Journal of Algebra 630 (2023) 394–433

Contents lists available at ScienceDirect

Journal of Algebra

journal homepage: www.elsevier.com/locate/jalgebra

# An explicit algorithm for normal forms in small overlap monoids

James D. Mitchell, Maria Tsalakou *

*University of St Andrews, United Kingdom*

A R T I C L E   I N F O

*Article history:*
Received 25 May 2021
Available online 2 May 2023
Communicated by Derek Holt

*Keywords:*
Monoid
Semigroup
Normal forms
Finite presentation
Small overlap
Small cancellation

A B S T R A C T

We describe a practical algorithm for computing normal forms for semigroups and monoids with finite presentations satisfying so-called small overlap conditions. Small overlap conditions are natural conditions on the relations in a presentation, which were introduced by J. H. Remmers and subsequently studied extensively by M. Kambites. Presentations satisfying these conditions are ubiquitous; Kambites showed that a randomly chosen finite presentation satisfies the $C(4)$ condition with probability tending to 1 as the sum of the lengths of relation words tends to infinity. Kambites also showed that several key problems for finitely presented semigroups and monoids are tractable in $C(4)$ monoids: the word problem is solvable in $O(\min\{|u|, |v|\})$ time in the size of the input words $u$ and $v$; the uniform word problem for $\langle A|R\rangle$ is solvable in $O(N^2 \min\{|u|, |v|\})$ where $N$ is the sum of the lengths of the words in $R$; and a normal form for any given word $u$ can be found in $O(|u|)$ time. Although Kambites' algorithm for solving the word problem in $C(4)$ monoids is highly practical, it appears that the coefficients in the linear time algorithm for computing normal forms are too large in practice.

In this paper, we present an algorithm for computing normal forms in $C(4)$ monoids that has time complexity $O(|u|^2)$ for input word $u$, but where the coefficients are sufficiently small to allow for practical computation. Additionally, we show that

* Corresponding author.
*E-mail addresses:* jdm3@st-andrews.ac.uk (J.D. Mitchell), mt200@st-andrews.ac.uk (M. Tsalakou).

https://doi.org/10.1016/j.jalgebra.2023.04.019

the uniform word problem for small overlap monoids can be solved in $O(N \min\{|u|, |v|\})$ time.


## 1. Introduction

In this paper we present an explicit algorithm for computing normal forms of words in so-called small overlap monoids. The problem of finding normal forms for finitely presented monoids, semigroups, and groups, is classical, and is widely studied in the literature; some classical examples can be found in [2,18], and more recently in [9]. In a monoid $M$ defined by a presentation $\langle A \mid R\rangle$, the *word problem* asks if given $u, v \in A^*$, does there exist an algorithm deciding whether or not $u$ and $v$ represent the same element of $M$ (i.e. an algorithm which outputs “yes” if $(u, v)$ belongs to the least congruence on $A^*$ containing $R$ and “no” otherwise)? The word problem is said to be *decidable* if such an algorithm exists, and *undecidable* if it does not. If an algorithm for computing normal forms for a monoid presentation $\mathcal{P} = \langle A \mid R\rangle$ is available, then the word problem in the monoid defined by $\mathcal{P}$ is decidable by computing normal forms of $u$ and $v$, and checking if these two words coincide. In 1947, Markov [15] and Post [16] independently proved that the word problem for monoids is undecidable, in general, and, as such, the problem of finding normal forms for arbitrary finitely presented monoids is also undecidable.

Although undecidable in general, there are many special cases where it is possible to determine the structure of a finitely presented monoid, and to solve the word problem, and there are several well-known algorithms for doing this. Of course, because the word problem is undecidable in general, none of these algorithms can solve the word problem for all finitely presented monoids. The Todd-Coxeter Algorithm [19] terminates if and only if the input finite presentation defines a finite monoid. If it does terminate, the output of the Todd-Coxeter Algorithm is the (left or right) Cayley graph of the monoid defined by the input presentation. The word problem is thus solved by following the paths in the Cayley graph starting at the identity and labelled by any $u, v \in A^*$, and checking whether these paths end at the same node. The Knuth-Bendix Algorithm [12], can terminate even when the monoid $M$ defined by the input (again finite) presentation is infinite. The output of the Knuth-Bendix Algorithm is a finite noetherian complete rewriting system defining the same monoid $M$ as the input presentation. The normal forms of such a rewriting system are just the words to which no rewriting rule can be applied, and a normal form can be obtained for any word, by arbitrarily applying relations in the rewriting system until no further relations apply. This permits the word problem to be solved in monoids where the Knuth-Bendix Algorithm terminates via the computation of normal forms.

In this paper we are concerned with a class of finitely presented monoids, introduced by Remmers [17], and studied further by Kambites [8], [9], and [11].

If $\mathcal{P} = \langle A \mid R \rangle$ is a monoid presentation, then we will refer to the left or right hand side of any pair $(u, v) \in R$ as a *relation word*. A word $w \in A^*$ is said to be a *piece* of $\mathcal{P}$ if $w$ is a factor of at least two distinct relation words, or $w$ occurs more than once as a factor of a single relation word (possibly overlapping). Note that if a relation word $u$ appears as one side of more than one relation in the presentation, then $u$ is not considered a piece. A monoid presentation $\mathcal{P}$ is said to satisfy the condition $C(n)$, $n \in \mathbb{N}$, if the minimum number of pieces in any factorisation of a relation word is at least $n$. If no relation word in $\mathcal{P}$ equals the empty word, then $\mathcal{P}$ satisfies $C(1)$. If no relation word can be written as a product of pieces, then we say that $\mathcal{P}$ satisfies $C(n)$ for all $n \in \mathbb{N}$. If a presentation satisfies $C(n)$, then it also satisfies $C(k)$ for every $k \in \mathbb{N}$ such that $1 \leq k < n$.

For example, the presentation $\mathcal{P} = \langle a, b, c \mid abc = cba \rangle$ satisfies $C(3)$. The set of pieces is $P = \{\varepsilon, a, b, c\}$ and each relation word can be written as a product of exactly 3 pieces. Hence $\mathcal{P}$ does not satisfy $C(4)$. Similarly, for $\mathcal{P} = \left\langle a, b, c \mid acba = a^2bc \right\rangle$ the set of pieces is $P = \{\varepsilon, a, b, c\}$ and $\mathcal{P}$ satisfies $C(4)$ but not $C(5)$. If $\mathcal{P} = \left\langle a, b, c, d \mid acba = a^2bc, acba = db^3d \right\rangle$, the set of pieces is $P = \{\varepsilon, a, b, c, d, b^2\}$ and $\mathcal{P}$ satisfies $C(4)$ but not $C(5)$, since the relation words $acba$ and $a^2bc$ can be written as the product of 4 pieces. For the presentation $\mathcal{P} = \left\langle a, b, c, d \mid a^2bc = a^2bd \right\rangle$ the set of pieces is $P = \{\varepsilon, a, b, a^2, a^2b\}$ and none of the relation words can be written as a product of pieces since neither $c$ nor $d$ are pieces.

If a finite monoid presentation satisfies the condition $C(4)$, then we will refer to the monoid defined by the presentation as a *small overlap monoid*. Remmers initiated the study of $C(3)$ monoids in the paper [17]; see also [4, Chapter 5]. If a monoid presentation $\langle A \mid R \rangle$ satisfies $C(3)$, then the number of words in any class of $R^{\#}$ is finite, and so the monoid defined by the presentation is infinite; see [4, Corollary 5.2.16]. The word problem is solvable in $C(3)$ monoids but the algorithm described in [4, Theorem 5.2.15] has exponential complexity. Groups with similar combinatorial conditions have also been studied and such groups are called *small cancellation groups*. Small cancellation groups have decidable word problem; see [13, Chapter 5] for further details.

In [10], Kambites' showed that the probability that a randomly chosen finite monoid presentation is $C(4)$ tends to 1 as the length of the presentation tends to infinity; and the rate of convergence appears to be rather high; see Table 1. Hence, in some sense, algorithms for small overlap monoids are widely applicable. In Kambites [11], an explicit algorithm (**WpPrefix**) is presented for solving the word problem for finitely presented monoids satisfying $C(4)$. If $u, v \in A^*$, then, provided that certain properties of the presentation are known already, Kambites' Algorithm requires $O(\min\{|u|, |v|\})$ time. In Kambites [9], among many other results, it is shown that there exists a linear time algorithm for computing normal forms in $C(4)$ monoids, given a preprocessing step that requires polynomial time in the size of the alphabet and the maximum length of a relation word. The normal form algorithm from [9] is not stated explicitly in [9], and it appears that the constants in the polynomial time preprocessing step are rather large; see Section 4 for further details. The purpose of this paper is to provide an explicit algorithm for computing normal forms in $C(4)$ monoids with sufficiently small coefficients to

**Table 1**
The number of 2-generated 1-relation monoids with the $C(4)$ condition where the maximum length of a relation word is $n$. The values for $n \geq 14$ were obtained from a uniform sample of 1000 pairs of words $(l, r)$ of length where $|l| = n$ and $|r| \in \{1, \ldots, n\}$.

