\ldots,k\}}$ such that $F' = \mathfrak{f}(\underline{Q}', \underline{i}', \underline{F}') = \bigcup_{d \in H} \mathrm{cp}(d, \underline{F}', \underline{Q}')$.

We have to examine two cases. Either there exists $\underline{p}' \in F'$ such that $\underline{p}' \notin \bigcup_{d \in E} \mathrm{cp}(d, \underline{F}', \underline{Q}')$, or there exists $\underline{p}' \in \bigcup_{d \in E} \mathrm{cp}(d, \underline{F}', \underline{Q}')$ such that $\underline{p}' \notin F'$. We only describe the first case, as the other one is treated symmetrically. Therefore, Lemma 1 implies that there exists $d \in H \setminus E$ such that $\underline{p}' \in \mathrm{cp}(d, \underline{F}', \underline{Q}')$. Let $\underline{p} \in \mathrm{cp}(d, \underline{F}, \underline{Q})$. Notice that $\underline{p} \notin F$ while each p_l has the same finality in B_l as p'_l in B'_l. Also remark that, as $(\mathrm{L}(A_1), \ldots, \mathrm{L}(A_k))$ and $(\mathrm{L}(A'_1), \ldots, \mathrm{L}(A'_k))$ are in $\mathcal{L}^{\underline{v}}$, for all $l \in \{1, \ldots, k\}$, $v_l = 0$ implies $i_l \in F_l$ and $i'_l \in F'_l$, and $v_l = 1$ implies $i_l \notin F_l$ and $i'_l \notin F'_l$.

Now consider the two k-tuples of DFAs $\underline{B}$ and $\underline{B}'$ such that $B_l = (\{a\}, Q_l, i_l, F_l, \beta_l)$ and $B'_l = (\{a\}, Q'_l, i'_l, F'_l, \beta'_l)$, where β_l and β'_l are defined, for all positive integer $l \leq k$ and all $(q, q') \in Q_l \times Q'_l$, by :

$$\beta_l^a(q) = \begin{cases} p_l \text{ if } q = i_l \\ q \ \text{ otherwise.} \end{cases} \text{ and } \beta_l'^a(q') = \begin{cases} p'_l \text{ if } q' = i'_l \\ q' \text{ otherwise.} \end{cases}$$

We notice that, for all $l \in \{1, \ldots, k\}$, B_l and B'_l recognize the same language L_l. Indeed, since i_l and i'_l have the same finality and p_l and p'_l have the same finality, one has to examine four cases which are summarized in Table 1.

Furthermore, we have $\underline{\beta}^a(\underline{i}) = \underline{p} \notin F$, and $\underline{\beta'}^a(\underline{i}') = \underline{p}' \in F'$, which means that $a \notin \mathrm{L}(\mathfrak{m}\underline{B})$ and $a \in \mathrm{L}(\mathfrak{m}\underline{B}')$, which contradicts the 1-uniformity of $\mathfrak{m}$. □

Table 1. Common values of $L(B_l)$ and $L(B'_l)$.

$L(B_l) = L(B'_l)$	$i_l \in F_l$	$i_l \notin F_l$
$p_l \in F_l$	$\{a^*\}$	$\{a\}^+$
$p_l \notin F_l$	$\{\epsilon\}$	$\emptyset$

6 Conclusion

We have shown that some very simple modifiers, namely product modifiers, encode a class of very low state complexity operations. This is a non-trivial example of a set of modifiers closed by composition whose associated regular operations are completely described. The proof techniques open perspectives to explore other classes of modifiers closed by composition. The aim for our future works is to establish a kind of atlas, as complete as possible, of the set of modifiers in relation to the theory of state complexity.

References

1. Brzozowski, J., Jirásková, G., Liu, B., Rajasekaran, A., Szykuła, M.: On the state complexity of the shuffle of regular languages. In: Câmpeanu, C., Manea, F., Shallit, J. (eds.) DCFS 2016. LNCS, vol. 9777, pp. 73–86. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41114-9_6
2. Caron, P., Hame-De-Le-Court, E., Luque, J.G., Patrou, B.: New tools for state complexity. Discret. Math. Theor. Comput. Sci. **22**(1) (2020)
3. Caron, P., Hame-De-Le-Court, E., Luque, J.G.: Algebraic and combinatorial tools for state complexity : application to the star-xor problem. In: Leroux, J., Raskin, J.F. (eds.) Proceedings Tenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2019, Bordeaux, France, 2–3 September 2019, vol. 305 of EPTCS, pp. 154–168 (2019)
4. Caron, P., Luque, J.-G., Mignot, L., Patrou, B.: State complexity of catenation combined with a boolean operation: a unified approach. Int. J. Found. Comput. Sci. **27**(6), 675–704 (2016)
5. Davies, S.: A general approach to state complexity of operations: formalization and limitations. In: Hoshi, M., Seki, S. (eds.) DLT 2018. LNCS, vol. 11088, pp. 256–268. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98654-8_21
6. Domaratzki, M.: State complexity of proportional removals. J. Automata Lang. Comb. **7**(4), 455–468 (2002)
7. Domaratzki, M., Okhotin, A.: State complexity of power. Theor. Comput. Sci. **410**(24–25), 2377–2392 (2009)
8. Gao, Y., Moreira, N., Reis, R., Sheng, Y.: A survey on operational state complexity. J. Automata Lang. Comb. **21**(4), 251–310 (2017)
9. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley, Boston (2007). Pearson international edition
10. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Boston (1979)
11. Jirásek, J., Jirásková, G., Szabari, A.: State complexity of concatenation and complementation. Int. J. Found. Comput. Sci. **16**(3), 511–529 (2005)
12. Jirásková, G.: State complexity of some operations on binary regular languages. Theor. Comput. Sci. **330**(2), 287–298 (2005)
13. Jirásková, G., Okhotin, A.: State complexity of cyclic shift. ITA **42**(2), 335–360 (2008)
14. Maslov, A.N.: Estimates of the number of states of finite automata. Soviet Math. Dokl. **11**, 1373–1375 (1970)
15. Sheng, Y.: State complexity of regular languages. J. Automata Lang. Comb. **6**(2), 221 (2001)