The Shortest Identities for Max-Plus Automata with Two States

Laure Daviaud[1] and Marianne Johnson[2]

[1]University of Warsaw, Poland
[2]School of Mathematics, University of Manchester, Manchester M13 9PL, UK.

This work was partially supported by the LIPA project, funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 683080).

Abstract

Max-plus automata are quantitative extensions of automata designed to associate an integer with every non empty word. A pair of distinct words is said to be an identity for a class of max-plus automata if each of the automata in the class computes the same value on the two words. We give the shortest identities holding for the class of max-plus automata with two states. For this, we exhibit an interesting list of necessary conditions for an identity to hold. Moreover, this result provides a counter-example of a conjecture of Izhakian, concerning the minimality of certain identities.

1 Introduction

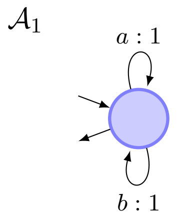
A natural question when dealing with computational models is to understand which pairs of inputs can be separated by the model, *i.e.* lead to different results. Or conversely, which pairs of distinct inputs will give the same computation. These pairs are called identities for the model.

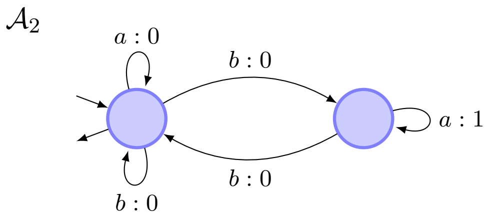
Regarding finite automata, two words are said to be separated by a given automaton if one is accepted and the other is rejected. When fixing an automaton, or even considering the class of automata with at most a certain number of states, we know that some pairs of distinct words are not separated. It is a simple argument of cardinality: the number of automata with a bounded number of states is finite and each of them computes a boolean value on a given word, while the number of words is infinite. However, when considering the full class, for every pair of distinct words, it is easy to construct an automaton accepting one and rejecting the other.

When dealing with quantitative extensions of automata, namely weighted automata, the situation is much more intricate. Weighted automata were introduced by Schützenberger in [12]. They compute functions from the set of words to the set of values of a semiring, allowing one to model quantities such as costs, gains or probabilities. The question of separating words (*i.e.* computing different values on the words) highly depends on the semiring. For probabilistic automata, or automata on the usual semiring $(\mathbb{R}, +, \times)$, it is known that there is an automaton (with two states) which separates every pair of distinct words.

In this paper we are interested in max-plus automata, which are weighted automata over the tropical semiring that compute functions from the set of *non-empty* words to the values of the semiring $\mathbb{Z}_{\max} = (\mathbb{Z} \cup \{-\infty\}, \max, +)$. From now on, for simplicity, we may use the notation $\mathbb{Z}_{\max}$ to denote the semiring or the set of its values $\mathbb{Z} \cup \{-\infty\}$. We note that some authors prefer to work with min-plus automata, which compute values in the min-plus semiring $\mathbb{Z}_{\min} = (\mathbb{Z} \cup \{+\infty\}, \min, +)$. Since the two semirings ($\mathbb{Z}_{\max}$ and $\mathbb{Z}_{\min}$) are isomorphic, the results presented here can be easily translated to the min-plus case.

A *max-plus automaton* is a finite automaton whose transitions are weighted by integers. An easy way to think about these weights is to consider them as amounts of money that you win when you go through a transition. Along a run, you accumulate this money (you sum the amounts; this sum is called the *weight of the run*), and your purpose is, given a word w, to go from an initial state to a final state, by reading w and grabbing the maximal amount of money you can. The *value (or weight) associated with* w is the maximum possible amount that you could win by reading the word w. Max-plus automata are thus particularly suitable to model gain maximisation, study the worst-case complexity of a program [2] or evaluate performance of discrete event systems [4, 5]. Let us give two examples on the alphabet $\{a, b\}$ (where initial and final states are denoted by ingoing and outgoing arrows respectively):





The automaton $\mathcal{A}_1$ associates with each word its length. The function computed by the automaton $\mathcal{A}_2$ is more complicated and we begin by describing its behaviour in particular cases. Consider a word of the form $ba^{k_1}ba^{k_2}b \cdots ba^{k_\ell}b$ where all the k_i are positive integers. Then, the value computed by the automaton is the maximum of the sums $k_{i_1} + k_{i_2} + \cdots + k_{i_m}$ where no two i_j are consecutive, that is to say, $i_{j+1} \geqslant i_j + 2$ for all j.

Two distinct words u and v are separated by a class of max-plus automata if there is an automaton in the class that associates two different values on the two words, otherwise they form an identity for the class, usually denoted $u = v$. But this question is much more intricate than in the previous cases of boolean automata and automata weighted over the usual semiring. We can show that a single max-plus automaton cannot separate all pairs of distinct words. It is again a simple cardinality argument: if the weights of the transitions of an automaton are between $-m$ and m, then the value associated to a word of length n is between $-mn$ and mn, while the number of words of length n on a finite alphabet Σ is $|\Sigma|^n$. For n large enough, there must exist two distinct words having the same value. It is also clear that given two distinct words, one can construct a max-plus automaton (with an arbitrarily large number of states) separating them. A major open question is the following:

Given a bound d, does there exist an identity for the set of max-plus automata with at most d states?

In that case, a simple cardinality argument fails. This question was first considered in [9] where it was answered positively for $d = 2$. The known identity for two states consists of a pair of words of length 20, but the problem seems very difficult to tackle in the general case. Shitov [13] proposed an identity for $d = 3$ consisting of a pair of words of length 1795308. Currently no generalisation of these results seems conceivable, the ultimate (very far) goal being to characterise the complete set of identities. This paper is motivated by the fact that a better understanding of the identities in the case $d = 2$ is already a first step for a better understanding of the general case.

Contribution. We focus on the class of max-plus automata with two states, denoted $\mathcal{C}$. It is easy to see that if $u = v$ is an identity which holds in $\mathcal{C}$ then u and v have the same length (see for example $\mathcal{A}_1$), defining the *length of the identity.* We give the unique two identities of minimal length (17) which hold in $\mathcal{C}$.

Theorem 1. *There are two identities (up to a renaming of the letters) of minimal length which hold in the class of max-plus automata with two states:*

$$a^2b^3a^3babab^3a^2 = a^2b^3ababa^3b^3a^2 \quad \textit{and} \quad ab^3a^4baba^2b^3a = ab^3a^2baba^4b^3a$$

To achieve this goal, we give a rather short list of necessary conditions for an identity to hold which together eliminate all the other possible candidates of length shorter than 18 (Proposition 3, Section 3). This list is short enough that it can be tested by computer. We also prove that this list is minimal in the sense that each of the conditions eliminates at least one pair of words, that cannot be eliminated using the other conditions alone. However, this list is probably not complete, and future works will consist in trying to extend it to fully characterise the identities holding in $\mathcal{C}$. We then prove that the identities given in the statement of Theorem 1 hold in $\mathcal{C}$ (Proposition 4, Section 4).

Link with matrices. This topic is closely related with the question of identities on semigroups of matrices over the tropical semiring. We consider matrices with entries in $\mathbb{Z} \cup \{-\infty\}$ and the product AB for two matrices A, B (provided the number of columns of A and the number of rows of B coincide, denoted here d) is defined as $(AB)_{i,j} = \max_{1 \leqslant k \leqslant d}(A_{i,k} + B_{k,j})$.

An identity $u = v$ is said to be satisfied by a semigroup of matrices if for all substitutions of the letters by matrices in the semigroup, the equality holds.

A max-plus automaton with d states can equivalently be represented by a semigroup of matrices of dimension d: the states are numbered $\{1, \ldots, d\}$ and for each letter, a square matrix $\mu(a)$ of dimension d is defined such that the (i, j)-coefficient contains the weight of the transition from state i to state j labelled by a or $-\infty$ if there is no such transition. Then μ extends to give a semigroup morphism $\mu : \Sigma^+ \to M_n(\mathbb{Z}_{\max})$. For a non-empty word w, it is straightforward to verify that $\mu(w)_{i,j}$ is the maximum of the weights of the runs from state i to state j, labelled by w. An initial vector I (*resp.* final vector F) with 1 row and d columns (*resp.* d rows and 1 column) and entries in $\{0, -\infty\}$ is defined by $I_i = 0$ (*resp.* $F_i = 0$) if and only if state i is initial (*resp.* final). The weight of a word w in $\mathcal{A}$ is exactly the value given by $I\mu(w)F \in \mathbb{Z}_{\max}$ (see for example [11] for more explanations). The max-plus automaton $\mathcal{A}_1$ illustrated on the previous page is represented by $\mu(a) = (1)$, $\mu(b) = (1)$ and $I = F = (0)$, while $\mathcal{A}_2$ is represented by:

$$\mu(a) = \begin{pmatrix} 0 & -\infty \\ -\infty & 1 \end{pmatrix} \qquad \mu(b) = \begin{pmatrix} 0 & 0 \\ 0 & -\infty \end{pmatrix} \qquad I = \begin{pmatrix} 0 & -\infty \end{pmatrix} \qquad F = \begin{pmatrix} 0 \\ -\infty \end{pmatrix}$$

Using this representation, it can be easily shown that $u = v$ is an identity which holds in $\mathcal{C}$ if and only if $u = v$ holds for the semigroup of square matrices of dimension 2.

Proposition 1. *An identity holds in $\mathcal{C}$ if and only if it holds for the semigroup of tropical square matrices of dimension* 2.

Proof. Consider an identity $u = v$ holding in $\mathcal{C}$. For every letter a, let M_a be a square tropical matrix of dimension 2, and denote by M_u and M_v the products obtained by substituting every letter a in u an v by M_a respectively. For all $i, j \in \{1, 2\}$, construct the automata $\mathcal{A}_{i,j}$ defined by $\mu(a) = M_a$ and such that the state i (*resp.* j) is initial (*resp.* final); $[\![\mathcal{A}_{i,j}]\!]$ denotes the function computed by the automaton. Then, $(M_u)_{i,j} = [\![\mathcal{A}_{i,j}]\!](u) = [\![\mathcal{A}_{i,j}]\!](v) = (M_v)_{i,j}$.

Conversely, consider an identity $u = v$ satisfied by the semigroup of tropical square matrices of dimension 2. Let $\mathcal{A} \in \mathcal{C}$ and μ, I, F its matrix definition. By substituting every letter a by the matrix $\mu(a)$ in u and v, we obtain that $[\![\mathcal{A}]\!](u) = I\mu(u)F = I\mu(v)F = [\![\mathcal{A}]\!](v)$. □

Then Theorem 1 implies the following theorem.