| $n$ | $C(4)$ monoids | monoids | ratio |
|---|---|---|---|
| 1 | 0 | 1 | 0.0 |
| 2 | 0 | 14 | 0.0 |
| 3 | 0 | 76 | 0.0 |
| 4 | 0 | 344 | 0.0 |
| 5 | 0 | 1,456 | 0.0 |
| 6 | 0 | 5,984 | 0.0 |
| 7 | 2 | 24,256 | 0.000082 |
| 8 | 26 | 97,664 | 0.000266 |
| 9 | 760 | 391,936 | 0.001939 |
| 10 | 17,382 | 1,570,304 | 0.011069 |
| 11 | 217,458 | 6,286,336 | 0.034592 |
| 12 | 1,994,874 | 25,155,584 | 0.079301 |
| 13 | 14,633,098 | 100,642,816 | 0.145396 |
| 14 | - | 4.026122e+08 | 0.186342 |
| 15 | - | 1.610531e+09 | 0.280811 |
| 16 | - | 6.442287e+09 | 0.374679 |
| 17 | - | 2.576948e+10 | 0.473369 |
| 18 | - | 1.030786e+11 | 0.594068 |
| 19 | - | 4.123155e+11 | 0.681053 |
| 20 | - | 1.649265e+12 | 0.732404 |
| 21 | - | 6.597065e+12 | 0.801495 |
| 22 | - | 2.638827e+13 | 0.843976 |
| 23 | - | 1.055531e+14 | 0.884619 |
| 24 | - | 4.222124e+14 | 0.929988 |
| 25 | - | 1.688850e+15 | 0.941666 |
| 26 | - | 6.755399e+15 | 0.963477 |
| 27 | - | 2.702160e+16 | 0.970104 |
| 28 | - | 1.080864e+17 | 0.977796 |
| 29 | - | 4.323456e+17 | 0.990216 |
| 30 | - | 1.729382e+18 | 0.989878 |
| 31 | - | 6.917529e+18 | 0.994861 |
| 32 | - | 2.767012e+19 | 0.995684 |
| 33 | - | 1.106805e+20 | 0.996879 |
| 34 | - | 4.427219e+20 | 0.998821 |
| 35 | - | 1.770887e+21 | 0.996402 |
| 36 | - | 7.083550e+21 | 0.999423 |
| 37 | - | 2.833420e+22 | 0.999776 |
| 38 | - | 1.133368e+23 | 0.997902 |
| 39 | - | 4.533472e+23 | 0.999928 |
| 40 | - | 1.813389e+24 | 0.99953 |
| 41 | - | 7.253555e+24 | 0.999982 |
| 42 | - | 2.901422e+25 | 0.999964 |
| 43 | - | 1.160569e+26 | 0.999986 |
| 44 | - | 4.642275e+26 | 0.99999 |
| 45 | - | 1.856910e+27 | 0.999972 |
| 46 | - | 7.427640e+27 | 1 |
| 47 | - | 2.971056e+28 | 1 |
| 48 | - | 1.188422e+29 | 0.999998 |
| 49 | - | 4.753690e+29 | 1 |
| 50 | - | 1.901476e+30 | 1 |

permit its practical use. If it is already known that the input presentation satisfies $C(4)$, and a certain decomposition of the relation words is known, then the time complexity of the algorithm we present is $O(|w|^2)$ for input word $w$, and the space complexity is the sum of $|A|$ and the lengths of all of the relation words in $R$. We will show that it is possible to show that the $C(4)$ condition holds, and that the required decomposition of the relation words can be found, in $O(N + n)$ time where $N$ is the sum of the lengths of the relation words, and $n$ is the number of relation words in Section 3.

In Section 2, we present some necessary background material, and establish some notation. In Section 3, we show that it is possible to determine the greatest $n \in \mathbb{N}$ such that a presentation $\mathcal{P}$ satisfies $C(n)$ in linear time in the sum of the lengths of the relation words in $\mathcal{P}$ using Ukkonen's Algorithm [20]. In Section 4, we discuss the normal form algorithm of Kambites from [9]. In Section 5, we describe, prove correct, and analyse the complexity of, a subroutine that is required in the practical normal form algorithm that is the main focus of this paper. Finally, in Section 6 we present our normal form algorithm, prove that it is correct, and analyse its complexity.

The algorithm for solving the word problem in $C(4)$ monoids given in [11], and the main algorithm from the present paper, were implemented by the authors in the C++ library libsemigroups [14].

## 2. Prerequisites

In this section we provide some of the prerequisites for understanding small overlap conditions and properties of small overlap monoids.

Let $A$ be a non-empty set, called an *alphabet*. A word $w$ over $A$ is a finite sequence $w = a_0a_1 \cdots a_m$, $m \geq 0$ of elements of $A$. The set of all words (including the empty word, denoted by $\varepsilon$) over $A$ with concatenation of words is called the *free monoid* on $A$ and is denoted by $A^*$. A *monoid presentation* is a pair $\langle A \,|\, R\rangle$ where $A$ is an alphabet and $R \subseteq A^* \times A^*$ is a set of *relations* on $A^*$. A monoid $M$ is defined by the presentation $\langle A \,|\, R\rangle$ if $M$ is isomorphic to $A^*/R^\#$ where $R^\# \subseteq A^* \times A^*$ is the least congruence on $A^*$ containing $R$. A *finitely presented monoid* is any monoid defined by a presentation $\langle A \,|R\rangle$ where $A$ and $R$ are finite, and such a presentation is called a *finite monoid presentation*. For the rest of the paper, $\mathcal{P} = \langle A \,|\, R\rangle$ will denote a finite monoid presentation where

$$R = \{(W_0, W_1), (W_2, W_3), \dots, (W_{n-2}, W_{n-1})\}.$$

If $s, t \in A^*$ are such that there exist $x_i, y_i \in A^*$ and $(W_i, W_{i+1})$ or $(W_{i+1}, W_i) \in R$ with $s = x_iW_iy_i$ and $t = x_iW_{i+1}y_i$, then we write $s \to t$. If there exists a sequence of words $s = w_0, w_1, \dots, w_n = t$ such that $w_i \to w_{i+1}$ for all $i \in \{0, \dots, n-1\}$, then we write $s \xrightarrow{*} t$ and we refer to such a sequence as a *rewrite sequence*. It is routine to verify that $(s, t) \in R^\#$ if and only if $s \xrightarrow{*} t$.

We say that a relation word $V$ is a *complement* of a relation word $W$ if there are relation words $V = r_0, r_1, \dots, r_{n-1} = W$ such that either $(r_i, r_{i+1}) \in R$ or $(r_{i+1}, r_i) \in R$ for $0 \leq i \leq n-1$. We say that a complement $V$ of $W$ is a *proper complement* of $W$ if $V \neq W$. The equivalence relation defined by the complements of the relation words is a subset of the congruence $R^\#$. We will write $u \equiv v$ to indicate that the words $u, v \in A^*$ represent the same element of the monoid presented by $\mathcal{P}$ (i.e. that $u/R^\# = v/R^\#$).

The *relation words* of the presentation $\mathcal{P}$ are $W_0, W_1, \dots, W_{n-1}$. A word $p \in A^*$ is called a *piece* if it occurs as a factor of $W_i$ and $W_j$ where $W_i \neq W_j$, or in two different places (possibly overlapping) in the same relation word in $R$. Note that the definition allows for the case when $i \neq j$ but $W_i = W_j$. In this case, neither $W_i$ nor $W_j$ is considered a piece (unless for other reasons), because although $W_i$ is a factor of $W_j$, it is not the case that $W_i \neq W_j$. By convention the empty word $\varepsilon$ is always a piece.

**Definition 2.1.1.** [cf. [8]] We say that a monoid presentation satisfies the condition $C(n)$, $n \in \mathbb{N}$, if no relation word can be written as the product of strictly less than $n$ pieces. The condition $C(1)$ describes those presentations where no relation word is equal to the empty word.

Having given the definition of the condition $C(4)$, we suppose for the remainder of the paper, that our fixed presentation $\mathcal{P} = \langle A|R\rangle$ satisfies the condition $C(4)$.

The following terms are central to the algorithms for $C(4)$ in [9,11] and are used extensively throughout the current paper.

We say that $s \in A^*$ is a *possible prefix* of a word $w \in A^*$ if $s$ is a prefix of some word $w_0 \in A^*$ such that $w \equiv w_0$. The *maximal piece prefix* of $u$ is the longest prefix of $u$ that is also a piece; we denote the maximal piece prefix of $u$ by $X_u$. The *maximal piece suffix* of $u$, $Z_u$, is the longest suffix of $u$ that is also a piece; denoted $Z_u$. The word $Y_u$ such that $u = X_uY_uZ_u$ is called the *middle word* of $u$. Since $u$ is a relation word of a presentation satisfying condition $C(4)$, $u$ cannot be written as a product of three pieces, and so the middle word $Y_u$ of $u$ cannot be a piece. In particular, the only relation word containing $Y_u$ as a factor is $u$.

Using the above notation every relation word $u$ in a $C(4)$ presentation can be written as a product of the form $X_uY_uZ_u$. Assume that $\overline{u}$ is a complement of $u$. Then $\overline{u} = X_{\overline{u}}Y_{\overline{u}}Z_{\overline{u}}$. We will write $\overline{X_u}$ instead of $X_{\overline{u}}$, $\overline{Y_u}$ instead of $Y_{\overline{u}}$, and $\overline{Z_u}$ instead of $Z_{\overline{u}}$. We say that $\overline{X_u}$ is a *complement* of $X_u$, $\overline{Y_u}$ is a complement of $Y_u$ and similarly for $Z_u$, $X_uY_u$ and $Y_uZ_u$.

