IDENTITIES IN PLACTIC, HYPOPLACTIC, SYLVESTER, BAXTER, AND RELATED MONOIDS

ALAN J. CAIN AND ANTÓNIO MALHEIRO

ABSTRACT. This paper considers whether non-trivial identities are satisfied by certain ‘plactic-like’ monoids that, like the plactic monoid, are closely connected to combinatorics. New results show that the hypoplactic, sylvester, Baxter, stalactic, and taiga monoids satisfy identities, and indeed give shortest identities satisfied by these monoids. The existing state of knowledge is discussed for the plactic monoid and left and right patience sorting monoids.

1. INTRODUCTION

The ubiquitous plactic monoid, whose elements can be viewed as semistandard Young tableaux, and which appears in such diverse contexts as symmetric functions [Mac08], representation theory [Ful97], algebraic combinatorics [Lot02], Kostka–Foulkes polynomials [LS81, LS78], Schubert polynomials [LS85, LS90], and musical theory [Jed11], is one of a family of ‘plactic-like’ monoids that are closely connected with combinatorics. These monoids include the hypoplactic monoid [KT97, Nov00], the sylvester monoid [HNT05], the taiga monoid [Pri13], the stalactic monoid [HNT07, Pri13], the Baxter monoid [Gir12], and the left and right patience sorting monoids [Rey07, CMS]. Each of these monoids is obtained by factoring the free monoid $A^*$ over the infinite ordered alphabet $A = \{1 < 2 < 3 < \ldots\}$ by a congruence that arises from a so-called insertion algorithm that computes a combinatorial object from a word. For instance, for the plactic monoid, the corresponding combinatorial objects are (semistandard) Young tableaux; for the sylvester monoid, they are binary search trees.

An identity is a formal equality between two words in the free monoid, and is non-trivial if the two words are distinct. A monoid $M$ satisfies such an identity if the equality in $M$ holds under every substitution of letters in the words by elements of $M$. For example, any commutative monoid satisfies the non-trivial identity $xy = yx$. A finitely generated group has polynomial growth if and only if it is virtually nilpotent [Gro81], and a virtually nilpotent group satisfies a non-trivial identity [Mal53, NT63]. Thus it is natural to ask whether every finitely generated semigroup or monoid with polynomial growth satisfies a non-trivial identity. Schnerson [Shl93] provided the first counterexample, but it remains to be seen whether there is a...
Table 1. Examples of non-trivial identities satisfied by ‘plactic-like’ monoids. The stated identities are always shortest non-trivial identities satisfied by the corresponding monoid, but there may be other identities of the same length.

<table>
<thead>
<tr>
<th>Monoid</th>
<th>Symbol</th>
<th>Identity satisfied</th>
<th>Discussed in</th>
<th>Original result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plactic</td>
<td>plac</td>
<td>None</td>
<td>Subsec. 3.7</td>
<td>[CKK$^+$]</td>
</tr>
<tr>
<td>Hypoplactic</td>
<td>hypo</td>
<td>$xyxy = yxyx$</td>
<td>Subsec. 3.2</td>
<td>Present paper</td>
</tr>
<tr>
<td>Sylvester</td>
<td>sylv</td>
<td>$xyxy = yxyx$</td>
<td>Subsec. 3.4</td>
<td>Present paper</td>
</tr>
<tr>
<td>#-sylvester</td>
<td>sylv$^#$</td>
<td>$yxyx = yxyx$</td>
<td>Subsec. 3.4</td>
<td>Present paper</td>
</tr>
<tr>
<td>Baxter</td>
<td>baxt</td>
<td>$yxxyxy = yxyxy$</td>
<td>Subsec. 3.5</td>
<td>Present paper</td>
</tr>
<tr>
<td>Stalactic</td>
<td>stal</td>
<td>$xyx = yxx$</td>
<td>Subsec. 3.3</td>
<td>Present paper</td>
</tr>
<tr>
<td>Taiga</td>
<td>taig</td>
<td>$xyx = yxx$</td>
<td>Subsec. 3.3</td>
<td>Present paper</td>
</tr>
<tr>
<td>Left patience sorting</td>
<td>IPS</td>
<td>None</td>
<td>Subsec. 3.6</td>
<td>CMS</td>
</tr>
<tr>
<td>Right patience sorting</td>
<td>rPS</td>
<td>None</td>
<td>Subsec. 3.6</td>
<td>CMS</td>
</tr>
</tbody>
</table>
‘natural’ finitely generated semigroup with polynomial growth that does not satisfy a non-trivial identity. It is easy to see that the finite-rank analogues of the plactic monoid and the other related monoids discussed above have polynomial growth. This naturally leads to the question of whether these monoids satisfy non-trivial identities, for if any of them failed to do so, it would certainly be a very natural example of a polynomial-growth monoid that does not satisfy a non-trivial identity.

Further motivation for this question comes from a result of Jaszuńska & Okniński, who proved that the Chinese monoid \([CEK^+01]\), which has the same growth type as the plactic monoid \([DK94]\) but which does not arise from such a natural combinatorial object, satisfies Adian’s identity \(xyyxxyxyyx = xyyxyxxyyx\) \([JO11\text{ Corollary 3.3.4}]\). (This is the shortest non-trivial identity satisfied by the bicyclic monoid \([Adi66\text{ Chapter IV, Theorem 2}]\).

The goal of this paper is to present new results showing that some of these monoids satisfy non-trivial identities, and to survey the state of knowledge for other monoids in this family. New results show that the hypoplactic, sylvester, baxter, stalactic, and taiga monoids satisfy non-trivial identities. A discussion of the situation for the left and right patience sorting monoids and plactic monoids completes the paper. Table 1 summarizes the results.

2. ‘Plactic-like’ monoids

In this section, we recall only the definition and essential facts about the various monoids; for further background, see \([Lot02\text{ Ch. 5}]\) on the plactic monoid, \([Nov00]\) on the hypoplactic monoid, \([HNT05]\) on the sylvester monoid; \([Pri13, \S 5]\) on the taiga monoid; \([Pri13]\) on the stalactic monoid; \([Gir12]\) on the Baxter monoid; and \([CMS]\) on the patience sorting monoids.

2.1. Alphabets and words. For any alphabet \(X\), the free monoid (that is, the set of all words, including the empty word) on the alphabet \(X\) is denoted \(X^*\). The empty word is denoted \(\varepsilon\). For any \(u \in X^*\), the length of \(u\) is denoted \(|u|\), and, for any \(x \in X\), the number of times the symbol \(x\) appears in \(u\) is denoted \(|u|_x\).

Throughout the paper, \(A = \{1 < 2 < 3 < \ldots\}\) is the set of natural numbers viewed as an infinite ordered alphabet, and \(A_n = \{1 < 2 < \ldots < n\}\) is set of the first \(n\) natural numbers viewed as a finite ordered alphabet.