Theorem 2. *There are two identities (up to a renaming of the letters) of minimal length which hold in the semigroup of tropical square matrices of dimension 2:*

$$a^2b^3a^3babab^3a^2 = a^2b^3ababa^3b^3a^2 \quad \textit{and} \quad ab^3a^4baba^2b^3a = ab^3a^2baba^4b^3a$$

Using the fact that the identities which hold in the semigroup of tropical square matrices of dimension 2 are the same as those which hold in the semigroup of tropical square matrices of dimension 2 with real entries (as explained in Section 2 below), Theorem 2 gives a counter-example to a conjecture of Izhakian concerning the structure of the identities of minimal length. Indeed, in [8, Conjecture 5.1] he provides a method of constructing identities satisfied by every subsemigroup of the semigroup of the tropical square matrices of dimension d consisting of matrices with maximal (tropical) rank (see [8] for detailed definitions), conjecturing that certain amongst these are of minimal length, but for $d = 2$, the shortest identities produced by this method have length greater than 17.

Organisation of the paper. In Section 2, we give first properties. In particular, we make some comments about working with weights in $\mathbb{Z}$ rather than $\mathbb{N}$, $\mathbb{Q}$ or $\mathbb{R}$, restricting the automata to have only one initial and one final state and considering only 2-letter alphabets. In Section 3, we give the list of conditions allowing us to eliminate all the pairs of words up to length 17 except two (up to renaming of the letters). In Section 4, we prove that these pairs do indeed form identities.

2 First properties

Given a word w and a letter a, we write $|w|$ to denote the length of w and $|w|_a$ to denote the number of occurrences of the letter a in w. If $w = w_0w_1 \cdots w_\ell$ with $w_0, w_1, \ldots, w_\ell$ letters, the positions of w are $0, 1, \ldots \ell$ and w_i is said to be the letter at position i. In a max-plus automaton $\mathcal{A}$, a run labelled by w from a state p to a state q with weight α will be denoted $p \xrightarrow{w\,:\,\alpha} q$. We denote by $[\![\mathcal{A}]\!]$ the function (from the set of non empty words over a finite alphabet Σ to $\mathbb{Z}_{\max}$) computed by $\mathcal{A}$. Let us recall that $\mathcal{C}$ denotes the class of all the max-plus automata with two states. More generally, for any positive integer d, we denote by $\mathcal{C}_d$ the class of all the max-plus automata with d states (so that $\mathcal{C}_2 = \mathcal{C}$). An identity over Σ, that is to say a pair of two distinct non empty words over Σ, denoted $u = v$, holds in $\mathcal{C}_d$ if and only if for all $\mathcal{A} \in \mathcal{C}_d$, $[\![\mathcal{A}]\!](u) = [\![\mathcal{A}]\!](v)$. For now, we fix an integer $d \geqslant 2$.

Content. If $\Sigma = \{a_1, \dots a_n\}$, the *content* of a word w is the n-tuple $(|w|_{a_1}, \dots, |w|_{a_n})$ of the number of occurrences of each of the letters in w.

Lemma 1. *If $u = v$ holds in $\mathcal{C}_d$, then u and v have the same content. In particular u and v have the same length.*

Proof. The number of occurrences of a letter a can be computed by a max-plus automaton with one state (both initial and final) with one transition for each letter of Σ, where the transition labelled by a has weight 1 and all other transitions have weight 0. (Note that this can be seen as a max-plus automaton with d states by simply adding states, and possibly transitions with weight 0). Thus, if the content of u and v differs then there exist an automaton in $\mathcal{C}_d$ computing two different values on these two words. □

Initial and final states. In the rest of the paper, we will freely use the following fact:

Lemma 2. *An identity $u = v$ holds in $\mathcal{C}_d$ if and only if it holds in the class of max-plus automata with d states having exactly one initial and one final state.*

Proof. Denote by $\mathcal{C}'_d$ the class of max-plus automata with d states and exactly one initial and one final state. Clearly, if $u = v$ holds in $\mathcal{C}_d$, it must also hold in $\mathcal{C}'_d$. Conversely, suppose $u = v$ holds in $\mathcal{C}'_d$ and let $\mathcal{A} \in \mathcal{C}_d$. Consider now the set $\mathcal{S}$ of the max-plus automata in $\mathcal{C}'_d$ obtained from $\mathcal{A}$ with a unique initial state chosen from amongst the initial states of $\mathcal{A}$ and a unique final state chosen from amongst the final states of $\mathcal{A}$. Since $u = v$ holds in $\mathcal{C}'_d$, we get:

$$[\![\mathcal{A}]\!](u) = \max_{\mathcal{B} \in \mathcal{S}} ([\![\mathcal{B}]\!](u)) = \max_{\mathcal{B} \in \mathcal{S}} ([\![\mathcal{B}]\!](v)) = [\![\mathcal{A}]\!](v)$$

and thus $u = v$ holds in $\mathcal{C}_d$. □

Weights. The set of identities which hold in $\mathcal{C}_d$ does not change when restricting the weights to have values in $\mathbb{N}$ or when allowing them to take values in $\mathbb{Q}$ or $\mathbb{R}$. Some directions are clear by definitions. We give ideas for the others.

From $\mathbb{Z}$ to $\mathbb{N}$. Consider an identity $u = v$ which holds in the class of d-state max-plus automata with weights in $\mathbb{N}$. It follows from the proof of Lemma 1 that $|u| = |v|$. Now let $\mathcal{A} \in \mathcal{C}_d$ and consider the max-plus automaton $\mathcal{A}_k$ obtained from $\mathcal{A}$ by adding the same integer k to the weight of all transitions in $\mathcal{A}$. Since $\mathcal{A}$ has finitely many transitions it is clear that we can choose k large enough so that $\mathcal{A}_k$ has weights in $\mathbb{N}$. Then we get, $[\![\mathcal{A}]\!](u) = [\![\mathcal{A}_k]\!](u) - k|u| = [\![\mathcal{A}_k]\!](v) - k|v| = [\![\mathcal{A}]\!](v)$, from which it follows that $u = v$ holds for all $\mathcal{A} \in \mathcal{C}_d$.

From $\mathbb{Q}$ to $\mathbb{Z}$. Consider an identity $u = v$ that holds in $\mathcal{C}_d$ and let $\mathcal{A}$ be a d-state max-plus automaton with weights in $\mathbb{Q}$. By multiplying all the weights on the transitions of $\mathcal{A}$ by a suitable non-zero integer k (*e.g.* the *lcm* of the denominators), we get a max-plus automaton $\mathcal{A}_k$ with weights in $\mathbb{Z}$, such that $[\![\mathcal{A}]\!](u) = \frac{1}{k}[\![\mathcal{A}_k]\!](u) = \frac{1}{k}[\![\mathcal{A}_k]\!](v) = [\![\mathcal{A}]\!](v)$.

From $\mathbb{R}$ to $\mathbb{Q}$. Consider an identity $u = v$ that holds in the class of d-state max-plus automata with weights in $\mathbb{Q}$ and let $\mathcal{A}$ be a d-state max-plus automaton with weights in $\mathbb{R}$. Let $(\mathcal{A}_m)_{m \in \mathbb{N}}$ be a sequence of max-plus automata constructed from $\mathcal{A}$ by changing all the real weights to rational weights in such a way that for every transition of $\mathcal{A}$ weighted by α, the sequence of weights $\alpha_m \in \mathbb{Q}$ of the corresponding transitions in $\mathcal{A}_m$ tends to α. Since limits can be commuted with maximum and sum over finite sets, we have:

$$[\![\mathcal{A}]\!](u) = \lim_{m \to \infty} [\![\mathcal{A}_m]\!](u) = \lim_{m \to \infty} [\![\mathcal{A}_m]\!](v) = [\![\mathcal{A}]\!](v)$$

Finally, we show that we need only to consider *full automata*. An automaton is said to be full if for every pair of states p, q and every letter a, there is a transition from p to q labelled by a.

Lemma 3. *An identity $u = v$ holds in $\mathcal{C}_d$ if and only if it holds in the subclass of $\mathcal{C}_d$ consisting of full automata.*

Proof. The if direction is clear by definition. For the converse direction, consider an identity $u = v$ which holds in the subclass of $\mathcal{C}_d$ consisting of full automata. Suppose for contradiction that $\mathcal{A} \in \mathcal{C}_d$ is an automaton falsifying the identity. For each integer k, construct the full automaton $\mathcal{A}_k$ from $\mathcal{A}$ by adding in any missing transitions and weighting these by k. If $[\![\mathcal{A}]\!](u) = -\infty$ (meaning that there is no accepting run on u) then for all k, the accepting runs on u in $\mathcal{A}_k$ necessarily take a transition weighted by k (we suppose that $\mathcal{A}$ has at least one initial and one final state). By assumption, $[\![\mathcal{A}]\!](v)$ must be finite, and denote by m the maximal weight on a transition of $\mathcal{A}$. Then, consider k smaller than $[\![\mathcal{A}]\!](v) - (|u| - 1)m$. We get $[\![\mathcal{A}_k]\!](v) = [\![\mathcal{A}_k]\!](u) \leqslant k + (|u| - 1)m < [\![\mathcal{A}]\!](v)$, which contradicts the fact that $[\![\mathcal{A}]\!](v)$ is finite. The same reasoning holds if $[\![\mathcal{A}]\!](v) = -\infty$. Otherwise, if $[\![\mathcal{A}]\!](u)$ and $[\![\mathcal{A}]\!](v)$ are both finite, and by considering k small enough, $[\![\mathcal{A}]\!](u) = [\![\mathcal{A}_k]\!](u) = [\![\mathcal{A}_k]\!](v) = [\![\mathcal{A}]\!](u)$. □

Number of letters. An identity on a 2-letter alphabet can be seen as an identity over a larger alphabet and it is easy to see that for all $k \geqslant 2$ the identity holds in the class of d-state max-plus automata over two letters if and only if it holds in the class of d-state max-plus automata over k letters.

Suppose now that $u = v$ is an identity holding in $\mathcal{C}_d$ over an alphabet Σ containing at least three letters. Since u and v are distinct, they must differ in some position, i say. Suppose then that $u_i \neq v_i$. Now, consider $\bar{u}$ and $\bar{v}$ obtained from u and v by replacing every letter, except v_i, by u_i. By construction $\bar{u}$ and $\bar{v}$ are distinct. We are going to prove that $\bar{u} = \bar{v}$ holds in the class of max-plus automata over Σ. Indeed, consider a d-state max-plus automaton $\mathcal{A}$ over Σ. Construct first an automaton $\mathcal{A}'$ obtained from $\mathcal{A}$ by removing all the transitions not labelled by u_i or v_i. Then construct an automaton $\mathcal{B}$ over Σ obtained from $\mathcal{A}'$, by adding copies of the transitions labelled by u_i for all the other letters, except v_i, *i.e.* for every transition $p \xrightarrow{u_i\,:\,\alpha} q$, and every letter $c \neq v_i$, add the transition $p \xrightarrow{c\,:\,\alpha} q$. Then,

$$\begin{aligned}
[\![\mathcal{A}]\!](\bar{u}) &= [\![\mathcal{A}']\!](\bar{u}) && \text{since } \bar{u} \text{ contains only } u_i\text{'s and } v_i\text{'s}\\
&= [\![\mathcal{B}]\!](\bar{u}) && \text{since } \bar{u} \text{ contains only } u_i\text{'s and } v_i\text{'s}\\
&= [\![\mathcal{B}]\!](u) && \text{since every letter } c \neq v_i \text{ mimics } u_i \text{ in } \mathcal{B}\\
&= [\![\mathcal{B}]\!](v) && \text{since } u = v \text{ is an identity over } \Sigma \text{ holding in } \mathcal{C}_d\\
&= [\![\mathcal{B}]\!](\bar{v}) && \text{since every letter } c \neq v_i \text{ mimics } u_i \text{ in } \mathcal{B}\\
&= [\![\mathcal{A}]\!](\bar{v}) && \text{since } \bar{v} \text{ contains only } u_i\text{'s and } v_i\text{'s}
\end{aligned}$$

Thus, if an identity over Σ holds in $\mathcal{C}_d$ then an identity of the same length using just two letters must also hold in $\mathcal{C}_d$.