A prefix of $w \in A^*$ that admits a factorization of the form $aXY$, for $XYZ$ a relation word, $a \in A^*$ and $X$ and $Y$ the maximal piece prefix and middle word of $XYZ$ respectively, is called a *relation prefix.* If $w \in A^*$ has relation prefixes $aXY$ and $a'X'Y'$ such that $|aXY| = |a'X'Y'|$ for some $a, a' \in A^*$, then $a = a'$, $X = X'$, and $Y = Y'$ as a direct consequence of the $C(4)$ condition. A relation prefix of the form $p = bX_0Y_0'X_1Y_1' \cdots X_{n-1}Y_{n-1}'X_nY_n$, $n \geq 1$ and $b \in A^*$, is called an *overlap prefix* if it satisfies the following:

(i) $Y_i'$ is a proper non-empty prefix of the middle word $Y_i$ of some relation word $X_iY_iZ_i$; and
(ii) there does not exist a factor in $p$ of the form $X_mY_m$ beginning before the end of $b$.

A relation prefix $aXY$ of a word $u$ is called a *clean relation prefix of* $u$ if $u$ does not have a prefix of the form $aXY'X_0Y_0$, where $Y'$ is a proper, non-empty prefix of $Y$. An overlap prefix of $u$ that is also a clean relation prefix is called a *clean overlap prefix of* $u$. If $p$ is a piece, then the word $u$ is called *$p$-active* if $pu$ has a relation prefix $aXY$ for some $a \in A^*$ such that $|a| < |p|$.

**Example 2.1.2.** Suppose that

$$\mathcal{P} = \left\langle a, b, c, d \,|\, a^2bc = acba,\ adca = bd^2b \right\rangle.$$

The relation words are $W_0 = a^2bc, W_1 = acba, W_2 = adca, W_3 = bd^2b$ and the set of pieces of this presentation is $P = \{\varepsilon, a, b, c, d\}$. Then $W_0$ is a proper complement of $W_1$, and $W_2$ is a proper complement of $W_3$. Since none of the relation words can be written as the product of less than 4 pieces, $\mathcal{P}$ is a $C(4)$ presentation. We have $X_{W_0} = a$, $Y_{W_0} = ab$, $Z_{W_0} = c$, $X_{W_1} = a$, $Y_{W_1} = cb$, $Z_{W_1} = a$, $X_{W_2} = a$, $Y_{W_2} = dc$, $Z_{W_2} = a$ and $X_{W_3} = b$, $Y_{W_3} = d^2$, $Z_{W_3} = b$.

Let $w = cba^2bd^2a$. The word $w$ has two relation prefixes: $cba^2b = cbX_{W_0}Y_{W_0}$ and $cba^2bd^2 = cba^2X_{W_3}Y_{W_3}$. The relation prefix $cbX_{W_0}Y_{W_0}$ is not clean since $w$ has a prefix of the form $cbX_{W_0}Y'_{W_0}X_{W_3}Y_{W_3}$, where $Y'_{W_0} = a$. In addition, $cbX_{W_0}Y'_{W_0}X_{W_3}Y_{W_3}$ is an overlap prefix since there is no factor of the form $X_{W_i}Y_{W_i}$ beginning before the end of $cb$. Let $p = a$. The word $w$ is $p$-active since $pw$ has the relation prefix $X_{W_1}Y_{W_1}$ and clearly $|\varepsilon| < |p|$.

The following results describe some properties of presentations satisfying the $C(4)$ condition mentioned in [8] as weak cancellativity properties.

**Proposition 2.1.3** *(Proposition 1 in [8]). Let $w$ be a word in $A^*$ and $aX_0Y'_0X_1Y'_1 \dots X_nY_n$ be an overlap prefix of $w$. Then there is no relation word contained in this prefix except possibly $X_nY_n$, in case $Z_n = \varepsilon$.*

In a word $w \in A^*$, an overlap prefix $aX_0Y'_0X_1Y'_1 \dots X_tY_t$ is always contained in some clean overlap prefix $aX_0Y'_0X_1Y'_1 \dots X_sY_s$ for $s \geq t$. In addition, if a word has a relation prefix, then the shortest relation prefix will be an overlap prefix. If a word $w$ contains a relation word $u$ as a factor, then it has a relation prefix of the form $aX_uY_u$ for some $a \in A^*$. It follows that it also has an overlap prefix, this is its shortest relation prefix. Since any overlap prefix is contained in a clean overlap prefix it also follows that $w$ has a clean overlap prefix. Hence, taking the contrapositive, if a word in $A^*$ does not have a clean overlap prefix, then it contains no relation words as factors.

If $aXYZ$ is a prefix of a word $w$, with $aXY$ an overlap prefix and $XYZ$ a relation word, then $aXY$ is a clean overlap prefix of $w$. If $aXY$ is a clean overlap prefix of a word $w$ and $\overline{XY}$ is a complement of $XY$, then $aXY$ and $a\overline{XY}$ are not necessarily clean overlap prefixes of words equivalent to $w$. However, it can be shown that such clean overlap prefixes are always overlap prefixes of words equivalent to $w$.

**Lemma 2.1.4** *(Lemma 2 in [8]). If a word $w \in A^*$ has clean overlap prefix $aXY$ and $w \equiv v$ for some $v \in A^*$, then $v$ either has $aXY$ or $a\overline{XY}$ for $\overline{XY}$ some complement of $XY$ as an overlap prefix; and no relation word in $v$ overlaps this prefix, unless it is $XYZ$ or $\overline{XYZ}$.*

In [11], Kambites describes an algorithm that takes as input two words and a piece of a given presentation that satisfies condition $C(4)$ and returns Yes if the words are equivalent and the piece is a possible prefix of the words and No if either of these does not happen. We will refer to this algorithm in the following sections of this paper and **WpPrefix**$(u, v, p)$ will be used to denote the result of this algorithm with input the words $u$ and $v$ and the piece $p$. In [11] it is shown that for a fixed $C(4)$ presentation, this algorithm decides whether $u$ and $v$ are equivalent and whether $p$ is a possible prefix of $u$ in time $O(\min(|u|, |v|))$ given the decomposition of relation words into the form $XYZ$ is known.

## 3. A linear time algorithm for the uniform word problem

In [8], Kambites explores the complexity of the so-called *uniform word problem* for $C(4)$ presentations. Given a finite monoid presentation $\langle A \mid R\rangle$ and two words in $A^*$, the uniform word problem asks whether the two words represent the same element of the monoid defined by $\langle A \mid R\rangle$. In particular, determining that the presentation satisfies the $C(4)$ condition is part of the uniform word problem for $C(4)$ presentations. In [8] it is shown that the uniform word problem for $C(4)$ presentations can be solved, in the RAM model of computation, in $O(|R|^2 \min(|u|, |v|))$ time for $u, v$ the two words in $A^*$ and $|R|$ the sum of the lengths of the distinct relation words in the presentation.

We will show that the uniform word problem for $C(4)$ presentations can be solved in $O(|R| \min(|u|, |v|))$ time, where $|R|$ is the sum of the lengths of the relation words, by using a generalized suffix tree to represent the relation words. In the RAM model, we may assume that the following operations are constant time: random access to the letters of a word $w \in A^*$ from an index; concatenation of words; comparison of letters from $A$ for a given total order on $A$.

Assume $A$ is an alphabet and $s = a_0a_1 \cdots a_{m-1} \in A^*$ such that $|s| = m$. We use the notation $s[i, j)$ for the factor $a_i \ldots a_{j-1}$ of $s$ that starts at position $i$ and ends at position $j - 1$ (inclusive). A word of length $m$ has $m$ non-empty suffixes $s[0, m), \ldots, s[m - 1, m)$. A word $x$ is a factor of $s$ if and only if it is a prefix of one of the suffixes of $s$.

**Definition 3.1.1.** [cf. Section 5.2 in [3]] A *suffix tree* for a word $s$ of length $m$ is a rooted directed tree with exactly $m$ leaf nodes numbered 0 to $m - 1$. The nodes of a suffix tree are of exactly one of the following types: the root; a leaf node; or an internal node. Each internal node has at least two children. Each edge of the tree is labelled by a nonempty factor of $s$ and no two edges leaving a node are labelled by words that begin with the same character. For any leaf $i$, $0 \le i < m$ the label of the path that starts at the root and ends at leaf $i$ is $s[i, m)$.

A *generalized suffix tree* is a suffix tree for a sequence of words $S = \{s_0, s_1, \ldots, s_{n-1}\}$. In a generalized suffix tree the leaf nodes are numbered by ordered pairs $(i, j)$ for $0 \le i \le n-1$ and $0 \le j \le |s_i|$. The label of the path that starts at the root node and ends at leaf $(i, j)$ is $s_i[j, m)$ for $m = |s_i|$. In a generalized suffix tree, a special unique character $\$_i$ is attached at the end of each word $s_i$ to ensure that each suffix corresponds to a unique leaf node in the tree. Thus a generalized suffix tree has exactly $N + n$ leaf nodes, where $N$ is the sum of the lengths of the words in $S$. See Fig. 1 for an example of generalized suffix tree.