2.2. Combinatorial objects and insertion algorithms. A Young tableau is a finite array of symbols from \(A\), with rows non-decreasing from left to right and columns strictly increasing from top to bottom, with shorter rows below longer ones, and with rows left-justified. An example of a Young tableau is

\[
\begin{array}{cccc}
1 & 1 & 1 & 2 \\
3 & 3 & 5 & 6 \\
6 & & & \\
\end{array}
\]

(2.1)

The following algorithm takes a Young tableau and a symbol from \(A\) and yields a new Young tableau:
Algorithm 2.1 (Schensted’s algorithm). Input: A Young tableau $T$ and a symbol $a \in A$.

1. If $a$ is greater than or equal to every entry in the topmost row of $T$, add $a$ as an entry at the rightmost end of $T$ and output the resulting tableau.

2. Otherwise, let $z$ be the leftmost entry in the top row of $T$ that is strictly greater than $a$. Replace $z$ by $a$ in the topmost row and recursively insert $z$ into the tableau formed by the rows of $T$ below the topmost. (Note that the recursion may end with an insertion into an ‘empty row’ below the existing rows of $T$.)

A quasi-ribbon tableau is a finite array of symbols from $A$, with rows non-decreasing from left to right and columns strictly increasing from top to bottom, that does not contain any $2 \times 2$ subarray (that is, of the form \[ \begin{array}{cc} & \vline \hline & \\
\end{array} \]). An example of a quasi-ribbon tableau is:

(2.2)
\[
\begin{array}{cccc}
1 & 1 & 1 & 2 \\
3 & 3 & 5 & 5 \\
6 & 6 & \\
\end{array}
\]

Notice that the same symbol cannot appear in two different rows of a quasi-ribbon tableau. There is also an insertion algorithm for quasi-ribbon tableau:

Algorithm 2.2 ([Nov00 Algorithm 4.4]). Input: A quasi-ribbon tableau $T$ and a symbol $a \in A$.

If there is no entry in $T$ that is less than or equal to $a$, output the tableau obtained by putting $a$ and gluing $T$ by its top-leftmost entry to the bottom of $a$.

Otherwise, let $x$ be the right-most and bottom-most entry of $T$ that is less than or equal to $a$. Put a new entry $a$ to the right of $x$ and glue the remaining part of $T$ (below and to the right of $x$) onto the bottom of the new entry $a$. Output the new tableau.

A right strict binary search tree is a labelled rooted binary tree where the label of each node is greater than or equal to the label of every node in its left subtree, and strictly less than every node in its right subtree. An example of a binary search tree is

(2.3)
\[
\begin{array}{c}
1 \\
1 & 2 \\
1 & 3 & 5 \\
1 & 3 & 5 & 6 \\
1 & 3 & 5 & 6 \\
\end{array}
\]

The insertion algorithm for right strict binary search trees adds the new symbol as a leaf node in the unique place that maintains the property of being a right strict binary search tree:

Algorithm 2.3 ([HNT05 § 3.3]). Input: A right strict binary search tree $T$ and a symbol $a \in A$.

If $T$ is empty, create a node and label it $a$. If $T$ is non-empty, examine the label $x$ of the root node; if $a \geq x$, recursively insert $a$ into the right subtree.
of the root node; otherwise recursively insert $a$ into the left subtree of the root node. Output the resulting tree.

A left strict binary search tree is a labelled rooted binary tree where the label of each node is strictly greater than the label of every node in its left subtree, and less than or equal to every node in its right subtree; see the left tree shown in (2.4) below for an example.

The insertion algorithm for left strict binary search trees is dual to Algorithm 2.3 above:

**Algorithm 2.4.** Input: A left strict binary search tree $T$ and a symbol $a \in \mathcal{A}$.

- If $T$ is empty, create a node and label it $a$. If $T$ is non-empty, examine the label $x$ of the root node; if $a \leq x$, recursively insert $a$ into the left subtree of the root node; otherwise recursively insert $a$ into the right subtree of the root note. Output the resulting tree.

The canopy of a (labelled or unlabelled) binary tree $T$ is the word over \{0, 1\} obtained by traversing the empty subtrees of the nodes of $T$ from left to right, except the first and the last, labelling an empty left subtree by 1 and an empty right subtree by 0. (See (2.4) below for examples of canopies.)

A pair of twin binary search trees consist of a left strict binary search tree $T_L$ and a right strict binary search tree $T_R$, such that $T_L$ and $T_R$ contain the same symbols, and the canopies of $T_L$ and $T_R$ are complementary, in the sense that the $i$-th symbol of the canopy of $T_L$ is 0 (respectively 1) if and only if the $i$-th symbol of the canopy of $T_R$ is 1 (respectively 0). The following is an example of a pair of twin binary search trees, with the complementary canopies 0110101 and 1001010 shown in grey:

(2.4)

A stalactic tableau is a finite array of symbols of $\mathcal{A}$ in which columns are top-aligned, and two symbols appear in the same column if and only if they are equal. For example,

(2.5)

is a stalactic tableau. The insertion algorithm is very straightforward:

**Algorithm 2.5** ([HNT07 § 3.7]). Input: A stalactic tableau $T$ and a symbol $a \in \mathcal{A}$.
If \( a \) does not appear in \( T \), add \( a \) to the left of the top row of \( T \). If \( a \) does appear in \( T \), add \( a \) to the bottom of the (by definition, unique) column in which \( a \) appears. Output the new tableau.

A binary search tree with multiplicities is a labelled binary search tree in which each label appears at most once, and where a non-negative integer called the multiplicity is assigned to each node label. An example of a binary search tree is:

\[
(2.6)
\]

(The superscripts on the labels in each node denote the multiplicities.)

**Algorithm 2.6.** ([PrHu93] Algorithm 3) *Input:* A binary search tree with multiplicities \( T \) and a symbol \( a \in A \).

If \( T \) is empty, create a node, label it by \( a \), and assign it multiplicity 1. If \( T \) is non-empty, examine the label \( x \) of the root node; if \( a < x \), recursively insert \( a \) into the left subtree of the root node; if \( a > x \), recursively insert \( a \) into the right subtree of the root note; if \( a = x \), increment by 1 the multiplicity of the node label \( x \).

An lPS tableau is a finite array of symbols of \( A \) in which columns are top-aligned, the entries in the top row are non-decreasing from left to right, and the entries in each column are strictly increasing from top to bottom. For example,

\[
(2.7)
\]

is an lPS tableau. The insertion algorithm is very straightforward:

**Algorithm 2.7** ([TY11 § 3.2]). *Input:* An lPS tableau \( T \) and a symbol \( a \in A \).

If \( a \) is greater than or equal to every symbol that appears in the top row of \( T \), add \( a \) to the right of the top row of \( T \). Otherwise, let \( C \) be the leftmost column whose topmost symbol is strictly greater than \( a \). Slide column \( C \) down by one space and add \( a \) as a new entry on top of \( C \). Output the new tableau.

An rPS tableau is a finite array of symbols of \( A \) in which columns are top-aligned, the entries in the top row are increasing from left to right, and the entries in each column are non-decreasing from top to bottom. For example,

\[
(2.8)
\]

is an rPS tableau. Again, the insertion algorithm is very straightforward:

**Algorithm 2.8** ([TY11 § 3.2]). *Input:* An rPS tableau \( T \) and a symbol \( a \in A \).
If $a$ is greater than every symbol that appears in the top row of $T$, add $a$ to the right of the top row of $T$. Otherwise, let $C$ be the leftmost column whose topmost symbol is greater than or equal to $a$. Slide column $C$ down by one space and add $a$ as a new entry on top of $C$. Output the new tableau.