Since we are interested in minimal length identities, in the rest of the paper we will consider only 2-letter alphabets.

3 Minimality

As explained at the end of the previous section, from now on we fix a 2-letter alphabet $\Sigma = \{a, b\}$.

In this section, we provide a list of conditions which must be all satisfied by the identities holding in $\mathcal{C}$. Thanks to this list and aided by a computer, we are left with exactly two pairs of words (up to exchanging a and b) of length shorter than 18 which are still candidates to be identities in $\mathcal{C}$. In the next section, we prove that they are indeed identities in $\mathcal{C}$.

3.1 Triangular identities

A max-plus automaton with two states p and q is said to be *triangular* if there is no transition either from p to q or from q to p. We denote by $\mathcal{C}_{\mathcal{T}}$ this class of automata. An identity holding in $\mathcal{C}$ must also hold in $\mathcal{C}_{\mathcal{T}}$.

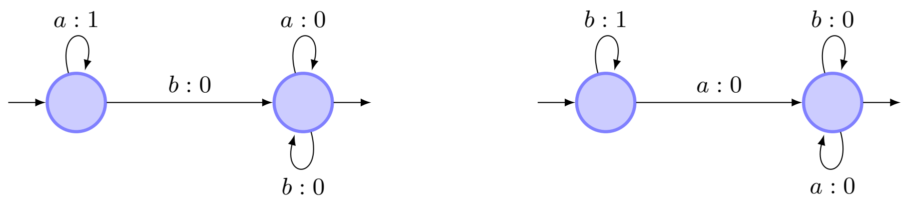
Identities holding in the class of triangular automata are much easier to study. They are fully characterised in [3], where it is proved that they are exactly the identities holding in the bicyclic monoid. More generally, several works [6, 7, 10, 1] study identities holding in the class of triangular max-plus automata with d states, which correspond to the semigroup of upper-triangular matrices, where it has been proved that such an identity always exists.

Let us recall that if $u = v$ holds in $\mathcal{C}_{\mathcal{T}}$, then u and v have the same content (since the automata constructed in Lemma 1 are indeed triangular).

Beginning and end of a word. The *first* (*resp. second*, *last*, *penultimate*) *block* of a word w is the first (*resp.* second, last, penultimate) maximal block of the same consecutive letter of w. For example, for $w = a^3b^2a^6b^7a^4b$, these blocks are respectively a^3, b^2, b and a^4.

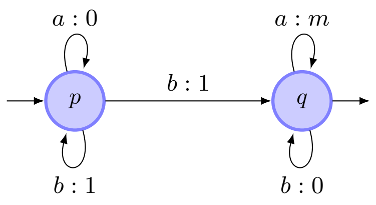
Lemma 4. *If $u = v$ is an identity holding in $\mathcal{C}_{\mathcal{T}}$, then u and v have the same first, second, last and penultimate blocks respectively.*

Proof. The left-most automaton in the following picture computes the length of the first block of a if the word starts with an a, and 0 otherwise, whilst the right-most automaton computes the length of the first block of b if the word begins with a b and 0 otherwise.



Similar automata can be constructed to compute the length of the last block, proving that if $u = v$ is an identity holding in $\mathcal{C}_{\mathcal{T}}$, then u and v should have the same first and last blocks, respectively.

Now suppose that $u = v$ holds in $\mathcal{C}_{\mathcal{T}}$. Then by the above argument, u and v both start with the same block. Without loss of generality we may suppose that this block is of the form a^k for some positive integer k. Moreover by Lemma 1 they also have the same number of occurrences of the letters a and b, denoted n and m respectively. Let us denote by b^j and b^ℓ the second blocks of u and v respectively. Consider the following max-plus automaton:



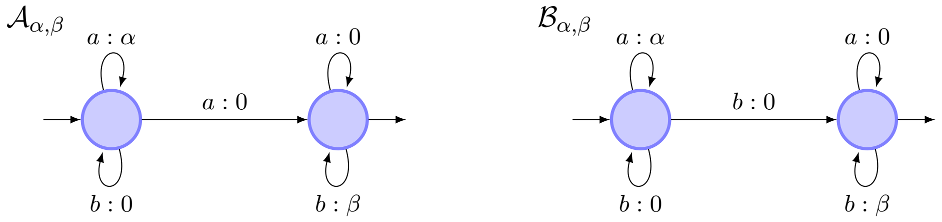
Since m is the number of occurrences of the letter b in both u and v, then on a run labelled by u or v, it is always preferable to traverse the transition $q \xrightarrow{a\,:\,m} q$ at state q at least once. It then follows easily that the maximal accepting runs on u and v respectively begin thus:

$$p \xrightarrow{a^k\,:\,0} p \xrightarrow{b^{j-1}\,:\,j-1} p \xrightarrow{b\,:\,1} q \xrightarrow{a\,:\,m} \dots \quad \text{and} \quad p \xrightarrow{a^k\,:\,0} p \xrightarrow{b^{\ell-1}\,:\,\ell-1} p \xrightarrow{b\,:\,1} q \xrightarrow{a\,:\,m} \dots$$

Thus the values computed on u and v are respectively $j + m(n-k)$ and $\ell + m(n-k)$. Since $u = v$ holds in $\mathcal{C}_\mathcal{T}$, we conclude that $j = \ell$.

A similar automaton can be constructed for dealing with the penultimate block. □

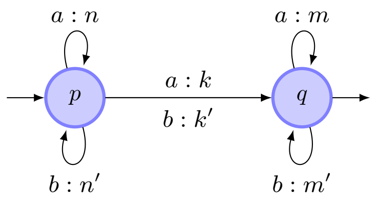
Triangular identities. We give here a variant of a property in [3]. We show that the identities $u = v$ which hold in $\mathcal{C}_\mathcal{T}$ are exactly those such that u and v have the same content, the same first and last blocks, and which hold in the class of max-plus automata of one of the following shape, where α and β are integers either both positive or both negative:



Let us denote $\mathcal{C}_\mathcal{T}{}'$ the class of automata of one of the previous shape:

Proposition 2. *An identity $u = v$ holds in $\mathcal{C}_\mathcal{T}$ if and only if u and v have the same content, the same first and last blocks, and the identity holds in $\mathcal{C}_\mathcal{T}{}'$.*

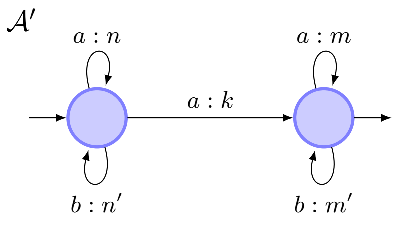
Proof. The if direction is clear by definition and Lemmas 1 and 4. Conversely, suppose that $u = v$ holds in $\mathcal{C}_\mathcal{T}{}'$, and consider $\mathcal{A} \in \mathcal{C}_\mathcal{T}$. First by the proof of Lemma 3, we can suppose that $\mathcal{A}$ is full and by Lemma 2, that $\mathcal{A}$ has exactly one initial and one final states. If they are the same, then since u and v have the same content, $[\![\mathcal{A}]\!](u) = [\![\mathcal{A}]\!](v)$. Otherwise, denote by p the initial state, and by q the final state different from p. Consider $\mathcal{A}$ as depicted in the following picture:

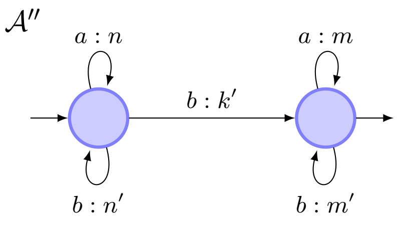


First, suppose that $n \geqslant m$ and $n' \geqslant m'$, then the maximal runs on u and v consist in looping around p, except maybe for the last block. Since u and v have the same last block then $[\![\mathcal{A}]\!](u) = [\![\mathcal{A}]\!](v)$. A similar reasoning holds if $n \leqslant m$ and $n' \leqslant m'$.

We can then suppose now that $n \leqslant m$ and $n' \geqslant m'$ or that $n \geqslant m$ and $n' \leqslant m'$.

First, consider the following automata:





Then for all words w, $[\![\mathcal{A}]\!](w) = \max([\![\mathcal{A}']\!](w), [\![\mathcal{A}'']\!](w))$. It is then sufficient to prove that $[\![\mathcal{A}']\!](u) = [\![\mathcal{A}']\!](v)$ and $[\![\mathcal{A}'']\!](u) = [\![\mathcal{A}'']\!](v)$ to get $[\![\mathcal{A}]\!](u) = [\![\mathcal{A}]\!](v)$ and conclude the proof. We will prove the first one, the other one being similar.

Construct first $\mathcal{B}'$ from $\mathcal{A}'$ by removing m (*resp.* n') from all the weights of the transitions labelled by a (*resp.* b). Then construct $\mathcal{B}$ from $\mathcal{B}'$ by removing $k - m$ from the weight of the transition labelled by a from p to q. The automaton $\mathcal{B}$ belongs to $\mathcal{C}_{\mathcal{T}}{}'$.

We get:

$$\begin{aligned}
[\![\mathcal{A}']\!](u) &= [\![\mathcal{B}']\!](u) + n'|u|_b + m|u|_a && \text{by construction}\\
&= [\![\mathcal{B}]\!](u) + (k-m) + m|u|_a + n'|u|_b && \text{since } p \text{ is initial, } q \text{ final and the transition}\\
& && \text{from } p \text{ to } q \text{ has to be taken exactly once}\\
&= [\![\mathcal{B}]\!](v) + (k-m) + m|v|_a + n'|v|_b && \text{since } u = v \text{ holds in } \mathcal{C}_{\mathcal{T}}{}' \text{ and}\\
& && u \text{ and } v \text{ have the same content}\\
&= [\![\mathcal{B}']\!](v) + m|v|_a + n'|v|_b && \text{since } p \text{ is initial, } q \text{ final and the transition}\\
& && \text{from } p \text{ to } q \text{ has to be taken exactly once}\\
&= [\![\mathcal{A}']\!](v) && \text{by construction}
\end{aligned}$$

□

If an identity holds for the class of all the max-plus automata of the form $\mathcal{A}_{\alpha,\beta}$ and $\mathcal{B}_{\alpha,\beta}$ for all integers α, β either both positive or both negative, the identity is said to be a *triangular identity*.