A suffix tree for a word $w$ over an alphabet $A$ of length $m$ can be constructed in $O(m)$ time for constant-size alphabets and in $O(m \log |A|)$ time in the general case with the use of Ukkonen's algorithm [20]. If $N$ is the sum of the lengths of the words in the set of words $S$, then, a generalized suffix tree for $S$ can also be constructed in $O(N + n)$ time; see [3, Section 6.4] for further details.

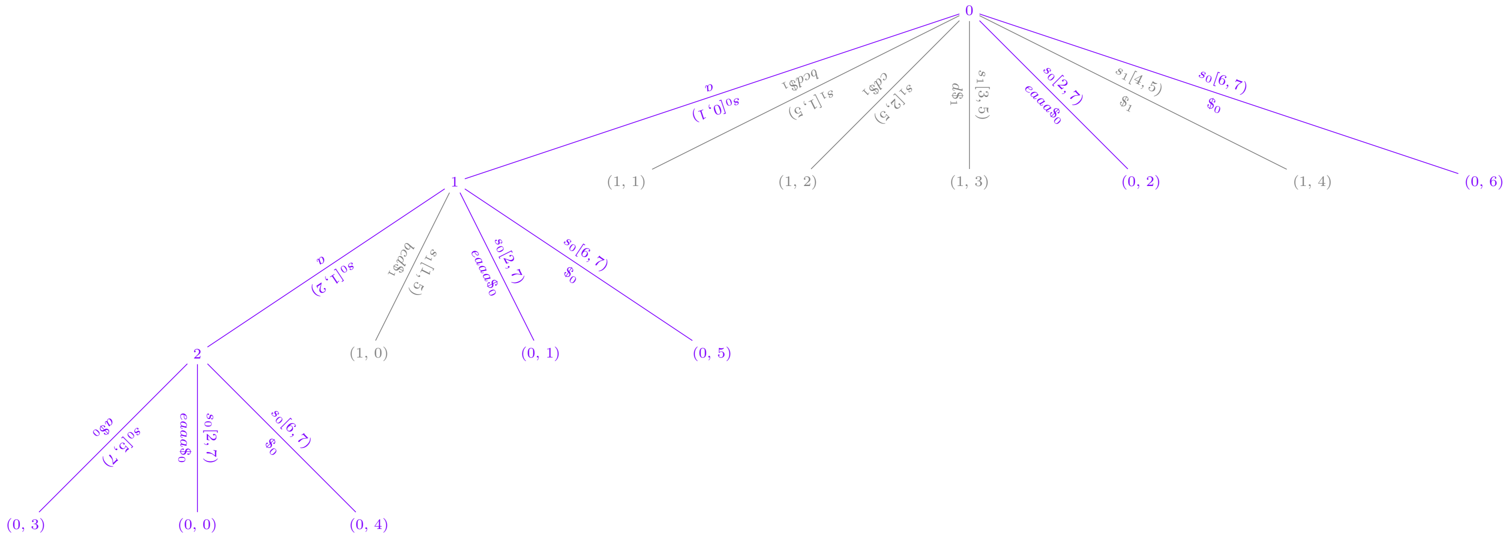


**Fig. 1.** The generalized suffix tree for the words $s_0 = a^2ea^3\$_0$ and $s_1 = abcd\$_1$. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

A generalized suffix tree for a set of words $S$ of total length $N$ has at most $2(N+n)$ nodes. By definition, such a suffix tree has exactly $N+n$ leaf nodes and one root. In addition, each internal node of the tree has at least two edges leaving it. These edges belong to paths that will eventually terminate at some leaf node. Hence there can exist at most $N+n-1$ internal nodes and the total number of nodes in a suffix tree is at most $2(N+n)$.

Generalized suffix trees can be constructed and queried in linear time to provide various information about a set of words. For example, for a sequence of $n$ words $S$ of total length $N$ we can find the longest subwords that appear in more than one word in $O(N+n)$ time, find the longest common prefix of two strings in $O(N+n)$ time, check if a word of length $m$ is a factor of some word in $S$ in $O(m)$ time; see Sections 2.7 to 2.9 in [3]. We are interested in utilizing generalized suffix trees in the study of $C(4)$ presentations. Since generalized suffix trees can be queried to find the longest subwords that appear in more than one word in the set of distinct relation words in $R$, we can use them to find maximal piece prefixes, for example. In order to do this, however, we need to build the generalized suffix tree of distinct relation words of a presentation since a word $p$ is a piece if it occurs as a factor of $W_i$ and $W_j$ where $W_i \neq W_j$, or in two different places (possibly overlapping) in the same relation word in $R$.

Given the set of relation words $R$ of a presentation we want to construct the generalized suffix tree of the set of distinct relation words in $R$ without altering the complexity of Ukkonen's algorithm. In practice, Ukkonen's algorithm for the generalized suffix tree of the set $S = \{s_0\$_0, s_1\$_1, \dots s_{n-1}\$_{n-1}\}$ starts by constructing the suffix tree $T$ for $s_0\$_0$. Then, for each $i \in \{1, \dots, n-1\}$, the edges and nodes that correspond to the word $s_i$ are added to $T$; see Section 6.4 in [3] for more detail. We denote this step of the procedure by $\texttt{AddWord}(T, s_i)$. In order to avoid adding the same word twice, we add the following step to the procedure for each $i \in \{1, \dots, n-1\}$: before calling $\texttt{AddWord}(T, s_i)$, we follow the path in $T$ that starts at the root node and is labelled by $s_i$. If this path ends at an internal node $\nu$ and one of the children of $\nu$ is a leaf node labelled by $(s_j, 0)$ for some $j$, then $s_j = s_i$ and hence we do not call $\texttt{AddWord}(T, s_i)$ for $s_i$. This additional step only requires traversing at most $|s_i| + 1$ nodes of $T$ for each $i \in \{1, \dots, n-1\}$. The total number of nodes of $T$ is bounded above by $2(N+n)$ and hence this step does not alter the complexity of the construction of the generalized suffix tree.

**Proposition 3.1.2.** *Let $\langle A \,|\, R\rangle$ be a finite monoid presentation such that the number of relation words in $R$ is $n$ for some $n \in \mathbb{N}$, and let $N$ be the sum of the lengths of the relation words in $R$. Then from the input presentation $\langle A \,|\, R\rangle$ the set of maximal piece prefixes, and suffixes, of $R$ can be computed in $O(N+n)$ time.*

**Proof.** Using Ukkonen's Algorithm, for example, a generalized suffix tree for the set $\{W_0\$_0, W_1\$_1, \dots, W_{n-1}\$_{n-1}\}$ of distinct relation words in $R$ can be constructed in $O(N+n)$ time. The maximal piece prefix of a relation word $W_r \in R$ can be found as follows. The path in the tree labelled by $W_r\$_r$ is followed from the root to the (unique) leaf node $(r, 0)$. Suppose that $v_0, v_1, \ldots, v_m$ are the nodes in the path from the root node $v_0$ to the leaf node $v_m = (r, 0)$. Then the maximal piece prefix of $u_r$ corresponds to the path $v_0, v_1, \ldots, v_{m-1}$. In other words, the maximal piece prefix of $W_r$ corresponds to the internal node that is the parent of the leaf node labelled $(r, 0)$. Hence the maximal piece prefix of each relation word can be determined in $O(|W_r|)$ time, and so every maximal piece prefix can be found in $O(N + n)$ time.

The maximal piece suffixes of the relation words can be found as follows. A generalized suffix tree for the set $\tilde{R}$ of reversals of the relation words in $R$ can be constructed in $O(N+n)$ time, and then used, as described above, to compute the maximal piece prefixes of the reversed relation words in $O(N + n)$ time also.

Alternatively, the generalized suffix tree for $R$ can be used to directly compute the maximal piece suffix of a given relation word $W_r$ by finding the maximum distance, from the root, of any internal node $n$ that is a parent of a leaf node labelled $(r, i)$ for any $i$. This maximum is the length of the maximal piece suffix of $W_r$. In this way, the maximum piece suffix of every relation word $W_r$ can be found in a single traversal of the nodes in the tree. Again, since there are $2(N + n)$ nodes in the tree, and the checks on each node can be performed in constant time, the maximal piece suffixes of all the relation words can be found in $O(N + n)$ time using this approach also. □

For example, the generalized suffix tree for the relation words in the presentation

$$\langle a, b, c, d, e \mid a^2ea^3 = abcd\rangle$$

can be seen in Fig. 1.

Using the approach described in the proof of Proposition 3.1.2, the path from the root 0 (corresponding to $a^2ea^3$) to the leaf node labelled by $(0, 0)$ consists of the root 0, internal nodes 1 and 2, and leaf node $(0, 0)$. Hence the maximal piece prefix of $a^2ea^3$ is $aa$, being the label of the path from the root to the parent 2 of the leaf node $(0, 0)$. Similarly, the maximal piece prefix of $abcd$ is $a$ corresponding to the parent 1 of the node $(1, 0)$. For the maximal piece suffix of $a^2ea^3$, the leaf nodes labelled $(0, i)$ for any $i$ with edge labelled by $\$_0$ are $(0, 4)$, $(0, 5)$, and $(0, 6)$. The parents of these nodes are 2, 1, and 0, respectively, and hence the maximal piece suffix of $a^2ea^3$ is $aa$. The only leaf node labelled $(1, i)$ and with edge labelled $\$_1$ is $(1, 4)$, and so the maximal piece suffix of $abcd$ is $\varepsilon$.