2.3. Monoids from insertion. Let $M \in \{ \text{plac, hypo, sylv, sylv}^#, \text{baxt, stal, taig, lPS, rPS} \}$ and $u \in \mathbb{A}^*$. Using the insertion algorithms described above, one can compute from $u$ a combinatorial object $P_M(u)$ of the type associated to $M$ in Table 2. If $M \neq \text{baxt}$, one computes $P_M(u)$ by starting with the empty combinatorial object and inserting the symbols of $u$ one-by-one using the appropriate insertion algorithm and proceeding through the word $u$ either left-to-right or right-to-left as shown in the table. For $M = \text{baxt}$, one uses a slightly different procedure: $P_{\text{baxt}}(u)$ is the pair of twin binary search trees $(P_{\text{sylv}^#}(u), P_{\text{sylv}}(u))$.

For each $M \in \{ \text{plac, hypo, sylv, sylv}^#, \text{baxt, stal, taig, lPS, rPS} \}$, define the relation $\equiv_M$ by

$$u \equiv_M v \iff P_M(u) = P_M(v).$$

In each case, the relation $\equiv_M$ is a congruence on $\mathbb{A}^*$, and so the factor monoid $M = \mathbb{A}^*/\equiv_M$ can be formed, and is named as in Table 2. The rank-$n$ analogue is the factor monoid $M_n = \mathbb{A}^*_n/\equiv_M$, where the relation $\equiv_M$ is naturally restricted to $\mathbb{A}^*_n \times \mathbb{A}^*_n$.

It follows from the definition of $\equiv_M$ (for any $M \in \{ \text{plac, hypo, sylv, sylv}^#, \text{baxt, stal, taig, lPS, rPS} \}$) that each element $[u]_{\equiv_M}$ of the factor monoid $M$ can be identified with the combinatorial object $P_M(u)$.

The evaluation (also called the content) of a word $u \in \mathbb{A}^*$, denoted $ev(u)$, is the infinite tuple of non-negative integers, indexed by $\mathbb{A}$, whose $a$-th element is $|u|_a$; thus this tuple describes the number of each symbol in $\mathbb{A}$ that appears in $u$. It is immediate from the definition of the monoids above that if $u \equiv_M v$, then $ev(u) = ev(v)$, and hence it makes sense to define the evaluation of an element $p$ of one of these monoids to be the evaluation of any word representing it.

Note that $M_n$ is a submonoid of $M$ for all $n$, and that $M_m$ is a submonoid of $M_n$ for all $m \leq n$. In each case, $M_1$ is a free monogenic monoid and thus commutative, but that $M_n$ is non-commutative for $n \geq 2$ (and thus $M$ is also non-commutative).

3. Identities

Subsections 3.2 to 3.5 find shortest non-trivial identities satisfied by hypo, stal, taig, sylv, sylv#, and baxt, in that order. Subsection 3.6 discusses the situation for lPS and rPS, and Subsection 3.7 discusses the current state of knowledge for plac.

3.1. Preliminaries. An identity over an alphabet $X$ is a formal equality $u = v$, where $u$ and $v$ are words in the free monoid $X^*$, and is non-trivial if $u$ and $v$ are not equal. The elements of $X$ are called variables. A monoid $M$ satisfies the identity $u = v$ if, for every homomorphism $\phi : X^* \to M$, the equality $\phi(u) = \phi(v)$ holds in $M$. It is often useful to think of this in the informal way mentioned in the introduction: $M$ satisfies the identity $u = v$ if equality in $M$ holds under every substitution of variables in the words $u$
Table 2. Insertion algorithms used to compute combinatorial objects, and the corresponding monoids. ‘L-to-R’ and ‘R-to-L’ abbreviate ‘left-to-right’ and ‘right-to-left’, respectively. The algorithm used to compute \( P_{\text{baxt}}(\cdot) \) is a special case and is discussed in the main text.

<table>
<thead>
<tr>
<th>Monoid</th>
<th>Symbol</th>
<th>Combinatorial object</th>
<th>Algorithm</th>
<th>Direction</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plactic</td>
<td>plac</td>
<td>Young tableau</td>
<td>2.1</td>
<td>L-to-R</td>
<td>( P_{\text{plac}}(3613151265) ) is (2.1)</td>
</tr>
<tr>
<td>Hypoplactic</td>
<td>hypo</td>
<td>Quasi-ribbon tableau</td>
<td>2.2</td>
<td>L-to-R</td>
<td>( P_{\text{hypo}}(3613151265) ) is (2.2)</td>
</tr>
<tr>
<td>Sylvester</td>
<td>sylv</td>
<td>Right strict binary search tree</td>
<td>2.3</td>
<td>R-to-L</td>
<td>( P_{\text{sylv}}(3613151265) ) is (2.3)</td>
</tr>
<tr>
<td>#-sylvester</td>
<td>sylv#</td>
<td>Left strict binary search tree</td>
<td>2.4</td>
<td>L-to-R</td>
<td>( P_{\text{sylv}}(3613151265) ) is (2.3)</td>
</tr>
<tr>
<td>Baxter</td>
<td>baxt</td>
<td>Pair of twin BSTs</td>
<td>—</td>
<td>—</td>
<td>( P_{\text{baxt}}(3613151265) ) is (2.4)</td>
</tr>
<tr>
<td>Stalactic</td>
<td>stal</td>
<td>Stalactic tableau</td>
<td>2.5</td>
<td>R-to-L</td>
<td>( P_{\text{stal}}(3613151265) ) is (2.5)</td>
</tr>
<tr>
<td>Taiga</td>
<td>taig</td>
<td>BST with multiplicities</td>
<td>2.6</td>
<td>R-to-L</td>
<td>( P_{\text{taig}}(3613151265) ) is (2.6)</td>
</tr>
<tr>
<td>Left patience sorting</td>
<td>IPS</td>
<td>IPS tableau</td>
<td>2.7</td>
<td>L-to-R</td>
<td>( P_{\text{IPS}}(3613151265) ) is (2.7)</td>
</tr>
<tr>
<td>Right patience sorting</td>
<td>rPS</td>
<td>rPS tableau</td>
<td>2.8</td>
<td>L-to-R</td>
<td>( P_{\text{rPS}}(3613151265) ) is (2.8)</td>
</tr>
</tbody>
</table>
and \(v\) by elements of \(M\). Note that if a monoid satisfies an identity \(u = v\), all of its submonoids and all of its homomorphic images satisfy \(u = v\).

**Lemma 3.1.** Let \(M\) be a monoid satisfying a non-trivial identity. Then the shortest (in terms of the sums of the lengths of the two sides) non-trivial identity satisfied by \(M\) is an identity over an alphabet with at most two variables.