Checking triangular identities. Checking if a given identity $u = v$ is triangular can be done by symbolic computation using the shape of the automata above. More precisely, for any position i in a word w, we denote by $w_{<i}$ (*resp.* $w_{>i}$) the prefix of w strictly before position i (*resp.* the suffix of w strictly after position i). We get:

$$[\![\mathcal{A}_{\alpha,\beta}]\!](w) = \max_{w_i = a} (\alpha|w_{<i}|_a + \beta|w_{>i}|_b) \quad \text{and} \quad [\![\mathcal{B}_{\alpha,\beta}]\!](w) = \max_{w_i = b} (\alpha|w_{<i}|_a + \beta|w_{>i}|_b)$$

The identity $u = v$ is triangular if and only if for all integers α, β of the same sign, $[\![\mathcal{A}_{\alpha,\beta}]\!](u) = [\![\mathcal{A}_{\alpha,\beta}]\!](v)$ and $[\![\mathcal{B}_{\alpha,\beta}]\!](u) = [\![\mathcal{B}_{\alpha,\beta}]\!](v)$.

An easy way to check them in a reasonable time for identities of small length is to note that the parameters can be bounded:

Lemma 5. *Given two words u and v of the same length ℓ, $[\![\mathcal{A}_{\alpha,\beta}]\!](u) = [\![\mathcal{A}_{\alpha,\beta}]\!](v)$ (resp. $[\![\mathcal{B}_{\alpha,\beta}]\!](u) = [\![\mathcal{B}_{\alpha,\beta}]\!](v)$) holds for all integers α, β either both positive or both negative if and only if it holds for all such α, β with $|\alpha|$, $|\beta|$ bounded by $2\ell^2$.*

Proof. We give the proof for $\mathcal{A}$, the case of $\mathcal{B}$ is similar. Suppose that there exist α, β of the same sign such that $[\![\mathcal{A}_{\alpha,\beta}]\!](u) > [\![\mathcal{A}_{\alpha,\beta}]\!](v)$. Then there is a position i in u with $u_i = a$ such that for all positions j in v with $v_j = a$, we have:

$$\alpha|u_{<i}|_a + \beta|u_{>i}|_b > \alpha|v_{<j}|_a + \beta|v_{>j}|_b$$

or equivalently, $\alpha(|u_{<i}|_a - |v_{<j}|_a) > \beta(|v_{>j}|_b - |u_{>i}|_b)$. Suppose first that $\alpha \geqslant \beta > 0$ (the other cases are similar).

For each j such that $|v_{>j}|_b - |u_{>i}|_b > 0$, we have:

$$\frac{|u_{<i}|_a - |v_{<j}|_a}{|v_{>j}|_b - |u_{>i}|_b} > \frac{\beta}{\alpha} > 0$$

whilst for each j' such that $|v_{>j'}|_b - |u_{>i}|_b < 0$, we have:

$$\frac{|u_{<i}|_a - |v_{<j'}|_a}{|v_{>j'}|_b - |u_{>i}|_b} < \frac{\beta}{\alpha} \leqslant 1$$

If all the positions in v satisfy the first (*resp.* second) condition, then choosing $\alpha = \ell$ and $\beta = 1$ (*resp.* $\alpha = 1$ and $\beta = 1$) still realises the conditions.

Otherwise, there exist $j \neq j'$ as above. Since the right-hand side of the first inequality above is strictly greater than the left-hand side of the second, we obtain:

$$\begin{aligned}&\frac{|u_{<i}|_a - |v_{<j}|_a}{|v_{>j}|_b - |u_{>i}|_b} - \frac{|u_{<i}|_a - |v_{<j'}|_a}{|v_{>j'}|_b - |u_{>i}|_b}\\ =&\frac{(|u_{<i}|_a - |v_{<j}|_a)(|u_{>i}|_b - |v_{>j'}|_b) + (|u_{<i}|_a - |v_{<j'}|_a)(|v_{>j}|_b - |u_{>i}|_b)}{(|v_{>j}|_b - |u_{>i}|_b)(|u_{>i}|_b - |v_{>j'}|_b)} > 0\end{aligned}$$

Notice that by our assumption, the two factors in the denominator are positive integers, from which it follows that the numerator is also a positive integer. Moreover, since each expression of the form $|w_{>j}|$ or $|w_{<j}|$ is bounded by above by the length of the word w, we see that the expression above must be bounded below by $\frac{1}{\ell^2}$.

Thus, for all $j \neq j'$ as above, the difference between $\frac{|u_{<i}|_a - |v_{<j}|_a}{|v_{>j}|_b - |u_{>i}|_b}$ and $\frac{|u_{<i}|_a - |v_{<j'}|_a}{|v_{>j'}|_b - |u_{>i}|_b}$ is at least $\frac{1}{\ell^2}$. Thus the interval $[\max_{j'}\left(\frac{|u_{<i}|_a - |v_{<j'}|_a}{|v_{>j'}|_b - |u_{>i}|_b}\right); \min_j\left(\frac{|u_{<i}|_a - |v_{<j}|_a}{|v_{>j}|_b - |u_{>i}|_b}\right)]$ is non empty and of length at least $\frac{1}{\ell^2}$. Thus there exist a rational with denominator $2\ell^2$ in this interval. It is then possible to find α' and β' with $0 < \beta' \leqslant \alpha' \leqslant 2\ell^2$ such that for each j with $|v_{>j}|_b - |u_{>i}|_b > 0$, we have:

$$\frac{|u_{<i}|_a - |v_{<j}|_a}{|v_{>j}|_b - |u_{>i}|_b} > \frac{\beta'}{\alpha'}$$

whilst for each j' with $|v_{>j'}|_b - |u_{>i}|_b < 0$, we have:

$$\frac{|u_{<i}|_a - |v_{<j'}|_a}{|v_{>j'}|_b - |u_{>i}|_b} < \frac{\beta'}{\alpha'}$$

Finally, for all positions j in v with $v_i = a$,

$$\alpha'|u_{<i}|_a + \beta'|u_{>i}|_b > \alpha'|v_{<j}|_a + \beta'|v_{>j}|_b$$

□

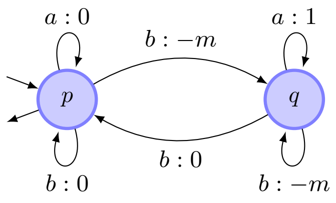
3.2 Block-permutation

Two words u and v are said to be *block-permuted* if u and v are composed of the same maximal blocks of the same consecutive letter but possibly in a different order. For example, $a^3b^2a^4b$ and $b^2a^3ba^4$ are block-permuted but $a^3b^2a^4b$ and a^2baba^4b are not.

Lemma 6. *If $u = v$ is an identity which holds in $\mathcal{C}$, then u and v are block-permuted.*

Proof. Consider an identity $u = v$ which holds in $\mathcal{C}$. Suppose that u and v are not block-permuted, and that the maximal blocks of occurrences of the letter a are different (the proof for the letter b is similar). Let us write $n_1 \geqslant n_2 \geqslant \ldots \geqslant n_\ell$ for the lengths (with multiplicities) of the maximal blocks of consecutive a in u (*resp.* $m_1 \geqslant m_2 \geqslant \ldots \geqslant m_{\ell'}$ for v).

By Lemma 1, u and v have the same content and so there must exist an index $i \in \{1, \ldots, \min(\ell, \ell')\}$ such that $n_j = m_j$ for all $j < i$, whilst $n_i \neq m_i$. Without loss of generality, suppose that $n_i > m_i$ and consider the following automaton where $m = n_i - 1$.



There are four options to read a word of the form ba^k (ignoring initial and final states for the moment): (1) around p with weight 0, (2) from p to q with weight $-m+k$, (3) around q with weight $-m+k$, or (4) from q to p with weight 0. Thus, if a maximal block of a is of length greater than m (except possibly the first or the last one), it should be read around q, otherwise, it should be read around p.

By Lemma 4, u and v must have the same first and last blocks. For each $k = 1, \ldots, \ell$, let $N(k)$ be the set of indices from $1 \leqslant t \leqslant k$ such that a^{n_t} is not the first block of u and v, nor the last block of u and v. It is now easy to see that the weight of u must be greater than or equal to $\sum_{j \in N(i)} (n_j - m)$, while the weight of v is $\sum_{j \in N(i-1)} (n_j - m)$, which is smaller than the weight of u. Since this contradicts the fact that $u = v$ holds in $\mathcal{C}$, we conclude that $n_i = m_i$ for all i; or in other words, u and v are block-permuted. $\square$

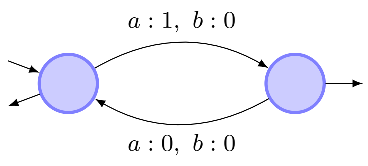
Corollary 1. *If $u = v$ is an identity that holds in $\mathcal{C}$, then u and v each contain at least 7 maximal blocks of the same consecutive letter.*

Proof. Consider an identity $u = v$ with $u = a^{k_1}b^{k_2}a^{k_3}b^{k_4}a^{k_5}b^{k_6}$. If it holds in $\mathcal{C}$, then by Lemma 4, v must start with $a^{k_1}b^{k_2}$ and end with $a^{k_5}b^{k_6}$. Finally, by Lemma 6, necessarily u and v are the same word. $\square$

3.3 Counting and parity conditions

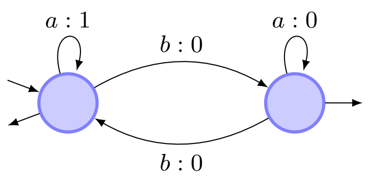
Finally the last conditions we consider involve a finite number of max-plus automata with weights within $\{0, 1\}$ dealing in some sense with parity and counting conditions.

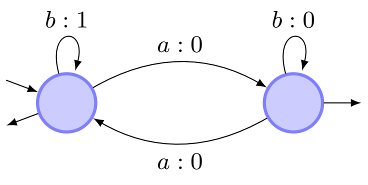
(C1). *The number of occurrences of the letter a in an even position.* This value is computed by the following automaton:



Note that, if two words have the same content, then the equality of this parameter for the two words implies the equality of all the other variants (number of b's in an odd position...). Indeed, the number of a's in an odd position is equal to the difference between the total number of a's and the number of a's in an even position. The number of b's in an even position is the difference between the total number of even positions and the number of a's in an even position.

(C2). *The number of occurrences of the letter a after an even number of b, and the number of occurrences of the letter b after an even number of a.* These values are computed respectively by the following automata:

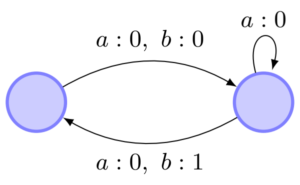


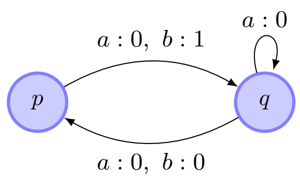


As in the previous condition, providing two words have the same content, the equality on these parameters implies the equality on the other variants (number of a's after an odd number of b's, etc.). Indeed, the number of a's after an odd number of b's is equal to the difference between the total number of a's and the number of a's after an even number of b's...