**Proposition 3.1.3.** *Let $\langle A \mid R\rangle$ be a finite monoid presentation such that the number of relation words in $R$ is $n$ for some $n \in \mathbb{N}$ and let $N$ be the sum of the lengths of the relation words in $R$. Then from the input presentation $\langle A \mid R\rangle$ it can be determined whether or not the presentation satisfies $C(4)$ in $O(N + n)$ time.*

**Proof.** In order to decide whether the presentation satisfies $C(4)$ we start by computing the maximal piece prefix $X_r$ and the maximal piece suffix $Z_r$ for each relation word $W_r$.

By Proposition 3.1.2, this step can be performed in $O(N+n)$ time. The presentation is $C(4)$ if for every relation word $W_r$, $|X_r|+|Z_r|<|W_r|$ and the middle word $Y_r$ is not a piece. It suffices to show that it can be determined in $O(N+n)$ time whether or not $Y_r$ is a piece for every $r$.

Any word $w \in A^*$ is a piece if and only if $w$ equals the longest prefix of $w$ that is a piece. In the proof of Proposition 3.1.2, we showed how to compute the maximal piece prefix of the relation words in $R$ using a generalized suffix tree in $O(N+n)$ time. The longest prefix of $Y_r$ that is a piece can be determined in $O(|Y_r|)$ time by finding the last internal node $\nu$ on the path from the root of the same generalized suffix tree labelled by $Y_r$; the node $\nu$ is the parent of the leaf node $(r, |X_r|)$. If $\nu$ is the root node, then the longest prefix of $Y_r$ that is a piece is $\varepsilon$. Otherwise, the longest prefix of $Y_r$ that is a piece is the label of the path from the root node to $\nu$. Hence determining whether or not every $Y_r$ is a piece can also be completed in total $O(N+n)$ time. □

If any of the words in $R$ is empty, then the presentation $\langle A \mid R\rangle$ is not $C(4)$. If all of the words are non-empty, then the number of relation words $n$ is bounded above by the sum of the lengths of the relation words $N$. Hence the $O(N+n)$ time complexity in Proposition 3.1.2 and Proposition 3.1.3 becomes $O(N)$.

The presentation $\langle a,b,c,d,e \mid a^2ea^3 = abcd\rangle$ can be seen to be $C(4)$ as follows. The only internal node on the path from the root of the suffix tree depicted in Fig. 1 labelled by $ea$ is the root itself. Hence the maximal piece prefix of $ea$ is $\varepsilon$ and so $ea$ is not a piece. Similarly, the only internal node on the path from the root labelled $bcd$ is the root itself, and so $bcd$ is not a piece either. Hence, by the proof of Proposition 3.1.3, the presentation $\langle a,b,c,d,e \mid a^2ea^3 = abcd\rangle$ is $C(4)$.

Proposition 3.1.3 allows us to prove the following theorem.

**Theorem 3.1.4.** *Let $\langle A \mid R\rangle$ be a finite monoid presentation such that the number of relation words in $R$ is $n$ for some $n \in \mathbb{N}$, let $N$ be the sum of the lengths of the relation words in $R$, and let $u, v \in A^*$ be arbitrary. Then the uniform word problem with input the presentation $\langle A \mid R\rangle$, and the words $u$ and $v$ can be solved in $O((N+n)\min(|u|,|v|))$ time.*

**Proof.** Given Proposition 3.1.3, the proof of this theorem is essentially identical to the proof of [8, Theorem 2]. □

## 4. Kambites' normal form algorithm

Let $A = \{a_0, a_1, \ldots, a_{n-1}\}$ be a finite alphabet and define a total order $<$ on the elements of $A$ by $a_0 < a_1 < \cdots < a_{n-1}$. We extend this to a total order over $A^*$, called the *lexicographic order* as follows. The empty word $\varepsilon$ is less than every other word in $A^*$. If $u = a_iu_0$ and $v = a_jv_0$ are words in $A^+$, $a_i, a_j \in A$ and $u_0, v_0 \in A^*$, then $u < v$ whenever $a_i < a_j$, or $a_i = a_j$ and $u_0 < v_0$.

As mentioned above, Kambites in [11] described an algorithm for testing the equivalence of words in $C(4)$ monoids. In [9] it was shown that given a monoid $M$ defined by a $C(4)$ presentation $\langle A \mid R\rangle$ and a word $w \in A^*$ there exists an algorithm that computes the minimum representative of the equivalence class of $w$ with respect to the lexicographic order on $A^*$. This minimum representative is also known as the *normal form* of $w$. It is not, perhaps, immediately obvious that such a minimal representative exists, because the lexicographic order is not a well order (it is not true that every non-empty subset of $A^*$ has a lexicographic least element). However, every equivalence class of a word in a $C(3)$ monoid is finite; see, for example, in [4, Corollary 5.2.16]. Since any presentation that satisfies $C(4)$ also satisfies $C(3)$, there exists a lexicographically minimal representative for any $w \in A^*$. We denote the lexicographically minimal word equivalent to $w \in A^*$ by $\min w$.

We note that since all of the equivalence classes of a $C(3)$ monoid are finite, any monoid satisfying $C(n)$ for $n \geq 3$, is infinite. Conversely, if $\langle A \mid R\rangle$ is a presentation for a finite monoid and this presentation satisfies $C(n)$ for some $n \in \mathbb{N}$, then $n \in \{1, 2\}$.

In [9] Kambites proved the following result.

**Proposition 4.1.1** *(Corollary 3 in [9]). Let $\langle A \mid R\rangle$ be a finite monoid presentation satisfying $C(4)$ and suppose that $A$ is equipped with a total order. Then there exists an algorithm which, given a word $w \in A^*$, computes in $O(|w|)$ time the corresponding lexicographic normal form for $w$.*

Although Proposition 4.1.1 asserts the existence of an algorithm for computing normal forms, this algorithm is not explicitly stated in [9]. In the following paragraphs we briefly discuss the algorithm arising from [9].

We require a number of definitions; see [1] for further details. A *transducer* $\mathcal{T} = \langle A, B, Q, q\_, Q_+, E\rangle$ is a 6-tuple that consists of an input alphabet $A$, an output alphabet $B$, a finite set of states $Q$, an initial state $q\_$, a set of terminal states $Q_+$ that is a subset of $Q$, and a finite set of transitions or edges $E$ such that $E \subset Q \times A^* \times B^* \times Q$. A pair $(u, v) \in A^* \times B^*$ is *accepted* by the transducer if there exist transitions $(q\_, u_0, v_0, q_1), (q_1, u_1, v_1, q_2), \ldots, (q_{n-1}, u_{n-1}, v_{n-1}, q_+) \in E$ such that $q_+ \in Q_+$ and $u = u_0u_1 \cdots u_{n-1}$, $v = v_0v_1 \cdots v_{n-1}$. The *relation* accepted by $\mathcal{T}$ is the set of all pairs accepted by $\mathcal{T}$. A relation accepted by a transducer is called a *rational relation.* A rational relation that contains a single pair $(u, v)$ for each $u \in A^*$ is called a *rational function.*

A *deterministic 2-tape finite automaton* is an 8-tuple $\mathcal{A} = \langle A, B, Q_1, Q_2, q\_, Q_+, \delta_1, \delta_2\rangle$ that consists of the tape-one alphabet $A$, the tape-two alphabet $B$, two disjoint state sets $Q_1$ and $Q_2$, an initial state $q\_ \in Q_1 \cup Q_2$, a set of terminal states $Q_+ \subset Q_1 \cup Q_2$ and two partial functions $\delta_1 : Q_1 \times A \cup \{\$\} \to Q_1 \cup Q_2$, $\delta_2 : Q_2 \times B \cup \{\$\} \to Q_1 \cup Q_2$ where \$ is a symbol not in $A$ and $B$. A path of length $n$ in $\mathcal{A}$ is a sequence of transitions of the form

$$(q_0, t_0, a_0, \delta_{t_0}(q_0, a_0))(\delta_{t_0}(q_0, a_0), t_1, a_1, \delta_{t_1}(q_1, a_1)) \cdots$$
$$(\delta_{t_{n-2}}(q_{n-2}, a_{n-2}), t_{n-1}, a_{n-1}, \delta_{t_{n-1}}(q_{n-1}, a_{n-1})),$$

where $t_i \in \{1,2\}$, $q_i \in Q_{t_i}$, $a_i \in A$ if $t_i = 1$ and $a_i \in B$ if $t_i = 2$ for $0 \leq i \leq n-1$ and $\delta_{t_i}(q_i, a_i) = q_{i+1}$ for $0 \leq i < n-1$. A path is called *successful* if $q_0 = q\_$ and $\delta_{t_{n-1}}(q_{n-1}, a_{n-1}) \in Q_+$. A pair $(u, v) \in A^* \times B^*$ is the label of a path if $u$ is the concatenation of all the letters $a_i$, $0 \leq i \leq n-1$ that belong in $A$ and $v$ is the concatenation of all the letters $a_i$, $0 \leq i \leq n-1$ that belong in $B$. A pair $(u, v) \in A^* \times B^*$ is *accepted* by $\mathcal{A}$ if it labels a successful path in $\mathcal{A}$. The *relation accepted by* $\mathcal{A}$ is the set of all pairs accepted by $\mathcal{A}$.