**Proof.** It is sufficient to show that, for any non-trivial identity over an alphabet with at most three variables, there is a strictly shorter identity with at most two variables. So suppose that \(u = v\) is a non-trivial identity over \(X\) satisfied by \(M\), where \(|X| \geq 3\). Interchanging \(u\) and \(v\) if necessary, assume \(|u| \leq |v|\). Let \(e_M\) be the identity (that is, the multiplicative neutral element) of \(M\).

First consider the case where \(u\) is a prefix of \(v\). By non-triviality, \(u\) is strictly shorter than \(v\), so that \(v = uw\) for some non-empty word \(w\). Let \(x \in X\) be a variable that appears in \(w\). So \(x^{[\varepsilon]} = x^{[\varepsilon]}\) is a non-trivial identity. To see that it is satisfied by \(M\), proceed as follows: Let \(\phi' : \{x\}^* \to M\) be a homomorphism. Extend \(\phi'\) to a homomorphism \(\phi : X \to M\) by defining \(\phi(z) = e_M\) for all \(z \in X \setminus \{x\}\). Note that \(\phi'(x^{[\varepsilon]}) = \phi(u)\) and \(\phi'(x^{[\varepsilon]}) = \phi(v)\) by the definition of \(\phi'\), and that \(\phi'(u) = \phi'(v)\) since \(M\) satisfies \(u = v\). Combining these equalities shows that \(\phi(x^{[\varepsilon]}) = \phi(x^{[\varepsilon]})\). Hence \(M\) satisfies the non-trivial identity \(x^{[\varepsilon]} = x^{[\varepsilon]}\) over the alphabet \(\{x\}\).

Now suppose \(u\) is not a prefix of \(v\). Suppose \(u = u_1 \cdots u_{|u|}\) and \(v = v_1 \cdots v_{|v|}\). Let \(i\) be minimal such that the variables \(u_i\) and \(v_i\) are different; suppose these variables are \(x\) and \(y\). Let \(u'\) and \(v'\) be obtained from \(u\) and \(v\) respectively by deleting all variables other than \(x\) and \(y\). By the choice of \(i\), \(u' = v'\) is a non-trivial identity. To see that it is satisfied by \(M\), proceed as follows: Let \(\phi' : \{x, y\}^* \to M\) be a homomorphism. Extend \(\phi'\) to a homomorphism \(\phi : X^* \to M\) by defining \(\phi(z) = e_M\) for all \(z \in X \setminus \{x, y\}\). Note that \(\phi(u) = \phi'(u')\) and \(\phi(v) = \phi'(v')\) by the definitions of the words \(u'\) and \(v'\) and the homomorphism \(\phi\); note also that \(\phi(u) = \phi(v)\) since \(M\) satisfies the identity \(u = v\). Combining these equalities shows that \(\phi'(u') = \phi'(v')\). So \(M\) satisfies the non-trivial identity \(u' = v'\) over the alphabet \(\{x, y\}\).

Note that in both cases the resulting identity is strictly shorter than the original one. \(\Box\)

**Lemma 3.2.** Let \(M\) be a monoid that contains a free monogenic submonoid and suppose that \(M\) satisfies an identity \(u = v\) over an alphabet \(X\). Then \(|u| = |v|\) and \(|u|^X = |v|^X\) for all \(x \in X\). Consequently, if \(u = v\) is non-trivial, then \(|X| \geq 2\).

**Proof.** Let \(a \in M\) generate a free monogenic submonoid of \(M\) and let \(e_M\) be the identity of \(M\). Let \(x \in X\). Define

\[
\phi_x : X^* \to M, \quad x \mapsto a, \quad z \mapsto e_M \text{ for all } z \in X \setminus \{x\}.
\]

Then \(a^{[\varepsilon]} = \phi_x(u) = \phi_x(v) = a^{[\varepsilon]}\) since \(M\) satisfies \(u = v\). Since \(a\) generates a free monogenic submonoid, \(|u|^X = |v|^X\). Since this holds for all \(x \in X\), it follows that \(|u| = |v|\). \(\Box\)
The length of an identity \( u = v \) with \(|u| = |v|\) is defined to be \(|u| = |v|\); note that this applies to any identity satisfied by a monoid that contains a free monogenic submonoid by Lemma 3.2.

Two identities \( u = v \) and \( u' = v' \) are equivalent if one can be obtained from the other by possibly renaming variables and swapping the two sides of the equality. For example, \( xyyx = yxyx \) and \( xyyx = yxyx \) are equivalent, since the second can be obtained from the first by interchanging the variables \( x \) and \( y \) and swapping the two sides.

**Lemma 3.3.**

1. Up to equivalence, the only length-2 non-trivial identity that can be satisfied by a monoid that contains a free monogenic submonoid is \( xy = yx \); thus any homogeneous monoid satisfying a length-2 identity is commutative.

2. Up to equivalence, the only length-3 non-trivial identities over a two-letter alphabet that can be satisfied by a monoid that contains a free monogenic submonoid are

\[
xy = xy, \quad xxy = yxx, \quad xyx = yxx.
\]

**Proof.**

1. Let \( u_1v_2 = v_1v_2 \) be a non-trivial identity satisfied by some monoid that contains a free monogenic submonoid. Suppose \( u_1 \) and \( v_1 \) are the same variable. Then Lemma 3.2 implies that \( u_2 \) and \( v_2 \) are also the same variable, contradicting non-triviality. Thus \( u_1 \) and \( v_1 \) are different variables. Then Lemma 3.2 implies that \( u_2 \) is the same variable as \( v_1 \) and \( v_2 \) is the same variable as \( u_1 \). Thus, up to equivalence, the identity is \( xy = yx \).

2. Let \( u_1u_2u_3 = v_1v_2v_3 \) (where \( u_i, v_i \in \{x, y\} \)) be a non-trivial identity satisfied by some homogeneous monoid. Let \( I \) be the set of indices \( i \in \{1, 2, 3\} \) where the variables \( u_i \) and \( v_i \) are the same. Clearly, non-triviality shows that \(|I| < 3\). If \(|I| = 2\), then, interchanging \( x \) and \( y \) if necessary, there is a unique \( i \in \{1, 2, 3\} \) where \( u_i \) is \( x \) and \( v_i \) is \( y \). Then \(|u|_x = 1 + |v|_x \), since the words \( u \) and \( v \) differ only at the \( i \)-th symbol; this contradicts Lemma 3.2. If \(|I| = 0\), then every symbol of \( u \) differs from every symbol of \( v \). Since \(|u| = 3\), at least one of \(|u|_x \) and \(|u|_y \) is at least 2. Interchanging \( x \) and \( y \) if necessary, assume \(|u|_x \geq 2\). Then \(|v|_x = |u|_y \leq 1\), which is a contradiction.