The two last conditions are more difficult to explain.

(C3). We consider the following automata, and in each case the automata obtained such that exactly one of the states is both initial and final, as well as those obtained by exchanging a and b.





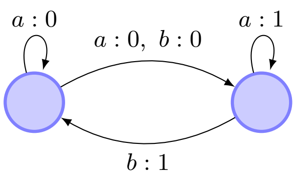
It can be checked that the words $ab^3ababa^3b^3a^2$ and $ab^3a^3babab^3a^2$ cannot be separated by any of the previously discussed conditions. However the automaton on the right, taking p to be both initial and final is able to do so, as we shall now show. The beginning of the two words are read deterministically until reaching the factor a^3. There, a non deterministic choice is made to optimise the weight obtained by reading the end of the word. This choice is made at different positions in the two words leading to two different weights. More precisely, the maximal run for the word $ab^3ababa^3b^3a^2$ is as follows:

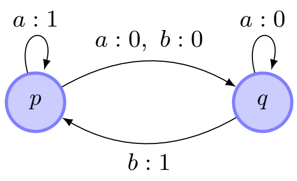
$$p \xrightarrow{ab^3\ :\ 1} p \xrightarrow{ababa\ :\ 0} \underbrace{q \xrightarrow{a^2\ :\ 0} p}_{\text{non det choice}} \xrightarrow{b^3\ :\ 2} q \xrightarrow{a^2\ :\ 0} p$$

while the one for $ab^3a^3babab^3a^2$ is as follows:

$$p \xrightarrow{ab^3 \,:\, 1} \underbrace{p \xrightarrow{a^2 \,:\, 0} q}_{\text{non det choice}} \xrightarrow{ababa \,:\, 2} p \xrightarrow{b^3 \,:\, 2} q \xrightarrow{a^2 \,:\, 0} p$$

(C4). We consider the following automata, and in each case the automata obtained such that exactly one of the states is both initial and final, as well as those obtained by exchanging a and b.





The words $ab^2a^2ba^2ba^4b^3a$ and $ab^2a^4ba^2ba^2b^3a$ cannot be separated by any of the previously discussed conditions, whilst the automaton on the right, taking q to be both initial and final is able to do so. The beginning of the two words are read deterministically until reaching the factor a^2 in the middle of the two words. This determinism forces to read the two first blocks of a with weight 0, while the other ones will be read with weight 1. This leads to different results because of the commutation of the blocks a^2 and a^4 in the two words. More precisely, a maximal run for the word $ab^2a^2ba^2ba^4b^3a$ is as follows:

$$q \xrightarrow{ab^2a^2b \,:\, 2} \underbrace{p \xrightarrow{a^2 \,:\, 1} q}_{\text{non det choice}} \xrightarrow{b \,:\, 1} p \xrightarrow{a^4 \,:\, 4} p \xrightarrow{b^3a \,:\, 1} q$$

while the one for $ab^2a^4ba^2ba^2b^3a$ is as follows:

$$q \xrightarrow{ab^2a^4b \,:\, 2} \underbrace{p \xrightarrow{a^2 \,:\, 1} q}_{\text{non det choice}} \xrightarrow{b \,:\, 1} p \xrightarrow{a^2 \,:\, 2} p \xrightarrow{b^3a \,:\, 1} q$$

An identity $u = v$ is said to satisfy **(C1)**, **(C2)**, **(C3)** or **(C4)** if the same values is computed on u and v by the automata given above.

Proposition 3. *There are exactly four triangular identities $u = v$ of length shorter than* 18 *satisfying* ***(C1)****,* ***(C2)****,* ***(C3)*** *and* ***(C4)*** *in which u and v are block-permuted and have the same first and last blocks.*

We can check all these conditions assisted by a computer. The programs are given in Section A of the appendix and list all the identities not eliminated by one of these conditions.

Moreover, this list of conditions is in some sense minimal since for each of them, there are examples of identities that are not eliminated when removing the condition from the list. These examples are exhibited by the programs and given in the appendix.

We remark that if the block-permutation condition holds then the only automata we need to consider which involve weights not within $\{0, 1\}$ are the ones corresponding to the triangular conditions. This list of conditions is probably not sufficient to characterise fully the identities which hold in $\mathcal{C}$, however, one can ask if we can extend it and keep this distinction between the triangular conditions with arbitrary weights and the other conditions involving only weights in $\{0, 1\}$.

There are exactly four remaining candidates:

$$a^2b^3a^3babab^3a^2 = a^2b^3ababa^3b^3a^2, \quad ab^3a^4baba^2b^3a = ab^3a^2baba^4b^3a$$

and the ones obtained by exchanging the roles of a and b. In the next section, we prove that they indeed hold in $\mathcal{C}$.

4 The shortest identities

In this section, we conclude the proof of Theorem 1 by proving that the remaining candidate identities hold in $\mathcal{C}$. By exchanging the role of a and b, it is sufficient to prove the following proposition:

Proposition 4. *The following two identities hold in $\mathcal{C}$:*

$$\textbf{\textit{(I1)}}\ a^2b^3a^3babab^3a^2 = a^2b^3ababa^3b^3a^2 \qquad \textit{and} \qquad \textbf{\textit{(I2)}}\ ab^3a^4baba^2b^3a = ab^3a^2baba^4b^3a$$

For a word $u = a_0 \cdots a_\ell$ of length $\ell + 1$, let us denote by $\tilde{u} = a_\ell \cdots a_0$ the *reverse* of u.

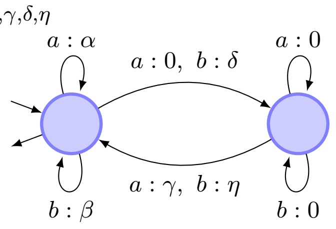
Lemma 7. *Let $u \in \Sigma^+$. If $[\![\mathcal{A}]\!](u) \geqslant [\![\mathcal{A}]\!](\tilde{u})$ for all $\mathcal{A}$ in $\mathcal{C}$, then $u = \tilde{u}$ is an identity which holds in $\mathcal{C}$.*

Proof. Consider an automaton $\mathcal{A}$ in $\mathcal{C}$. By hypothesis, $[\![\mathcal{A}]\!](u) \geqslant [\![\mathcal{A}]\!](\tilde{u})$. Construct now $\mathcal{B}$ obtained from $\mathcal{A}$ by reversing the transitions, *i.e.* there is a transition $p \xrightarrow{c\,:\,\alpha} q$ in $\mathcal{A}$ if and only if there is a transition $q \xrightarrow{c\,:\,\alpha} p$ in $\mathcal{B}$. Moreover the initial (*resp.* final) states of $\mathcal{B}$ are defined from the final (*resp.* initial) states of $\mathcal{A}$. By this construction, $[\![\mathcal{A}]\!](\tilde{u}) = [\![\mathcal{B}]\!](u) \geqslant [\![\mathcal{B}]\!](\tilde{u}) = [\![\mathcal{A}]\!](u)$. Thus $[\![\mathcal{A}]\!](\tilde{u}) = [\![\mathcal{A}]\!](u)$ for all $\mathcal{A}$ in $\mathcal{C}$ and hence $u = \tilde{u}$ holds in $\mathcal{C}$. ☐

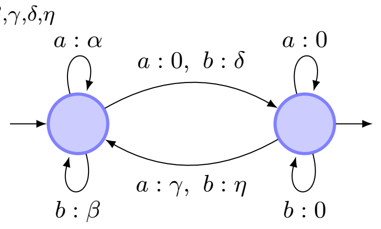
Remark that the two identities **(I1)** and **(I2)** are of the form $u = \tilde{u}$.

Lemma 8. *Given two words u and v of the same content, $[\![\mathcal{A}]\!](u) \geqslant [\![\mathcal{A}]\!](v)$ for all $\mathcal{A}$ in $\mathcal{C}$ if and only if $[\![\mathcal{B}]\!](u) \geqslant [\![\mathcal{B}]\!](v)$ for all $\mathcal{B}$ of one of the following two forms, where α, β, γ, δ, η are integers:*

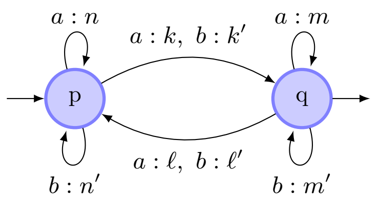
$\mathcal{A}_{\alpha,\beta,\gamma,\delta,\eta}$



$\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}$



Proof. The if direction is clear by definition. Conversely, denote by $\mathcal{C}'$ the class of automata described in the statement of the proposition. Suppose that $[\![\mathcal{B}]\!](u) \geqslant [\![\mathcal{B}]\!](v)$ for all $\mathcal{B} \in \mathcal{C}'$. Consider $\mathcal{A} \in \mathcal{C}$. First, by the proof of Lemma 3, we can suppose that $\mathcal{A}$ is full and by Lemma 2, that $\mathcal{A}$ has exactly one initial and one final state. Suppose that these two states are different. If not, a similar reasoning will hold, involving $\mathcal{A}_{\alpha,\beta,\gamma,\delta,\eta}$ instead of $\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}$. We represent $\mathcal{A}$ in the following picture:



First, construct $\mathcal{A}'$ from $\mathcal{A}$ by removing m (*resp.* m') from all the weights of the transitions labelled by a (*resp.* b). Then construct $\mathcal{B}$ from $\mathcal{A}'$ by removing $k - m$ from the weights of the transitions labelled by a and b from p to q and adding $k - m$ from the weights of the transitions labelled by a and b from q to p. By construction, $\mathcal{B}$ is in $\mathcal{C}'$. We get:

$$\begin{aligned}
[\![\mathcal{A}]\!](u) &= [\![\mathcal{A}']\!](u) + m'|u|_b + m|u|_a && \text{by construction}\\
&= [\![\mathcal{B}]\!](u) + (k-m) + m'|u|_b + m|u|_a && \text{since } p \text{ is initial and } q \text{ final, thus on an}\\
& && \text{accepting run, the transitions from } p \text{ to } q\\
& && \text{are taken (in total) exactly once more}\\
& && \text{than the transitions from } q \text{ to } p\\
&\geqslant [\![\mathcal{B}]\!](v) + (k-m) + m'|v|_b + m|v|_a && \text{since } \mathcal{B} \text{ is in } \mathcal{C}' \text{ and}\\
& && u \text{ and } v \text{ have the same content}\\
&\geqslant [\![\mathcal{A}']\!](v) + m'|v|_b + m|v|_a && \text{for the same reason as above}\\
&\geqslant [\![\mathcal{A}]\!](v) && \text{by construction}
\end{aligned}$$

□

Let us consider **(I1)** and denote $u = a^2b^3a^3babab^3a^2$. By Lemmas 7 and 8, for proving that **(I1)** holds in $\mathcal{C}$, it is sufficient to prove that for all integers $\alpha, \beta, \gamma, \delta, \eta$, $[\![\mathcal{A}_{\alpha,\beta,\gamma,\delta,\eta}]\!](u) \geqslant [\![\mathcal{A}_{\alpha,\beta,\gamma,\delta,\eta}]\!](\tilde{u})$ and $[\![\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}]\!](u) \geqslant [\![\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}]\!](\tilde{u})$. Consider $\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}$. Aided by computer, we compute symbolically the values on u and on its reverse in $\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}$ as a tropical polynomial in α, β, γ, δ, η. Each monomial term corresponds to the weight of an accepting (but not necessarily maximal weight) run. For example, the monomial term $8\alpha + 8\beta$ corresponds to an accepting run on u (when reading u around the initial state and going to the final state on the last transition) and in fact on $\tilde{u}$ also. We denote by M_u, $M_{\tilde{u}}$ and M the set of monomials appearing only in the computation of u, only in the computation of $\tilde{u}$ or for both, respectively. We compute these three sets aided by a computer.