Let $\text{lex}(R^{\#}) = \{(u, v) \in A^* \times A^* \mid u \equiv v \text{ and } v \text{ be a lexicographic normal form}\}$. In [9] it is shown that $R^{\#}$ is a rational relation and $\text{lex}(R^{\#})$ is a rational function. According to Lemma 5.3 in [7], $\text{lex}(R^{\#})$ can effectively be computed from a finite transducer for $R^{\#}$. Kambites describes the construction of a finite transducer for $\text{lex}(R^{\#})$ in [9]. The steps for constructing this transducer are the following:

- Starting from the $C(4)$ presentation $\langle A \mid R\rangle$, an abstract machine called a 2-tape deterministic prefix rewriting automaton with bounded expansion can be computed. The construction is given in the proof of Theorem 2 in [9]. The relation accepted by this automaton is $R^{\#}$.
- Using the construction in the proof of Theorem 1 in [9], the 2-tape deterministic prefix rewriting automaton can be used to construct a transducer $\mathcal{T}$ realizing $R^{\#}$.

Let $\delta$ be the length of the longest relation word in $R$ and let $P$ be the set of pieces of the presentation. The set $A^{\leq k}$ for $k \in \mathbb{N}$ consists of all words in $A^*$ with length less or equal to $k$. Similarly, $A^{<k}$ consists of all words in $A^*$ with length less than $k$. In addition, let \$ be a new symbol not in $A$. The set $A^{<k}\$$ consists of words $u\$$ such that $u \in A^{<k}$.

The state set of the transducer $\mathcal{T}$, given in the proof of Theorem 1 in [9], is the set $C \times C \times P$ where $C$ is the set

$$A^{\leq 3\delta} \cup A^{<3\delta}\$.$$

Hence, the number of states of the transducer is extremely large even for relatively small presentations.

For example, let $\langle a, b, c \mid a^2bc = acba\rangle$ be the presentation. In this case $|A| = 3, \delta = 4$ and $P = \{\varepsilon, a, b, c\}$. The size of the state set $Q = C \times C \times P$ of the corresponding transducer is $|C|^2 \cdot |P| = 4|C|^2$. Since $C = A^{\leq 12} \cup A^{<12}\$$,

$$|C| = \sum_{i=0}^{12} 3^i + \sum_{i=0}^{11} 3^i = 1062881$$

and $|Q| = 4518864080644$.

Another approach arising from [9] for the computation of normal forms is the construction of a deterministic 2-tape automaton accepting $\text{lex}(R^{\#})$. This also begins by constructing the transducer $\mathcal{T}$. The process arising from [9] for the construction of the automaton is: perform the two steps given above to construct the transducer $\mathcal{T}$, then:

- using the construction in the proof of Proposition 1 in [9], a deterministic 2-tape automaton accepting $R^{\#}$ can be constructed starting from the transducer $\mathcal{T}$;
- the proof of Theorem 5.1 in [6] describes the construction of a deterministic 2-tape automaton accepting $\text{lex}(R^{\#})$, starting from the deterministic 2-tape automaton that accepts $R^{\#}$.

The state set $Q = Q_1 \cup Q_2$ of the 2-tape automaton that accepts $R^{\#}$ in the second step is the same as the state set of the transducer $\mathcal{T}$, partitioned in two disjoint sets $Q_1$ and $Q_2$. The state set $Q' = Q'_1 \cup Q'_2$ of the 2-tape automaton that accepts $\text{lex}(R^{\#})$ is the union of the sets $Q'_1 = Q_1 \times 2^{Q_1}$ and $Q'_2 = Q_2 \times 2^{Q_1}$, hence the number of states of this automaton is greater than the number of states of the transducer $\mathcal{T}$.

Although the approach described in [9] allows normal forms for words in a $C(4)$ presentation to be found in linear time, it is impractical to use a transducer with such a large state set. The current article arose out of a desire to have a practical algorithm for computing normal forms in $C(4)$ monoids.

## 5. Possible prefix algorithm

Before describing the procedure for finding normal forms, we describe an algorithm that takes as input a word $w_0$ and a possible prefix piece $p$ of $w_0$ and returns a word equivalent to $w_0$ with prefix $p$. As mentioned above the algorithm **WpPrefix**, described in [11] can decide whether a piece $p$ is a possible prefix of some word $w_0$ by calling **WpPrefix**$(w_0, w_0, p)$.

**Algorithm 1 - ReplacePrefix**$(w_0, p)$.

**Input:** A word $w_0$ and a piece $p$ such that **WpPrefix**$(w_0, w_0, p)$ =Yes.
**Output:** A word equivalent to $w_0$ with prefix $p$.
**ReplacePrefix**$(w_0, p)$:

**if** $w_0$ does not have prefix $p$ and $w_0 = aXYw'$ with $aXY$ a clean overlap prefix **then**
  $u \leftarrow$**ReplacePrefix**$(w', Z)$ with $Z$ deleted
  $w_0 \leftarrow a\overline{XYZ}u$ where $\overline{XYZ}$ is a proper complement of $XYZ$ such that $p$ is a prefix of $a\overline{X}$
**end if**
**return** $w_0$

**Lemma 5.1.1.** *Let $w \in A^*$ be arbitrary. If there exists a piece $p$ that is a possible, but not an actual, prefix of $w$, then the shortest relation prefix of $w$ is a clean overlap prefix.*

**Proof.** Since $p$ is a possible prefix but not a prefix of $w$, $w$ contains at least one relation word and hence has a relation prefix. Let $aXY$ be the shortest relation prefix of $w$. Then $aXY$ is an overlap prefix. If $aXY$ is not clean, then $w$ has a prefix of the form $aXY'X_0Y_0$ such that $Y'$ is a proper non-empty prefix of $Y$. Hence the shortest clean overlap prefix of $w$ contains $aXY'$ and hence $aXY'$ is also a prefix of every $v$ such that $v \equiv w$ by Lemma 2.1.4. Let $w_0$ be a word equivalent to $w$ that has prefix $p$. Then either $p$ is a prefix of $aXY'$ or $p$ contains $aXY'$. In the former case this would mean that $p$ is also a prefix of $w$, which is a contradiction. In the latter case $XY'$ is a factor of $p$. Since $p$ is a piece this implies that $XY'$ is also a piece which is a contradiction since $X$ is the maximal piece prefix of the relation word $XYZ$. We have shown that the shortest relation prefix of $w$ is a clean overlap prefix, as required. □

**Lemma 5.1.2.** *Let $w \in A^*$ be arbitrary. If $w$ has a piece $p$ as a possible, but not an actual, prefix, then the shortest relation prefix of $w$ can be found in constant time, given the suffix tree for the relation words in $R$.*

**Proof.** Suppose that $S = \{W_0, W_1, \ldots, W_{n-1}\}$ is the set of relation words and let $\delta$ be the length of the longest relation word in $R$. We want to find the shortest relation prefix $tX_{W_i}Y_{W_i}$ for some $t \in A^*$, and $W_i \in S$. Since $X_{W_i}Y_{W_i}$ and $p$ are factors of relation words, $|p|, |X_{W_i}Y_{W_i}| \leq \delta$. Since $tX_{W_i}Y_{W_i}$ is the shortest relation prefix of $w$, $t$ is prefix of every word equivalent to $w$, and hence $t$ is a proper prefix of $p$. In particular, $|t| < |p| < \delta$. If $|w| \geq 2\delta$, then we define $v$ to be the prefix of $w$ of length $2\delta$; otherwise, we define $v$ to be $w$. In order to find the shortest relation prefix of $w$ it suffices to find the shortest relation prefix of $v$. For a given presentation, the length of $v$ is bounded above by the constant value $2\delta$.

In practice, in order to find the shortest relation prefix of $v$, we construct a suffix tree for all words $X_{W_i}Y_{W_i}$ such that $X_{W_i}Y_{W_i}Z_{W_i}$ is a relation word of the presentation. This is done in $O(N + n)$ time, for $N$ the sum of the lengths of the relation words in the presentation. A factor of $v$ has the form $X_{W_i}Y_{W_i}$ for some $i$ if and only if this factor labels a path that starts at the root node of the tree and ends at some leaf node labelled by $(i, 0)$. Hence the shortest relation prefix of $v$ can be found by traversing the nodes of the tree at most $|v|$ times. Since the length of $v$ is at most $2\delta$ this can be achieved in constant time.

The complexity of this procedure is $O((N + n)|v|) = O(2\delta(N + n))$ which is independent of the choice of $w$. □

Next, we will show that Algorithm 1 is valid.

**Proposition 5.1.3.** *If $w_0, p \in A^*$ are such that $p$ is piece and a possible prefix of $w_0$, then **ReplacePrefix(**$w_0, p$**)** returns a word that is equivalent to $w_0$ and has prefix $p$ in $O(|w_0|)$ time, given the suffix tree for the relation words in $R$.*

**Proof.** We will prove that the algorithm returns the correct result using induction on the number $k$ of recursive calls in line 2. Note that if $p$ is a possible prefix of $w_0$ and $w_0$ contains no relation words, then $p$ is a prefix of $w_0$. On the other hand, if $p$ is not a prefix of $w_0$, then $w_0$ must contain a relation word, and hence a clean overlap prefix.