The only remaining possibility is \(|I| = 1\). Interchanging \( x \) and \( y \) if necessary, assume that \( u_i \) and \( v_i \) are both \( x \) for exactly one \( i \in \{1, 2, 3\} \). For each of the three possibilities, the conditions \(|u|_x = |v|_x \) and \(|u|_y = |v|_y \) require the other symbols in \( u \) must be \( x \) and \( y \), with the corresponding symbols in \( v \) being the opposite. Each possibility gives one of the given identities. \( \square \)

Let \( M \in \{\text{plac, hypo, sylv, taig, stal, baxt, lPS, rPS}\} \). Then \( M \) contains the free monogenic submonoid \( M_1 \). (Note that \( M_1 \) thus satisfies the non-trivial identity \( xy = yx \).) By Lemma 3.2 if \( M \) satisfies a non-trivial identity, it must be over an alphabet with at least two variables. The aim is to find shortest non-trivial identities satisfied by each monoid \( M \): Lemma 3.1 and 3.2 show that it will suffice to consider identities \( u = v \) over \( \{x, y\} \) with \(|u| = |v| \). Note that since none of the monoids \( M \) is commutative, any identity must have length at least 3.
3.2. Hypoplactic monoid. The authors proved the following result using a quasi-crystal structure for the hypoplactic monoid [CM17, Theorem 9.3]; this subsection presents a direct proof.

**Proposition 3.4.** The hypoplactic monoid satisfies the following non-trivial identities
\[ xyxy = yxyx = yxxy = yxyx; \]
\[ xxyx = xyxx. \]
Furthermore, up to equivalence, these are the shortest non-trivial identities satisfied by the hypoplactic monoid.

**Proof.** The first goal is to show that these identities are satisfied by hypo.
Let \( s, t \in \text{hypo} \), and let \( p, q \in \mathcal{A}^* \) be words representing \( s, t \), respectively.
Let \( B = \{ a_1 < \ldots < a_k \} \) be the set of symbols in \( \mathcal{A} \) that appear in at least one of \( p \) and \( q \).

By Algorithm 2.2, symbols \( a_i+1 \) and \( a_i \) are on the same row of \( P_{\text{hypo}}(w) \) (for any \( w \in B^* \)) if and only if the word \( w \) does not contain a symbol \( a_i \) somewhere to the right of a symbol \( a_i+1 \) (since inserting this symbol \( a_i \) results in the part of the quasi-ribbon tableau that contains \( a_i+1 \) being glued below the entry \( a_i \)).

Suppose \( ppqq \) contains a symbol \( a_i \) somewhere to the right of a symbol \( a_i+1 \). Then, regardless of whether these symbols both lie in \( u \), both lie in \( v \), or one lies in \( u \) and the other lies in \( v \), the word \( ppqq \) also contains a symbol \( a_i \) to the right of a symbol \( a_i+1 \). Similar reasoning establishes that if any of the words \( ppqq, qppq, qppq, ppqp, ppqp, or pqpp, \) contains a symbol \( a_i \) somewhere to the right of a symbol \( a_i+1 \), so do all the others. Hence either the symbols \( a_i \) and \( a_i+1 \) are on the same row in all of \( P_{\text{hypo}}(ppqq), \)
\( P_{\text{hypo}}(qppq), P_{\text{hypo}}(ppqp), P_{\text{hypo}}(ppqp), \)
and \( P_{\text{hypo}}(pqpp), \) or on different rows in all of them.

A quasi-ribbon tableau is clearly determined by its evaluation and by knowledge of whether adjacent different entries are on the same row. Thus, noting that \( \text{ev}(ppqq) = \text{ev}(qppq) = \text{ev}(ppqp) = \text{ev}(pqpp), \) it follows from the previous paragraph that
\[ stst = P_{\text{hypo}}(ppqp) = tsst = P_{\text{hypo}}(qppq) \]
\[ = tsts = P_{\text{hypo}}(qqpq) = stts = P_{\text{hypo}}(pqpp). \]

Similarly, noting that \( \text{ev}(ppqp) = \text{ev}(pqpp), \) it follows that
\[ ssts = P_{\text{hypo}}(ppqp) = P_{\text{hypo}}(pqpp) = stss. \]

The next goal is to show that, up to equivalence, these are the only length-4 non-trivial identities satisfied by hypo. Let \( u = v \) be a length-4 non-trivial identity over \( \{ x, y \} \) satisfied by hypo. By Lemma 3.2, \( |u_x| = |v_x| \) and \( |u_y| = |v_y| \).

By Algorithm 2.2, symbols 1 and 2 are on different rows of \( P_{\text{hypo}}(w) \) (for any \( w \in \{1, 2\}^* \)) only if the word \( w \) contain a symbol 1 somewhere to the right of a symbol 2. Suppose \( u \) contains a factor \( xy \). Let \( \phi : \{ x, y \}^* \to \text{hypo} \) map \( x \) to 2 and \( y \) to 1; then \( P_{\text{hypo}}(\phi(u)) \) has 1 and 2 on different rows. The same must hold for \( P_{\text{hypo}}(\phi(v)) \), and thus \( v \) must contain a factor \( xy \). The converse is similar, and the reasoning is parallel for factors \( yx \). Thus \( u \)
contains a factor \( xy \) (respectively, \( yx \)) if and only if \( v \) contains a factor \( xy \) (respectively, \( yx \)).

If both \( u \) and \( v \) contain only factors \( xy \) (respectively, \( yx \)), then \( u = x^{|u|_x} y^{|u|_y} = x^{|v|_x} y^{|v|_y} = v \), which contradicts non-triviality. Similarly, it is impossible for \( u \) and \( v \) to contain only factors \( yx \). So both \( u \) and \( v \) contain factors \( xy \) and \( yx \).

Interchanging \( x \) and \( y \) if necessary, assume \(|u|_x = |v|_x \geq |u|_y = |v|_y |y| = 1 \). Since \( u \) and \( v \) both contain \( xy \) and \( yx \), this implies that \( u,v \in \{xxyx, xyxx\} \). Now consider the case \(|u|_x = |v|_x = 2 \) and \(|u|_y = |v|_y = 2 \). Again, since \( u \) and \( v \) both contain \( xy \) and \( yx \), this implies that \( u,v \in \{xxyy, xyyx, yxyy, yyyx\} \). Thus \( u = v \) is equivalent to one of the identities in the statement.

The final goal is to prove that no length-3 non-trivial identity is satisfied by \( \text{hypo} \). Consider the length-3 identities from Lemma 3.3(2):

\[
\begin{align*}
xx & = yxx, & xyy & = yxx, & xy & = yxy.
\end{align*}
\]

If one puts \( x = 1 \) and \( y = 2 \) into the first two identities and \( x = 2 \) and \( y = 1 \) in the third, one sees that \( \text{hypo} \) satisfies none of them:

\[
\begin{align*}
\text{P}_{\text{hypo}}(112) & = 1 1 2 \neq 1 1 2 = \text{P}_{\text{hypo}}(121); \\
\text{P}_{\text{hypo}}(112) & = 1 1 2 \neq 1 1 2 = \text{P}_{\text{hypo}}(211); \\
\text{P}_{\text{hypo}}(212) & = 1 2 2 \neq 1 2 2 = \text{P}_{\text{hypo}}(122).
\end{align*}
\]

Thus \( \text{hypo} \) satisfies no length-3 identity. □

3.3. Stalactic and taiga monoids.

**Lemma 3.5.** The stalactic monoid satisfies the non-trivial identity \( xyx = yxx \).

**Proof.** Let \( x,y \in \text{stal} \) and let \( u,v \in \mathcal{A}^* \) be words representing \( x,y \), respectively. Since the tree \( P_{\text{stal}}(w) \) is computed by applying Algorithm 2.5 to each symbol in the word \( w \in \mathcal{A}^* \), proceeding right to left, it is clear that the order of the rightmost appearances of each symbol in \( w \) determines order of symbols in the top row of the stalactic tableau \( P_{\text{stal}}(w) \). Since the order of rightmost appearances of each symbol in the words \( uuv \) and \( vuv \) are the same, the order of symbols in the top row of \( P_{\text{stal}}(uuv) \) and \( P_{\text{stal}}(vuv) \) is also the same. Since \( ev(uuv) = ev(vuv) \), the length of corresponding columns in \( P_{\text{stal}}(uuv) \) and \( P_{\text{stal}}(vuv) \) are equal. Hence \( xyx = P_{\text{stal}}(uuv) = P_{\text{stal}}(vuv) = yxx \). □

**Lemma 3.6.** The taiga monoid does not satisfy either of the identities \( xxy = xyx \) or \( xxy = yxx \), and does not satisfy a non-trivial identity of length 2.