Finally, we prove that for each monomial in $M_{\tilde{u}}$ and each choice of parameters $\alpha, \beta, \gamma, \delta, \eta \in \mathbb{Z}$, there is a monomial in $M_u \cup M$ which is greater on the values $\alpha, \beta, \gamma, \delta, \eta$. This concludes the proof that for all integers $\alpha, \beta, \gamma, \delta, \eta$, $[\![\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}]\!](u) \geqslant [\![\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}]\!](\tilde{u})$. To do so, there is no need to consider the monomials in $M_{\tilde{u}}$ in which neither γ nor η appears. Indeed, we already checked in the previous section that **(I1)** satisfies the triangular conditions. Thus, $u = \tilde{u}$ holds for triangular automata. Consider $\mathcal{B}'_{\alpha,\beta,\delta}$ constructed from $\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}$ by removing the transitions from the final state to the initial state. A monomial in $M_{\tilde{u}}$ in which neither γ nor η appears corresponds to an accepting run in $\mathcal{B}'_{\alpha,\beta,\delta}$, and hence is bounded above by $\mathcal{B}'_{\alpha,\beta,\delta}(\tilde{u}) = \mathcal{B}'_{\alpha,\beta,\delta}(u)$, since this automaton is triangular. The latter is clearly bounded above by $\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}(u)$. So for all monomials in $M_{\tilde{u}}$ in which neither γ nor η and each choice of parameters $\alpha, \beta, \gamma, \delta, \eta \in \mathbb{Z}$, there is necessarily a monomial in $M_u \cup M$ which is greater on the values $\alpha, \beta, \gamma, \delta, \eta$. Finally, the set $M_{\tilde{u}}$ without these monomials is of reasonable size and we are able to complete the computations by hand.

Similar computations hold for $\mathcal{A}_{\alpha,\beta,\gamma,\delta,\eta}$ and for **(I2)**. They are all provided in Section B of the appendix.

5 Conclusion

In this paper, we give the shortest identities which hold in the class of max-plus automata with two states. We hope that a better understanding of this case is a first step towards a better understanding of the general case. In particular, we give an interesting list of conditions which are sufficient to achieve this goal. Future works will consist in trying to understand better these conditions and how to extend this list to fully characterise the sets of identities for max-plus automata with two states.

References

[1] Y. Chen, X. Hu, Y. Luo, and O. Sapir. The finite basis problem for the monoid of two-by-two upper triangular tropical matrices. *Bull. Aust. Math. Soc.*, 94(1):54–64, 2016.

[2] T. Colcombet, L. Daviaud, and F. Zuleger. Size-change abstraction and max-plus automata. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 2014.

[3] L. Daviaud, M. Johnson, and M. Kambites. Identities in upper triangular tropical matrix semigroups and the bicyclic monoid. 2017. preprint, http://arxiv.org/abs/1612.04219.

[4] S. Gaubert. Performance evaluation of $(\max, +)$ automata. *IEEE Trans. Automat. Control*, 40(12):2014–2025, 1995.

[5] S. Gaubert and J. Mairesse. Modeling and analysis of timed Petri nets using heaps of pieces. *IEEE Trans. Automat. Control*, 44(4):683–697, 1999.

[6] Z. Izhakian. Semigroup identities in the monoid of triangular tropical matrices. *Semigroup Forum*, 88(1):145–161, 2014.

[7] Z. Izhakian. Erratum to: Semigroup identities in the monoid of triangular tropical matrices [MR3164156]. *Semigroup Forum*, 92(3):733, 2016.

[8] Z. Izhakian. Semigroup identities of tropical matrix semigroups of maximal rank. *Semigroup Forum*, 92(3):712–732, 2016.

[9] Z. Izhakian and S. W. Margolis. Semigroup identities in the monoid of two-by-two tropical matrices. *Semigroup Forum*, 80(2):191–218, 2010.

[10] J. Okniński. Identities of the semigroup of upper triangular tropical matrices. *Comm. Algebra*, 43(10):4422–4426, 2015.

[11] J. Sakarovitch. *Elements of Automata Theory.* Cambridge University Press, 2009.

[12] M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.

[13] Y. Shitov. A semigroup identity for tropical 3×3 matrices. 2014. To appear in Ars Mathematica Contemporanea 14 (2018), 15–23.

A Eliminating identities

We prove here Proposition 3. We give a program which lists all the pairs of words u, v which are block-permuted and have the same first and last blocks, and which satisfy conditions **(C1)**, **(C2)**, **(C3)** and **(C4)** and the triangular conditions.

A word is encoded in an array such that the content at index j is the length of the j-th maximal block of the same consecutive letter. For exemple $a^3b^2a^5b^4a^2$ is encoded in the array `[|3;2;5;4;2|]`. Given a word, we consider all the words that are block-permuted and have the same first and last blocks, that is to say all the arrays having the same content at index 0 and at the last index, and such that the contents of the even indices (*resp.* odd indices) are permuted. We then test the conditions **(C1)**, **(C2)**, **(C3)** and **(C4)** and the triangular conditions. We list all the pairs satisfying these requirements up to length 17. We use the matrix representation of automata. For example the automaton $\mathcal{A}_{\alpha,\beta}$ regarding the triangular condition is encoded into the matrix `[|`α`;0;i;0|]` for the letter a and the matrix `[|0;i;i;`β`|]` for the letter b, where `i` encodes $-\infty$. We test the triangular condition by computing the product obtained by substituting the letters of a word (encoded by an array) by the matrices `[|`α`;0;i;0|]` and `[|0;i;i;`β`|]`. We could perform the computation for α, β up to the bound given in Lemma 5 (*i.e.* 578). However, since we do not know if our conditions are sufficient, we note that the main aim of the program is merely to rule out possibilities; experimentally, we found that we could rule out a large number of possibilities using values bounded by 10. We also directly compute the product of the matrices for **(C3)** and **(C4)**, while **(C1)** and **(C2)** are checked by counting.

For each of the conditions ($\star$), we give here a pair of words satisfying all the above mentionned conditions except ($\star$).

Triangular condition: $aba^3b^2aba = abab^2a^3ba$

(C1): $abab^2ababa^2b^2aba = abab^2a^2babab^2aba$

(C2): removing the automaton on the left: $ab^3a^2ba^4b^3a = ab^3a^4ba^2b^3a$

removing the automaton on the right: $aba^3b^2ab^4a^3b = aba^3b^4ab^2a^3b$

(C3): removing the automaton on the left:

$$a^2b^3ababa^3b^3a = a^2b^3a^3babab^3a$$

its symetric when exchanging the role of a and b:

$$ab^2a^3babab^3a^3b = ab^2a^3b^3ababa^3b$$

removing the automaton on the right:

$$ab^3ababa^3b^3a^2 = ab^3a^3babab^3a^2$$

its symetric when exchanging the role of a and b:

$$aba^3babab^3a^3b^2 = aba^3b^3ababa^3b^2$$

(C4): removing the automaton on the left:

$$ab^3a^2ba^2ba^4b^2a = ab^3a^4ba^2ba^2b^2a$$

its symetric when exchanging the role of a and b:

$$abab^2aba^2bab^2a^2ba = abab^2a^2bab^2aba^2ba$$

removing the automaton on the right:

$$ab^2a^2ba^2ba^4b^3a = ab^2a^4ba^2ba^2b^3a$$

its symetric when exchanging the role of a and b:

$$aba^2b^2aba^2bab^2aba = aba^2bab^2aba^2b^2aba$$

```
(* Hypothesis:
all the vectors have length at least 7,
and contains only positive integers
i plays the role of minus infinity,
and is replaced by a very negative integer *)


(********** TOOLS **********)

let sum a b i =
  if (a = i || b = i)
then i else (a + b);;

let maximum a b i =
  if (a = i && b = i) then i else max a b;;

(*product of two matrices*)
let product_matrix m n i =
  match m,n with
  |[|a;b;c;d|], [|e;f;g;h|] -> [|maximum (sum a e i) (sum b g i) i;
                                 maximum (sum a f i) (sum b h i) i;
                                 maximum (sum c e i) (sum d g i) i;
                                 maximum (sum c f i) (sum d h i) i|];
  | _ -> [|0; 0; 0; 0|];;

(*exponentiation of a matrix*)
let rec exponentiation_matrix m k i =
  if (k = 1) then m
  else
    let n = exponentiation_matrix m (k/2) i in
      if (k mod 2 = 0) then product_matrix n n i
      else product_matrix (product_matrix n n i) m i;;

(*computing the product by substituting letters by matrices in the array t*)
let product t m n i =
  let l = (Array.length t) in
  let p = ref (exponentiation_matrix m t.(0) i) in
    for j = 1 to l-1 do
      if (j mod 2 = 1) then
        p := product_matrix (!p) (exponentiation_matrix n (t.(j)) i) i
      else
        p := product_matrix (!p) (exponentiation_matrix m (t.(j)) i) i;
    done;
  !p;;

(*product of triangular matrices*)
let triangular_product_matrix m n i =
  match m,n with
  |[|a;b;c;d|], [|e;f;g;h|] -> [|sum a e i;
                                 maximum (sum a f i) (sum b h i) i;
                                 i;
                                 sum d h i|];
  | _ -> [|0; 0; 0; 0|];;
```

```
(*exponentiation of triangular matrices*)
let rec triangular_exponentiation_matrix m k i =
  if (k = 1) then m
  else
    let n = triangular_exponentiation_matrix m (k/2) i in
      if (k mod 2 = 0) then triangular_product_matrix n n i
      else triangular_product_matrix (triangular_product_matrix n n i) m i;;

(********** TESTING CONDITIONS **********)

(**** TRIANGULAR ****)

(*function computed by the triangular automaton A  with
  parameters a, b if the word starts with a *)
let triangular_productA_a t a b i =
  let l = (Array.length t) in
  let m = ref [|0; i; i; 0|] in
    for j = 0 to (l/2 - 1) do
      let n1 = ref (triangular_exponentiation_matrix [|a;0;i;0|] (t.(2*j)) i) in
      let n2 = ref (triangular_exponentiation_matrix [|0;i;i;b|] (t.(2*j+1)) i) in
        m := triangular_product_matrix (!m) (!n1) i;
        m := triangular_product_matrix (!m) (!n2) i;
    done;
    if (l mod 2 = 1) then
      m := triangular_product_matrix !m
           (triangular_exponentiation_matrix [|a;0;i;0|] (t.(l-1)) i)
           i;
  !m.(1);;

(*function computed by the triangular automaton A with
  parameters a, b if the word starts with b*)
let triangular_productA_b t a b i =
  let l = (Array.length t) in
  let m = ref [|0; i; i; 0|] in
    for j = 0 to (l/2 - 1) do
      let n1 = ref (triangular_exponentiation_matrix [|0;i;i;b|] (t.(2*j)) i) in
      let n2 = ref (triangular_exponentiation_matrix [|a;0;i;0|] (t.(2*j+1)) i) in
        m := triangular_product_matrix !m !n1 i;
        m := triangular_product_matrix !m !n2 i;
    done;
    if (l mod 2 = 1) then
      m := triangular_product_matrix !m
           (triangular_exponentiation_matrix [|0;i;i;b|] (t.(l-1)) i)
           i;
  !m.(1);;

(*function computed by the triangular automaton B  with
  parameters a, b if the word starts with a *)
```

```
let triangular_productB_a t a b i =
  let l = (Array.length t) in
  let m = ref [|0; i; i; 0|] in
    for j = 0 to (l/2 - 1) do
      let n1 = ref (triangular_exponentiation_matrix [|a;i;i;0|] (t.(2*j)) i) in
      let n2 = ref (triangular_exponentiation_matrix [|0;0;i;b|] (t.(2*j+1)) i) in
        m := triangular_product_matrix (!m) (!n1) i;
        m := triangular_product_matrix (!m) (!n2) i;
    done;
    if (l mod 2 = 1) then
      m := triangular_product_matrix !m
           (triangular_exponentiation_matrix [|a;i;i;0|] (t.(l-1)) i)
           i;
  !m.(1);;

(*function computed by the triangular automaton A with
  parameters a, b if the word starts with b*)
let triangular_productB_b t a b i =
  let l = (Array.length t) in
  let m = ref [|0; i; i; 0|] in
    for j = 0 to (l/2 - 1) do
      let n1 = ref (triangular_exponentiation_matrix [|0;0;i;b|] (t.(2*j)) i) in
      let n2 = ref (triangular_exponentiation_matrix [|a;i;i;0|] (t.(2*j+1)) i) in
        m := triangular_product_matrix !m !n1 i;
        m := triangular_product_matrix !m !n2 i;
    done;
    if (l mod 2 = 1) then
      m := triangular_product_matrix !m
           (triangular_exponentiation_matrix [|0;0;i;b|] (t.(l-1)) i)
           i;
  !m.(1);;

(*true iff t1 and t2 satisfies the triangular
  condition for parameters a and b*)
let triangular_equal t1 t2 a b i =
  (triangular_productA_a t1 a b i = triangular_productA_a t2 a b i)
  && (triangular_productA_b t1 a b i = triangular_productA_b t2 a b i)
  && (triangular_productB_a t1 a b i = triangular_productB_a t2 a b i)
  && (triangular_productB_b t1 a b i = triangular_productB_b t2 a b i);;

(*triangular condition up to bound d for the parameters*)
let triangular t1 t2 d i =
  let x = ref true in
  let a = ref 0 in
  let b = ref 0 in
    while ((!a <= d) && !x) do
      while ((!b <= d) && !x) do
        x := triangular_equal t1 t2 (!a) (!b) i;
        b := !b + 1;
      done;
      b := 0;
      a := !a + 1;
    done;
  a := 0;
  b := 0;
    while ((!a >= (-d)) && !x) do
      while ((!b >= (-d)) && !x) do
        x := triangular_equal t1 t2 (!a) (!b) i;
        b := !b - 1;
      done;
      b := 0;
      a := !a - 1;
    done;
  !x;;

(**** C1 ****)

(*count the number of the letter which starts the word
  occuring in even positions in the word*)
let even_in_even t =
  let l = Array.length t in
  let s = ref 0 in
  let p = ref 0 in
  let j = ref 0 in
    while (!j < l) do
      if (!p = 0 && t.(!j) mod 2 = 1) then
         s := !s + (t.(!j)/2) + 1
      else
         s := !s + (t.(!j)/2);
      if (!j < l-1) then
         p := (!p + (t.(!j) mod 2) + (t.(!j+1) mod 2)) mod 2;
      j := !j + 2;
    done;
  !s;;

(*condition c1*)
let c1 t1 t2 = (even_in_even t1 = even_in_even t2);;

(**** C2 ****)

(*count the number of the letter which starts the word
  occuring after an even number of the other letter*)
let even_after_even t =
  let l = Array.length t in
  let s = ref 0 in
  let p = ref 0 in
  let j = ref 0 in
    while (!j < l) do
```

```
      if !p = 0 then
        s := !s + t.(!j);
      if (!j < l-1) then
p := (!p + (t.(!j+1) mod 2)) mod 2;
      j := !j + 2;
    done;
  !s;;

(*count the number of the letter which does not start the
  word occuring after an even number of the other letter*)
let odd_after_even t =
  let l = Array.length t in
  let s = ref 0 in
  let p = ref 0 in
  let j = ref 0 in
    while (!j<l) do
      p := (!p + (t.(!j) mod 2)) mod 2;
      if ((!p = 0) && (!j < l-1))then
        s := !s + t.(!j+1);
      j := !j+2;
    done;
  !s;;

(*condition c2*)
let c2 t1 t2 = (odd_after_even t1 = odd_after_even t2)
&&  (even_after_even t1 = even_after_even t2);;

(**** C3 and C4 ****)

(*compute the matrices associated with the automata of conditions C3 and C4*)
let automaton1 t i =
  product t [|i;0;0;0|] [|i;0;1;i|] i;;

let automaton1sym t i =
  product t [|i;0;1;i|] [|i;0;0;0|] i;;

let automaton2 t i =
  product t [|i;0;0;0|] [|i;1;0;i|] i;;

let automaton2sym t i =
  product t [|i;1;0;i|] [|i;0;0;0|] i;;

let automaton3 t i =
  product t [|0;0;i;1|] [|i;0;1;i|] i;;

let automaton3sym t i =
  product t [|i;0;1;i|] [|0;0;i;1|] i;;

let automaton4 t i =
  product t [|1;0;i;0|] [|i;0;1;i|] i;;

let automaton4sym t i =
  product t [|i;0;1;i|] [|1;0;i;0|] i;;

(*selection of the state both initial and final*)
let diag_equal m n =
  (m.(0) = n.(0)) && (m.(3) = n.(3));;

(*conditions c3 and c4*)
let c3 t1 t2 i =
  diag_equal (automaton1 t1 i) (automaton1 t2 i)
  && diag_equal (automaton1sym t1 i) (automaton1sym t2 i)
  && diag_equal (automaton2 t1 i) (automaton2 t2 i)
  && diag_equal (automaton2sym t1 i) (automaton2sym t2 i)
  && diag_equal (automaton3 t1 i) (automaton3 t2 i)
  && diag_equal (automaton3sym t1 i) (automaton3sym t2 i)
  && diag_equal (automaton4 t1 i) (automaton4 t2 i)
  && diag_equal (automaton4sym t1 i) (automaton4sym t2 i);;

(********** BLOCK PERMUTATION AND FIRST AND LAST BLOCKS **********)

(*concatenation of two lists*)
let rec concat f1 f2 =
  match f1 with
  |[] -> f2;
  |h::q -> h::(concat q f2);;

(*remove the content at index k in v*)
let remove_index v k =
  let l = Array.length v in
  let t = Array.make (l-1) 0 in
    for j = 0 to k - 1 do
      t.(j) <- v.(j);
    done;
    for s = k to l - 2 do
      t.(s) <- v.(s+1);
    done;
  t;;

(*lists all the arrays obtained by permutating the contents
  in t at the indices contained in v*)
let rec permutation_indices t v =
  let l = Array.length v in
    match l with
    |0 -> [];
    |1 -> [t];
    |_ -> begin
          let f = ref [] in
          let s = Array.length t in
            for j = 0 to l - 1 do
              let u = Array.make s 0 in
                for k = 0 to s - 1 do
              u.(k) <- t.(k);
                done;
                u.(v.(0)) <- t.(v.(j));
                u.(v.(j)) <- t.(v.(0));
                f := concat (permutation_indices u (remove_index v j)) !f;
            done;
          !f;
          end;;

(*gives an array containing the odd indices of t,
  except the first and the last or second-last*)
let odd_indices t =
  let l = Array.length t in
  let v = Array.make ((l-4)/2) 0 in
    for j = 0 to ((l-4)/2) - 1 do
      v.(j) <- 2*j + 3;
    done;
  v;;

(*gives an array containing the even indices of t,
  except the zero and the last or second-last*)
let even_indices t =
  let l = Array.length t in
    if l mod 2 = 0 then
      begin
        let v = Array.make ((l-4)/2) 0 in
          for j = 0 to ((l-4)/2) - 1 do
            v.(j) <- 2*j + 2;
          done;
  v;
      end
    else
      begin
        let v = Array.make (((l-4)/2) + 1) 0 in
          for j = 0 to ((l-4)/2) do
            v.(j) <- 2*j + 2;
          done;
        v;
      end;;

(*checking if two arrays are different*)
let different_array t1 t2 =
  let b = ref false in
  let l = Array.length t1 in
  let j = ref 0 in
```

```
    if l <> Array.length t2 then
      b:= true
    else
      while (!j < l) && not(!b) do
        if (t1.(!j) <> t2.(!j)) then
          b:= true;
        j := !j + 1;
      done;
  !b;;

(*remove the occurences of t in the list f*)
let rec clean_list t f =
  match f with
  |[] -> [];
  |h::q -> if different_array t h then
             h::(clean_list t q)
           else
             clean_list t q;;

(*erase copies in a list*)
let rec remove_twins f =
  match f with
  |[] -> [];
  |h::q -> h::(remove_twins (clean_list h q));;

(*gives a list without copy of the permutation of t
  at indices in v*)
let permutation_clean t v =
  remove_twins (permutation_indices t v);;

(*permutations of the indices in v  of all the element of a list*)
let rec permutation_list f v =
  match f with
  |[] -> [];
  |h::q -> remove_twins (concat (permutation_clean h v) (permutation_list q v));;

(*list of the block-permutation of t*)
let permutation t =
  let v1 = odd_indices t in
  let v2 = even_indices t in
    permutation_list (permutation_clean t v1) v2;;

(********** LISTS THE POSSIBLE IDENTITIES **********)

(*check if an identity satisfies c1, c2, c3 and the triangular conditions*)
let identity t1 t2 d i =
   (c1 t1 t2) && (c2 t1 t2) && (c3 t1 t2 i) && (triangular t1 t2 d i);;

(*lists the possible identities from t and a list*)
```

```
let rec identity_with_a_list t f d i =
  match f with
  |[] -> [];
  |h::q -> if (identity t h d i) then
     (t,h)::(identity_with_a_list t q d i)
   else
     identity_with_a_list t q d i;;

(*lists the possible identities contained in a list*)
let rec identity_list f d i =
  match f with
  |[] -> [];
  |h::q -> concat (identity_with_a_list h q d i) (identity_list q d i);;

(*lists the possible identities composed with two permutations of t*)
let identities t d i =
  identity_list (permutation t) d i;;

(*lists the possible identities composed with two permutations
  of all the elements of a list*)
let rec identities_list f d i =
  match f with
  |[] -> [];
  |h::q -> concat (identities h d i) (identities_list q d i);;

(*print tools*)
let print_array t =
  for j = 0 to Array.length t - 1 do
    print_int t.(j); print_string ";";
  done;;

let rec print_list f =
  match f with
  |[] -> print_string "";
  |(t1,t2)::q -> print_array t1; print_string "=";
                     print_array t2; print_newline(); print_list q;;

let rec concat_all j f =
  match f with
  |[] -> [];
  |h::q -> (j::h)::(concat_all j q);;

(*gives all the lists of l decreasing positive integers of sum k*)
let rec decreas_tool l k z =
  match l with
  |1 -> if ((k <= z) && (k <> 0)) then [[k]] else [];
  |_ -> let f = ref [] in
        for j = 1 to (min k z) do
          f := concat (concat_all j (decreas_tool (l-1) (k-j) j)) (!f);
          done;
        !f;;

let decreas l k = decreas_tool l k (k-l+1);;

(*copy*)
let copy t =
  let l = Array.length t in
  let u = Array.make l 0 in
    for j = 0 to l-1 do
      u.(j) <- t.(j);
    done;
  u;;

(*fill the every other indices of an aray from j according to a list of contents
  under good assumptions*)
let rec fill_rec t f j =
  match f with
  |[] -> let u = copy t in u;
  |h::q -> begin
             let u = fill_rec t q (j+2) in
               u.(j) <- h;
               u;
           end;;

(*fill the even indices of an aray according to a list of contents
  under good assumptions*)
let fill_even t f =
  fill_rec t f 2;;

(*fill the odd indices of an aray according to a list of contents
  under good assumptions*)
let fill_odd t f =
  fill_rec t f 3;;

(*constructs a list of t knowing the first, second, last, second-last blocks
  and the lists above*)
let rec representant_tool t j0 j1 jsl jl f feven =
  match feven with
  |[] -> [];
  |h::q -> begin
            let l = Array.length t in
             t.(0) <- j0;
             t.(1) <- j1;
             t.(l-2) <- jsl;
             t.(l-1) <- jl;
             let u = fill_even (fill_odd t f) h in
               u::(representant_tool t j0 j1 jsl jl f q);
           end;;
```

```
let rec representant t j0 j1 jsl jl fodd feven =
  match fodd with
  |[] -> [];
  |h::q -> concat (representant_tool t j0 j1 jsl jl h feven)
              (representant t j0 j1 jsl jl q feven);;

(*print the possible identities with l >= 7 blocks and  k >= 7 letters*)
let all_the_identities l k d i =
    for j0 = 1 to (k - (l-1)) do
      for j1 = 1 to (k - (l-2-j0)) do
        for jsl = 1 to (k - (l-3-j0-j1)) do
          for jl = 1 to (k - (l-4-j0-j1-jsl)) do
            if (l mod 2 = 0) then
              for odd = ((l-4)/2) to (k - ((l-4)/2) - j0 - j1 - jsl - jl) do
                let even = (k - odd - j0 - j1 - jsl - jl) in
                let fodd = decreas ((l-4)/2) odd in
                let feven = decreas ((l-4)/2) even in
                let t = Array.make l 1 in
                let f = representant t j0 j1 jsl jl fodd feven in
                  print_list (identities_list f d i);
               done
            else
              for odd = ((l-4)/2) to (k - ((l-4)/2 + 1) - j0 - j1 - jsl - jl) do
                let even = (k - odd - j0 - j1 - jsl - jl) in
                let fodd = decreas ((l-4)/2) odd in
                let feven = decreas ((l-4)/2 + 1) even in
                let t = Array.make l 1 in
                let f = representant t j0 j1 jsl jl fodd feven in
                  print_list (identities_list f d i);
               done;
          done;
        done;
      done;
    done;;


for l = 7 to 16 do
  for k = l+1 to 17 do
    print_int k;
    print_string " letters and ";
    print_int l;
    print_string " blocks ";
    print_newline();
    all_the_identities l k 10 (-1000);
  done;
done;;
```