We first consider the base case, when $k = 0$. Let $p$ be a piece and $w_0$ a word such that **ReplacePrefix**$(w_0, p)$ terminates without making a recursive call. This only happens in case $p$ is already a prefix of $w_0$ and the algorithm returns $w_0$ in line 5. Hence when $k = 0$ the word returned by **ReplacePrefix**$(w_0, p)$ is $w_0$ and has prefix $p$.

Next, we let $k > 0$ and assume that the algorithm returns the correct result when termination occurs after strictly fewer than $k$ recursive calls. Now let $p$ be a piece and $w_0$ a word such that **ReplacePrefix**$(w_0, p)$ terminates after $k$ recursive calls. It suffices to prove that the first recursive call returns the correct output.

If $p$ is already a prefix of $w_0$ a recursive call does not happen, hence we are in the case where $p$ is not a prefix of $w_0$. Since $p$ is a possible prefix of $w_0$, there exists a word that is equivalent but not equal to $w_0$ and that has $p$ as a prefix. This means that $w_0$ has a relation prefix and hence it has a clean overlap prefix of the form $aXY$. By Lemma 2.1.4, every word equivalent to $w_0$ has $a\overline{XY}$ for $\overline{XY}$ a complement of $XY$, as a prefix. Hence since $p$ is not a prefix of $w_0$, $p$ must be a prefix of $a\overline{XY}$, for $\overline{XY}$ a proper complement of $XY$. Since $p$ is a piece, $|p| \leq |a\overline{X}|$ because otherwise a prefix of $\overline{XY}$ longer than $\overline{X}$ would be a piece. Hence $p$ is a prefix of $a\overline{X}$. It follows that there exists a word equivalent to $w_0$ in which $aXY$ is followed by $Z$ and we can rewrite $XYZ$ to $\overline{XYZ}$. This implies that if $w'$ is the suffix of $w_0$ following $aXY$, then $Z$ is a possible prefix of $w'$. In particular, by the inductive hypothesis, **ReplacePrefix**$(w', Z)$ is $Zu$ for some $u \in A^*$ and $a\overline{XYZ}u$ is a word equivalent to $w_0$ that has prefix $p$. Therefore, by induction, the algorithm will return $a\overline{XYZ}u$ in line 5 after making the recursive call in line 2.

It remains to show that the output of **ReplacePrefix**$(w_0, p)$ can be computed in $O(|w_0|)$ time. The recursive calls within **ReplacePrefix**$(w_0, p)$ always have argument which is a factor, even a suffix, of $w_0$. Hence if **WpPrefix**$(w_0, w_0, p)$=Yes, then $p$ is a possible prefix of $w_0$, and the number of recursive calls in Algorithm 1 is bounded above by the length of $w_0$.

Let $\delta$ be the length of the longest relation word of our presentation. In line 1, we begin by checking if $p$ is a prefix of $w_0$. Clearly, this can be done in $|p|$ steps and since $p$ is a piece, $|p| < \delta$. In line 1, we also search for the clean overlap prefix of $w_0$. As shown in Lemmas 5.1.1 and 5.1.2, this can be done in constant time. Next, in line 2 we delete a prefix of length $|Z|$ from the output of **ReplacePrefix**$(w', Z)$. Since $|Z| < \delta$, the complexity of this step is also constant for a given presentation. The search for a complement $\overline{XYZ}$ of $XYZ$ such that $p$ is a prefix of $a\overline{X}$ can be performed in constant time since the number of relation words is constant for a given presentation and $|p| < \delta$. In line 3, we concatenate words to obtain a word equivalent to $w$. In every recursive call we concatenate three words hence the complexity of this step is also constant. As we have already seen, the number of recursive calls of the algorithm is bounded above by the length of $w_0$, hence **ReplacePrefix**$(w_0, p)$ can be computed in $O(|w_0|)$ time. □

For the following examples we will use the notation $w_i$ and $p_i$ for the parameters of the $i$th recursive call of **ReplacePrefix**$(w, p)$ and we let $w_0 = w$ and $p_0 = p$.

**Example 5.1.4.** Let $\mathcal{P}$ be the presentation

$$\langle a, b, c, d \mid acba = a^2bc,\ acba = db^2d \rangle$$

and we let $w = acbdb^2d$. The set of pieces of $\mathcal{P}$ is $P = \{\varepsilon, a, b, c, d\}$. Let $W_0 = acba$, $W_1 = a^2bc$, $W_2 = db^2d$. Clearly $X_{W_0} = a$, $Y_{W_0} = cb$, $Z_{W_0} = a$, $X_{W_1} = a$, $Y_{W_1} = ab$, $Z_{W_1} = c$ and $X_{W_2} = d$, $Y_{W_2} = b^2$, $Z_{W_2} = d$. The algorithm **WpPrefix**$(w, w, d)$ returns Yes and we want to find **ReplacePrefix**$(w, d)$.

We begin with $w_0 = acbdb^2d$, $p_0 = d$ and $u_0 = \varepsilon$. Clearly $w_0$ does not begin with $d$ but using the process described in Lemma 5.1.2, we can find the clean overlap prefix of $w_0$ which is $acb = X_{W_0}Y_{W_0}$ and hence $w_0$ satisfies the conditions of line 1. In line 2, $w_1 \leftarrow db^2d$, $p_1 \leftarrow a$ and in order to compute $u_1$ we need to compute **ReplacePrefix**$(db^2d, a)$. Since $w_1$ does not begin with $a$ we need to find the clean overlap prefix of $w_1$ which is $db^2 = X_{W_2}Y_{W_2}$. Now $w_2 = d$, $p_2 = d$ and **ReplacePrefix**$(d, d)$ returns $d$. Now $w_1$ will be rewritten to a complement of $db^2d$ that begins with $a$, hence we choose one of $W_0$ and $W_1$. If we choose $W_0$, $w_1 \leftarrow acba$ and $w_0 \leftarrow db^2dcba$. If we choose $W_1$, $w_1 \leftarrow a^2bc$ and $w_0 \leftarrow db^2dabc$. In both cases the algorithm returns a word equivalent to $w$ that begins with $d$.

## 6. A practical normal form algorithm

In this section we describe a practical algorithm for computing lexicographically normal forms in $C(4)$ monoids. This section has four subsections: the first contains a description of the algorithm; the second a proof that the algorithm returns a word equivalent to the input word; the third contains a proof that the algorithm returns the lexicographically least word equivalent to the input word; and in the final section we consider the complexity of the algorithm.

### 6.1. Statement of the algorithm

In this section, we describe the main algorithm of this paper for computing the lexicographically least word equivalent to an input word. Roughly speaking, the input word is read from left to right, clean overlap prefixes of the form $uXY$ for $u \in A^*$ are found and replaced with a lexicographically smaller word if possible. Subsequently, the next clean overlap prefix of this form after $uXY$ is found, and the process is repeated. The algorithm is formally defined in Algorithm 2.

**Algorithm 2 - NormalForm**$(w_0)$.

**Input:** A word $w_0 \in A^*$.
**Output:** The lexicographically least word $v \in A^*$ such that $v \equiv w_0$.

1: $W \leftarrow \varepsilon$, $v \leftarrow \varepsilon$, $w \leftarrow w_0$
2: **while** $w \neq \varepsilon$ **do**
3:   **if** $W = X_rY_rZ_r$, $w = Z_rw'$, $w'$ is $\overline{Z_r}$-active for some proper complement $\overline{Z_r}$ of $Z_r$, $w'$ is not $Z_r$-active and $a$ is a suffix of $\overline{Z_r}$ with $aw' = X_sY_sw''$ and **WpPrefix**$(w'', w'', Z_s)$=Yes **then**
4:     **if** there exists a proper complement of $X_sY_sZ_s$ with prefix $a$ that is lexicographically less than $X_sY_sZ_s$ **then**
5:       $X_tY_tZ_t \leftarrow$ the lexicographically minimal proper complement of $X_sY_sZ_s$ that has prefix $a$
6:     **else**
7:       $X_tY_tZ_t \leftarrow \varepsilon$
8:     **end if**
9:     **if** $X_tY_tZ_t \neq \varepsilon$, $X_t = ab$ and **WpPrefix**$(w_0, vZ_rbY_tZ_tt, \varepsilon)$ = Yes where $Z_st$ =**ReplacePrefix**$(w'', Z_s)$ **then**
10:       $W \leftarrow X_tY_tZ_t$
11:       $v \leftarrow vZ_rbY_t$
12:       $w \leftarrow Z_tt$
13:     **else**
14:       $W \leftarrow X_sY_sZ_s$
15:       $v \leftarrow vZ_rX''_sY_s$ where $X_s = aX''_s$
16:       $w \leftarrow$ **ReplacePrefix**$(w'', Z_s)$
17:     **end if**
18:   **else if** $w$ has a clean overlap prefix of the form $aXY$ and $w = aXYw'$ **then**
19:     **if** **WpPrefix**$(w', w', Z)$ = No **then**
20:       $W \leftarrow \varepsilon$
21:       $v \leftarrow vaXY$
22:       $w \leftarrow w'$
23:     **else**
24:       $W \leftarrow$ the lexicographically minimal complement $X'Y'Z'$ of $XYZ$
25:       $v \leftarrow vaX'Y'$
26:       $w \leftarrow Z'w''$, where **ReplacePrefix**$(w', Z) = Zw''$
27:     **end if**
28:   **else**
29:     $v \leftarrow vw$
30:     $w \leftarrow \varepsilon$
31:   **end if**
32: **end while**
33: **return** $v$

### *6.2. Equivalence*

In this section we show that **NormalForm** terminates and the word returned is equivalent to the input word $w_0$. We begin by observing that **NormalForm** rewrites $v$ and $w$ in lines 11-12, 15-16, 21-22, 25-26, and 29-30. For the remainder of this section, $v_i$ and $w_i$ will be $v$ and $w$ after the $i$-th time the algorithm has rewritten $v$ and $w$.