**Proof.** To see that \( \text{taig} \) does not satisfy either of the identities \( xxy = xyx \) or \( xxy = yxx \), note that the rightmost symbol of \( w \in \mathcal{A}^* \) labels the root node of \( P_{\text{taig}}(w) \), but the rightmost symbols of both these identities are different. Thus substituting 1 for \( x \) and 2 for \( y \) shows that neither identity can be satisfied by \( \text{taig} \).
Finally, \textit{taig} does not satisfy a non-trivial identity of length 2 since it is clearly non-commutative.

Since the taiga monoid is a homomorphic image of the stalactic monoid \cite[§ 5]{Prim13}, any identity that is satisfied by \textit{stal} is satisfied by \textit{taig}. Combining this with Lemmata 3.3, 3.5, and 3.6 gives the following results:

**Proposition 3.7.** The stalactic monoid satisfies the non-trivial identity \( xyx = yxx \). Furthermore, this is the unique shortest non-trivial identity satisfied by the stalactic monoid.

**Proposition 3.8.** The taiga monoid satisfies the non-trivial identity \( xyx = yxx \). Furthermore, this is the unique shortest non-trivial identity satisfied by the taiga monoid.

### 3.4. Sylvester monoid

By the definition of a binary search tree, if a tree has a node with label \( a \), then any other node with label \( a \) is either above it or in its left subtree. This shows that there is a unique path from the root to a leaf node that contains all nodes labelled \( a \). In particular, there is at most one leaf node with label \( a \). Furthermore, there is a unique node \( a \) that is both the leftmost node with label \( a \) and the node labelled \( a \) that is furthest from the root. Similarly, there is a unique node \( a \) that is both the rightmost node with label \( a \) and the node labelled \( a \) that is closest to the root.

**Lemma 3.9.** The sylvester monoid is left-cancellative.

**Proof.** Let \( u, v \in \mathcal{A}^* \) and \( a \in \mathcal{A} \). Suppose that \( P_{\text{sylv}}(au) = P_{\text{sylv}}(av) \). That is, the binary search trees \( P_{\text{sylv}}(au) \) and \( P_{\text{sylv}}(av) \) are equal. Note further that the last symbol \( a \) inserted into both binary search trees is a leaf node. Deleting this (unique) leaf node labelled \( a \) from the equal trees \( P_{\text{sylv}}(au) \) and \( P_{\text{sylv}}(av) \) leaves equal binary search trees. That is, \( P_{\text{sylv}}(u) \) and \( P_{\text{sylv}}(v) \) are equal.

Since \( u \) and \( v \) are arbitrary words and \( a \) represents an arbitrary generator, it follows that \textit{sylv} is left-cancellative. \( \square \)

**Lemma 3.10.** The sylvester monoid does not satisfy an identity equivalent to one of the form \( uxx = vyx \), for any \( u, v \in \{x,y\}^* \).

**Proof.** Using the Algorithm 2.3, one sees that

\[
\begin{align*}
P_{\text{sylv}}(\cdots 22) &= \begin{array}{c} 2 \\ \end{array} \quad \text{and} \quad P_{\text{sylv}}(\cdots 12) &= \begin{array}{c} 2 \\ \end{array}
\end{align*}
\]

Note that these binary search trees differ in the left child of the root node. Thus to see that the identity \( uxx = vyx \) cannot be satisfied by \textit{sylv}, one can substitute 2 for \( x \) and 1 for \( y \). \( \square \)

**Lemma 3.11.** Let \( p, q, r \in \textit{sylv} \) be such that \( ev(p) = ev(q) = ev(r) \). Then \( pr = qr \).

**Proof.** Let \( u, v, w \in \mathcal{A}^* \) be words representing \( p, q, r \), respectively. The tree \( P_{\text{sylv}}(x) \) is computed by applying Algorithm 2.3 to each symbol in \( x \), proceeding right to left. That is, \( P_{\text{sylv}}(uw) \) and \( P_{\text{sylv}}(vw) \) are obtained by inserting
\[ u \text{ and } v \text{ (respectively) into } P_{\text{sylv}}(w). \] Since \( \text{ev}(u) = \text{ev}(v) = \text{ev}(w) \), every symbol that appears in \( u \) or \( v \) also appears in \( w \) and thus in \( P_{\text{sylv}}(w) \).

Consider how further symbols from \( u \) or \( v \) are inserted into \( P_{\text{sylv}}(w) \):

- Let \( a \) be the smallest symbol that appears in \( u \), \( v \), and \( w \). Then the leftmost node of \( P_{\text{sylv}}(w) \) must be labelled by \( a \). Thus, during the computation of \( P_{\text{sylv}}(uw) \) and \( P_{\text{sylv}}(vw) \), all symbols \( a \) in \( u \) and \( v \) will certainly be inserted into the left subtree of this leftmost node in \( P_{\text{sylv}}(w) \), and no other symbols are inserted into this left subtree by the minimality of \( a \).

- Let \( c \) be some symbol other than \( a \) that appears in \( u \), \( v \), and \( w \), and let \( b \) be the maximum symbol less than \( c \) appearing in \( u \), \( v \), and \( w \). Let \( N_c \) be the leftmost node in \( P_{\text{sylv}}(w) \) labelled by \( c \) and let \( N_b \) be the rightmost node in \( P_{\text{sylv}}(w) \) labelled by \( b \).

Suppose \( v \) is the label of the lowest common ancestor \( N_v \) of the nodes \( N_b \) and \( N_c \). If \( N_v \) is neither \( N_b \) nor \( N_c \), then \( b \leq v < c \), which implies \( v = b \) by the choice of \( b \), which contradicts the fact that \( N_b \) is the rightmost node labelled \( b \). Thus the lowest common ancestor of \( N_b \) and \( N_c \) must be one of \( N_b \) or \( N_c \). That is, one of the following must hold:

- \( N_b \) is above \( N_c \). Then \( N_c \) is in the right subtree of \( N_b \), since \( b < c \). Since there is no node that is to the right of \( N_b \) and to the left of \( N_c \), it follows that \( N_c \) has an empty left subtree. Thus any symbol \( c \) in \( u \) or \( v \) will be inserted into this currently empty left subtree of \( N_c \), and no other symbols will be inserted into this subtree by the maximality of \( b \) among the symbols less than \( c \).