B The minimal identities

We prove here Proposition 4. We write $u = a^2b^3a^3babab^3a^2$ and $v = ab^3a^4baba^2b^3a$ so that **(I1)** and **(I2)** have the form $u = \tilde{u}$ and $v = \tilde{v}$ respectively. We write $M_{\tilde{u}}^{\mathcal{A}}$ the set of the monomials arising from $\mathcal{A}_{\alpha,\beta,\gamma,\delta,\eta}$ for $\tilde{u}$ only (similar notations are used for $\tilde{v}$ and $\mathcal{B}_{\alpha,\beta,\gamma,\delta,\eta}$).

Using a formal calculation software, we obtain:

(I1)

$M_{\tilde{u}}^{\mathcal{A}}$ is the set of the following monomials:

- $\alpha + 5\gamma + 5\delta + 3\eta$
- $3\alpha + 3\beta + \gamma + \delta$

$M_{\tilde{u}}^{\mathcal{B}}$ is the set of the following monomials:

- $5\alpha + 4\delta + 3\eta$
- $5\alpha + 5\beta$
- $5\alpha + 2\beta + 2\delta + \eta$
- $4\alpha + 5\beta$
- $4\alpha + 4\beta + \delta$
- $3\alpha + 4\beta$
- $3\alpha + 3\beta + \delta$
- $\alpha + 3\beta + 3\gamma$

(I2)

$M_{\tilde{v}}^{\mathcal{A}}$ is the set of the following monomials:

- $5\alpha + 4\beta + \gamma + \delta$
- $4\alpha + 3\beta + \gamma + \delta$

$M_{\tilde{v}}^{\mathcal{B}}$ is the set of the following monomials

- $6\alpha + \beta + 4\delta + 3\eta$
- $5\alpha + 4\delta + 4\eta$
- $6\alpha + 3\beta + 2\delta + \eta$
- $5\alpha + 4\delta + 3\eta$
- $5\alpha + 5\beta$
- $5\alpha + 2\beta + 2\delta + \eta$
- $4\alpha + 5\beta$
- $4\alpha + 4\beta + \delta$
- $3\alpha + 4\beta$
- $3\alpha + 3\beta + \delta$

Using the part of the program concerning triangular conditions, given in the previous section and the bound in Lemma 5, we have checked that identities **(I1)** and **(I2)** are triangular identities. So, as explained in the body of the paper, there is no need to consider the monomials that do not contain any γ and η. For the remaining ones, it remains to prove that there are monomials for u and v which are greater or equal.

(I1)

$M_{\tilde{u}}^{\mathcal{A}}$:

- $\alpha + 5\gamma + 5\delta + 3\eta$

  Monomials for u in $\mathcal{A}$:

  $\alpha + 2\gamma + 2\delta + 6\eta$ greater if $-\gamma - \delta + \eta \geqslant 0$

  $\alpha + 6\gamma + 6\delta + 2\eta$ greater if $\gamma + \delta - \eta \geqslant 0$

- $3\alpha + 3\beta + \gamma + \delta$

  Monomials for u in $\mathcal{A}$:

  $2\alpha + 2\beta + \gamma + \delta$ greater if $-\alpha - \beta \geqslant 0$

$4\alpha+4\beta+\gamma+\delta$ greater if $\alpha+\beta\geqslant 0$

$M_{\tilde{u}}^{\mathcal{B}}$:

- $5\alpha+4\delta+3\eta$
  - Monomials for u in $\mathcal{B}$:
    - $3\alpha+4\delta+3\eta$ greater if $-\alpha\geqslant 0$
    - $6\alpha+4\delta+3\eta$ greater if $\alpha\geqslant 0$
- $5\alpha+2\beta+2\delta+\eta$
  - Monomials for u in $\mathcal{B}$:
    - $4\alpha+2\beta+2\delta+\eta$ greater if $-\alpha\geqslant 0$
    - $6\alpha+2\beta+2\delta+\eta$ greater if $\alpha\geqslant 0$
- $\alpha+3\beta+3\gamma$
  - Monomials for u in $\mathcal{B}$:
    - $\alpha+\beta+3\gamma$ greater if $-\beta\geqslant 0$
    - $\alpha+4\beta+3\gamma$ greater if $\beta\geqslant 0$

(I2)

$M_{\tilde{v}}^{\mathcal{A}}$:

- $5\alpha+4\beta+\gamma+\delta$
  - Monomials for v in $\mathcal{A}$:
    - $8\alpha+7\beta+\gamma+\delta$ greater if $\alpha+\beta\geqslant 0$
    - $\alpha+\gamma+\delta$ greater if $-\alpha-\beta\geqslant 0$
- $4\alpha+3\beta+\gamma+\delta$
  - Monomials for v in $\mathcal{A}$:
    - $8\alpha+7\beta+\gamma+\delta$ greater if $\alpha+\beta\geqslant 0$
    - $\alpha+\gamma+\delta$ greater if $-\alpha-\beta\geqslant 0$

$M_{\tilde{v}}^{\mathcal{B}}$:

- $6\alpha+\beta+4\delta+3\eta$
  - Monomials for v in $\mathcal{B}$:
    - $7\alpha+\beta+4\delta+3\eta$ greater if $\alpha\geqslant 0$
    - $2\alpha+\beta+4\delta+3\eta$ greater if $-\alpha\geqslant 0$
- $5\alpha+4\delta+4\eta$
  - Monomials for v in $\mathcal{B}$:
    - $6\alpha+4\delta+4\eta$ greater if $\alpha\geqslant 0$
    - $2\alpha+4\delta+4\eta$ greater if $-\alpha\geqslant 0$
- $6\alpha+3\beta+2\delta+\eta$
  - Monomials for v in $\mathcal{B}$:
    - $8\alpha+3\beta+2\delta+\eta$ greater if $\alpha\geqslant 0$
    - $\alpha+3\beta+2\delta+\eta$ greater if $-\alpha\geqslant 0$
- $5\alpha+4\delta+3\eta$
  - Monomials for v in $\mathcal{B}$:
    - $7\alpha+4\delta+3\eta$ greater if $\alpha\geqslant 0$
    - $2\alpha+4\delta+3\eta$ greater if $-\alpha\geqslant 0$
- $5\alpha+2\beta+2\delta+\eta$
  - Monomials for v in $\mathcal{B}$:
    - $7\alpha+2\beta+2\delta+\eta$ greater if $\alpha\geqslant 0$
    - $2\alpha+2\beta+2\delta+\eta$ greater if $-\alpha\geqslant 0$