- \( N_c \) is above \( N_b \). Then \( N_b \) is in the left subtree of \( N_c \), since \( b < c \). Since there is no node that is to the right of \( N_b \) and to the left of \( N_c \), it follows that \( N_b \) is the left child of \( N_c \), and that \( N_b \) has an empty right subtree. Thus any symbol \( c \) in \( u \) or \( v \) will be inserted into this currently empty right subtree of \( N_b \), and no other symbols will be inserted into this subtree by the maximality of \( b \) among the symbols less than \( c \).

Combining these cases, one sees that every symbol \( d \) from \( u \) or \( v \) is inserted into a particular previously empty subtree of \( P_{\text{sylv}}(w) \), dependent only on the value of the symbol \( d \) (and not on its position in \( u \) or \( v \)), and that unequal symbols are inserted into different subtrees. Since \( \text{ev}(u) = \text{ev}(v) \), the same number of symbols \( d \) are inserted, for each such symbol \( d \). Hence \( P_{\text{sylv}}(uw) = P_{\text{sylv}}(vw) \) and thus \( pr = P_{\text{stal}}(uw) = P_{\text{stal}}(vw) = qr \). \( \square \)

**Proposition 3.12.** The sylvester monoid satisfies the non-trivial identity \( xyxy = yxxy \). Furthermore, up to equivalence, this is the unique shortest identity satisfied by the sylvester monoid.

**Proof.** Let \( x, y \in \text{sylv} \). Let \( p = r = xy \) and \( q = yx \). Then \( \text{ev}(p) = \text{ev}(q) = \text{ev}(r) \), so \( pr = qr \) by [Lemma 3.11]. Thus \( \text{sylv} \) satisfies \( xyxy = yxxy \).

Since \( \text{hypo} \) is a homomorphic image of \( \text{sylv} \) [Pri13 Example 6], and \( \text{hypo} \) satisfies no length-3 identity by [Proposition 3.4], it follows that \( \text{sylv} \) cannot satisfy a length-3 identity.
So let \( u = v \) be some length-4 identity satisfied by \( \text{sylv} \). Suppose \( u = u_1u_2u_3u_4 \) and \( v = v_1v_2v_3v_4 \), where \( u_i, v_i \in \{x, y\} \). Since the rightmost symbol in a word \( w \) determines the root node of \( P_{\text{sylv}}(w) \), it follows that \( u_4 \) and \( v_4 \) are the same symbol; interchanging \( x \) and \( y \) if necessary, assume this is \( y \). If \( u_1 \) and \( v_1 \) were the same symbol, then the left-cancellativity of \( \text{sylv} \) would imply that \( \text{sylv} \) satisfied the length-3 identity \( u_2u_3u_4 = v_2v_3v_4 \), which contradicts the previous paragraph. Thus \( u_1 \) and \( v_1 \) are different symbols. Interchanging \( u \) and \( v \) if necessary, assume that \( u_1 \) is \( x \) and \( v_1 \) is \( y \). The condition \( |u|_y = |v|_y \) implies that at least one of \( u_2 \) and \( u_3 \) is \( y \), which yields the following three possibilities for \( u \):

\[
xyyy, \ xyxy, \ xxyy.
\]

The conditions \( |u|_x = |v|_x \) and \( |u|_y = |v|_y \) now imply the following four possibilities for the identity \( u = v \) (with the first possibility for \( u \) above giving two different identities):

\[
xyyy = yxyy, \ xyxy = yyxy, \ xxyy = yxyy, \ xxyy = yxyy.
\]

The last of these is the identity already proven to hold in \( \text{sylv} \). The second and third cannot hold in \( \text{sylv} \) by Lemma 3.10. In the first identity, putting \( x = 1 \) and \( y = 2 \) proves that it is not satisfied by \( \text{sylv} \):

\[
P_{\text{sylv}}(1222) = \begin{array}{c} 1 \end{array} \begin{array}{c} 2 \end{array} \begin{array}{c} 2 \end{array} \begin{array}{c} 2 \end{array} \neq \begin{array}{c} 1 \end{array} \begin{array}{c} 2 \end{array} \begin{array}{c} 2 \end{array} = P_{\text{sylv}}(2122)
\]

Thus \( xxyy = yxyy \) is the unique shortest identity satisfied by \( \text{sylv} \).

Since the sylvester and \( \# \)-sylvester monoids are anti-isomorphic, the following results follow by dual reasoning:

**Lemma 3.13.** The \( \# \)-sylvester monoid is right-cancellative.

**Lemma 3.14.** The \( \# \)-sylvester monoid does not satisfy an identity equivalent to one of the form \( xxyu = xyv \), for any \( u, v \in \{x, y\}^* \).

**Lemma 3.15.** Let \( p, q, s \in \text{sylv}^\# \) be such that \( \text{ev}(p) = \text{ev}(q) = \text{ev}(s) \). Then \( sp = sq \).

**Proposition 3.16.** The \( \# \)-sylvester monoid satisfies the non-trivial identity \( yxyx = yxyy \). Furthermore, up to equivalence, this is the unique shortest identity satisfied by the \( \# \)-sylvester monoid.

### 3.5. Baxter monoid.

**Lemma 3.17.** Let \( p, q, r, s \in \text{baxt} \) be such that \( \text{ev}(p) = \text{ev}(q) = \text{ev}(r) = \text{ev}(s) \). Then \( spr = sqr \).

**Proof.** Let \( t, u, v, w \in \mathcal{A}^* \) represent \( p, q, r, s \), respectively.

Let \( t' = wt \) and \( u' = uw \); note that \( \text{ev}(t') = \text{ev}(u') \). By Lemma 3.11 applied to the elements \( P_{\text{sylv}}(t'), P_{\text{sylv}}(u'), P_{\text{sylv}}(v) \), it follows that \( P_{\text{sylv}}(t'v) = P_{\text{sylv}}(u'v) \); thus \( P_{\text{sylv}}(wtv) = P_{\text{sylv}}(wuv) \).

Let \( t'' = tv \) and \( u'' = uw \); note that \( \text{ev}(t'') = \text{ev}(u'') \). By Lemma 3.15 applied to the elements \( P_{\text{sylv}}^\#(t''), P_{\text{sylv}}^\#(u''), P_{\text{sylv}}^\#(w) \), it follows that \( P_{\text{sylv}}^\#(wtv) = P_{\text{sylv}}^\#(wuv) \); thus \( P_{\text{sylv}}^\#(wtv) = P_{\text{sylv}}^\#(wuv) \).
Hence
\[ \text{spr} = P_{\text{baxt}}(wwv) = (P_{\text{sylv}}(wvt), P_{\text{sylv}}(wtv)) \]
\[ = (P_{\text{sylv}}(wuv), P_{\text{sylv}}(wwv)) = P_{\text{baxt}}(wwv) = sqp. \]

**Proposition 3.18.** The Baxter monoid satisfies the identities
\[ yxyxy = yxyxy \text{ and } xyxyx = xyxyx. \]

Furthermore, up to equivalence, these are the unique shortest non-trivial identities satisfied by the Baxter monoid.

**Proof.** Let \( x, y \in \text{baxt} \). Let \( p = r = xy \) and \( q = s = yx \). Then \( \text{ev}(p) = \text{ev}(q) = \text{ev}(r) = \text{ev}(s) \). Thus, by Lemma 3.17 \( yxyxy = \text{spr} = \text{sqr} = yxyxy \). The same reasoning with \( s = xy \) shows that \( xyxyx = yxyxy \).

The next step is to show that, up to equivalence, these are the only identities of length 6 satisfied by \( \text{baxt} \). So suppose that \( u = v \) is an identity of length 6 satisfied by \( \text{baxt} \), with \( u = u_1u_2 \cdots u_6 \) and \( v = v_1v_2 \cdots v_6 \), where \( u_i, v_i \in \{x, y\} \). Since the first symbol in a word \( w \) determines the root symbol of the left-hand tree in the pair \( P_{\text{baxt}}(u) \), and since the last symbol determines the root symbol of the right-hand tree, it follows that \( u_1 \) and \( v_1 \) must be the same variable, and that \( u_6 \) and \( v_6 \) must be the same variable. By Lemmata 3.10 and 3.14 \( u_2 \) and \( v_2 \) must be the same variable, and \( u_5 \) and \( v_5 \) must be the same variable. Since \( \text{sylv} \) and \( \text{sylv}^\# \) are both homomorphic images of \( \text{baxt} \) [Gir12, Proposition 3.7], the identity \( u = v \) is also satisfied by \( \text{sylv} \) and \( \text{sylv}^\# \). Since \( \text{sylv} \) is left-cancellative by Lemma 3.9 \( u_3u_4v_5u_6 = v_3v_4v_5v_6 \) is satisfied by \( \text{sylv} \) and is thus equivalent to \( xyxy = yxyx \). The aim is to characterize \( u = v \) up to equivalence, so assume that \( u_3u_4v_5u_6 = v_3v_4v_5v_6 \) actually is the identity \( xyxy = yxyx \).

Since \( \text{sylv}^\# \) is right-cancellative by Lemma 3.13 \( u_1u_2u_3u_4 = v_1v_2v_3v_4 \) is satisfied by \( \text{sylv}^\# \) and is thus equivalent to \( yyx = yxyx \), which is equivalent (by swapping the two sides) to \( xyxy = yxyx \) and (by interchanging \( x \) and \( y \)) to \( xyxy = xyxy \). Combining these with \( u_3u_4v_5u_6 = v_3v_4v_5v_6 \) from the previous paragraph shows that \( u = v \) is (up to equivalence) either \( xyxyx = yxyxy \) or \( xyxyx = yxyxy \).

Finally, it is necessary to show that no length-5 identity is satisfied by \( \text{baxt} \). Suppose \( u = v \) is an identity of length 5 satisfied by the Baxter monoid. Suppose that \( u = u_1u_2 \cdots u_5 \) and \( v = v_1v_2 \cdots v_5 \), where \( u_i, v_i \in \{x, y\} \). As before, considering the root symbols of the left-hand and right-hand trees in \( P_{\text{baxt}}(w) \) shows that \( u_1 \) and \( v_1 \) must be the same symbol, and that \( u_5 \) and \( v_5 \) must be the same symbol.

Since \( \text{sylv} \) and \( \text{sylv}^\# \) are both homomorphic images of \( \text{baxt} \) [Gir12, Proposition 3.7], both of them satisfy the identity \( u = v \). Furthermore, \( \text{sylv}^\# \) is right-cancellative and so satisfies the identity \( u_1u_2u_3u_4 = v_1v_2v_3v_4 \); while \( \text{sylv} \) is left-cancellative and so satisfies the identity \( u_2u_3u_4u_5 = v_2v_3v_4v_5 \). By Proposition 3.12, the unique length-4 identity satisfied by \( \text{sylv} \) is \( xyxy = yxyx \), so the identity \( u = v \) is either \( xyxyx = yxyxy \) or \( xyxyx = yxyxy \). Deleting the rightmost symbol \( y \) from each of these identities yields \( xyxy = xyxy \) and \( yxyx = yxyx \), one of which must be the identity \( u_1u_2u_3u_4 = v_1v_2v_3v_4 \) satisfied by \( \text{sylv}^\# \). This is a contradiction, since by Proposition 3.16 the only length-4 identity satisfied by \( \text{sylv}^\# \) is \( yxyx = yxyx \). \( \square \)
3.6. **Left and right patience sorting monoids.** The present authors and Silva [CMS § 4.2] have shown that the elements $P_{lPS}(21)$ and $P_{lPS}(1)$ generate a free submonoid of $lPS$. Since free monoids of rank at least 2 satisfy no non-trivial identities, it follows that $lPS$ does not satisfy a non-trivial identity. Furthermore, $lPS_n$ for $n \geq 2$ satisfies no non-trivial identities, while $lPS_1$, as a monogenic monoid, is of course commutative and satisfies the identity $xy = yx$.

For any $n$, the monoid $rPS_n$ satisfies the identity $(xy)^{n+1} = (xy)^n yx$ but does not satisfy any non-trivial identity of length less than or equal to $n$ [CMS § 4.2], which immediately implies that $rPS$ does not satisfy any non-trivial identity.

3.7. **Plactic monoid.** Recently, the present authors together with Klein, Kubat, and Okniński used the combinatorics of Young tableaux to prove that $plac_n$ does not satisfy any identity of length less than or equal to $n$ [CKK+ Proposition 3.1], which implies that $plac$ does not satisfy a non-trivial identity.

For finite-rank plactic monoids, some more partial results are known. First, $plac_1$ is monogenic and thus commutative and satisfies $xy = yx$. Kubat & Okniński [KO15] have shown that $plac_2$ satisfies Adian’s identity $x y y y x x y x x = x y y x x y y x x$, and that $plac_3$ satisfies the identity $pq qq pp = pp qq pp$, where $p(x, y)$ and $q(x, y)$ are respectively the left and right side of Adian’s identity (and so the identity $pq qq pp = pp qq pp$ has sixty variables $x$ or $y$ on each side). Furthermore, $plac_3$ does not satisfy Adian’s identity [KO15, p. 111–2].

Furthermore $plac_3$ satisfies the identity $pq pp q q = pp qq pp$, where as above $p(x, y)$ and $q(x, y)$ are respectively the left and right side of Adian’s identity [CKK+ Corollary 5.4]. The proof technique was rather different, via an embedding of $plac_3$ into the direct product of two copies of the monoid of $3 \times 3$ upper-triangular tropical matrices, which in turn is shown to satisfy the given identity.

**Conjecture 3.19.** For each $n \geq 4$, there is a non-trivial identity satisfied by $plac_n$.

**References**


Centro de Matemática e Aplicações, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829–516 Caparica, Portugal.

*Email address:* a.cain@fct.unl.pt

Departamento de Matemática & Centro de Matemática e Aplicações, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829–516 Caparica, Portugal.

*Email address:* ajm@fct.unl.pt