The following result will be used to prove that Algorithm 2 terminates and that the word returned by the algorithm is equivalent to its input. We have already proved that if a piece $p$ is a possible prefix of a word $v$, then algorithm **ReplacePrefix**$(v, p)$ returns a word equivalent to $v$ with prefix $p$. If $w \in A^*$, $XY$ is a clean overlap prefix of $w$, $w'$ is the suffix of $w$ following $XY$, and $Z$ is a possible prefix of $w'$, then $w \equiv XYZu$ where $Zu = \mathbf{ReplacePrefix}(w', Z)$. This is straightforward since $Z$ is a possible prefix of $w'$ and hence $w' \equiv Zu$ for $Zu = \mathbf{ReplacePrefix}(w', Z)$.

**Lemma 6.2.1.** *Assume that $w_0 \in A^*$ is the input to **NormalForm**. Then at each step of **NormalForm**$(w_0)$, $v_i w_i \equiv w_0$.*

**Proof.** We proceed by induction on $i$. For $i = 0$, $v_0 = \varepsilon$ and hence $v_0 w_0 = w_0$. Let $k \in \mathbb{N}$ and assume that $v_k w_k \equiv w_0$. We will prove that $v_{k+1} w_{k+1} \equiv w_0$.

In the cases of lines 21-22 and 29-30, it is clear that some prefix of $w_k$ is transferred to the end of $v_{k+1}$. In particular, $v_k w_k = v_{k+1} w_{k+1} \equiv w_0$. In lines 15-16, a prefix of $w_k$ is transferred to the end of $v_{k+1}$ again and Algorithm 1 is applied. Hence $v_k w_k \equiv v_{k+1} w_{k+1} \equiv w_0$. In lines 25-26 we rewrite the relation word $XYZ$ to $\overline{XYZ}$. Since $XYZ$ begins after the beginning of $w_k$, there exists some $s$ equivalent to $w_k$ which is obtained by the application of this rewrite. Hence $v_k w_k \equiv v_k s$ with $a\overline{XY}$ being a prefix of $s$, and $v_k s = v_{k+1} w_{k+1}$. It follows that $v_{k+1} w_{k+1} \equiv w_0$. Finally, in the case of lines 11-12 the result follows immediately from the use of **WpPrefix** in line 9. □

In [4, Theorem 5.2.14] it is shown that if $w_0, w \in A^*$ are such that $w \equiv w_0$, then

$$|w| < \delta |w_0|$$

where $\delta$ is the maximum length of a relation word in $R$. Since in every step of **NormalForm**$(w_0)$, $v_i w_i \equiv w_0$ by Lemma 6.2.1, we conclude that $|v_i w_i| \le \delta |w_0|$ for all $i$. Algorithm 2 terminates when $w_i = \varepsilon$. Since the length of $v_{i+1}$ is strictly greater than the length of $v_i$, Algorithm 2 terminates for any $w_0 \in A^*$ and the while loop of line 2 will be repeated at most $\delta |w_0|$ times.

Combining Lemma 6.2.1 with the fact that **NormalForm** terminates, we obtain the following corollary.

**Corollary 6.2.2.** *If $w_0 \in A^*$ is arbitrary, then the word $v$ returned by **NormalForm**$(w_0)$ is equivalent to $w_0$.*

### *6.3. Minimality*

We require the following definition and a number of related results to establish that the word returned by **NormalForm**$(w_0)$ is the lexicographic minimum word equivalent to $w_0$.

**Definition 6.3.1.** Let $w \in A^*$. A middle word $Y$ is called a *special* middle word of $w$ if $w = pYq$ for some $p, q \in A^*$ and there exists a word $p'XYZq'$ that is equivalent to $w$, $p'X \equiv p$, and $Zq' \equiv q$.

In other words, $Y$ is a special middle word of $w$ if it is a subword of $w$ and there exists a word equivalent to $w$ containing $XYZ$ as a factor in the obvious place. Note that if a relation word $XYZ$ is a factor of $w$, then it follows directly from the definition that $Y$ is a special middle word of $w$. Since middle words are not pieces, it follows that a middle word $Y_i$ will never occur as a factor of a middle word $Y_j$ unless $Y_i = Y_j$. So, if $Y_i$ and $Y_j$ are special middle words of a word $w$ and they begin at the same position in $w$, then $Y_i = Y_j$. In the following lemma, we prove that the special middle words of an arbitrary word $w$ do not overlap with each other.

**Lemma 6.3.2.** *Let $Y_i$ and $Y_j$ be special middle words of $w$ where $Y_i$ occurs strictly before $Y_j$. Then $w = pY_iqY_jr$ for some $p, q, r \in A^*$.*

**Proof.** Assume that $Y_i$ and $Y_j$ are such that $Y_i$ and $Y_j$ overlap as factors in $w$. Let $Y_i = xy$ and $Y_j = yz$ be such that $w = pY_izr = pxY_jr = pxyzr$ for some $x, y, z \in A^*$.

Since $Y_i$ is a special middle word of $w = pY_izr$, there exists $r' \in A^*$ such that $zr \equiv Z_ir'$. If $z$ is a prefix of $Z_i$, then $yz = Y_j$ is a factor of the relation word $X_iY_iZ_i$, a contradiction since $Y_j$ is not a piece. If $z$ is a prefix of $Z_ir'$ that is longer than $Z_i$, then the suffix $yZ_i$ of $X_iY_iZ_i$ which is longer than $Z_i$ is a factor of $Y_j$ and this contradicts the definition of $Z_i$. It follows that $z$ is not a prefix of $Z_ir$ and so $Z_ir' \neq zr$ and, in particular, $zr$ contains a relation word as a factor and hence $zr$ has a relation prefix and a clean overlap prefix. Hence $zr = aXYq'$ for $aXY$ a clean overlap prefix and some $q' \in A^*$. In addition, $|a| < |z|$ since $a$ is contained in every word equivalent to $aXYq'$ by Lemma 2.1.4, $aXYq' \equiv Z_ir'$ and $z$ is not a prefix of $Z_i$. Since $|a| < |z|$, there exists a suffix $X'$ of $z$ that is a prefix of $X$ and $zr = aXYq' = zX''Yq'$ where $X''$ is such that $X = X'X''$. Since $Y_j$ is a special middle word of $w$ and $w = pxY_jr = pxY_jX''Yq'$, $X''Yq'$ is equivalent to $Z_jt$ for some $t \in A^*$. Clearly, $Z_j$ is not a prefix of $X''Y$ because that would imply that a suffix of $X_jY_jZ_j$ longer than $Z_j$ is a factor of $XY$. In addition, $X''Y$ is not a prefix of $Z_j$ because $Y$ is not a piece. It follows that $Z_jt \neq X''Yq'$. Since $Z_jt \equiv X''Yq'$ but $Z_j$ is not a prefix of $X''Yq'$ it follows that $X''Yq'$ has a clean overlap prefix $bX_*Y_*$ with $|b| < |X''Y|$ because otherwise $X''Y$ would be a factor of all words equivalent to $X''Yq'$ and hence a factor of $Z_j$. If a prefix of $X''Y$ longer than $X''$ is a factor of $b$, then a prefix of $XY$ longer than $X$ is a factor of $Y_jZ_j$, a contradiction. It follows that either $X_*Y_*$ is a factor of $X''Y$ or $Y$ is a factor of $X_*Y_*$ and both cases lead to a contradiction.

We conclude that $Z_j$ cannot be a prefix of any word equivalent to $r$. In particular, it follows that $Y_j$ is not a special middle word of $w$ which contradicts the initial assumption and hence $Y_i$, $Y_j$ do not overlap. □

We can order the special middle words of a word $w \in A^*$ by their order of appearance as factors of $w$ from left to right. In particular, for every $w \in A^*$ we will refer to the *sequence of special middle words* $(Y_0, Y_1, \dots, Y_n)$ of $w$ where $i < j$ whenever $Y_i$ occurs to the left of $Y_j$ in $w$.

The next lemma collects some basic facts about the decomposition of the relation words $u$ into $X_uY_uZ_u$ that follow more or less immediately from the definition of the $C(4)$ condition.

**Lemma 6.3.3.** *Let $W_i = X_iY_iZ_i$ and $W_j = X_jY_jZ_j$ be arbitrary relation words in $R$ such that $W_i \neq W_j$.*

(i) *If a suffix of $Y_iZ_i$ is a prefix of $X_jY_j$, then a suffix of $Z_i$ is a prefix of $X_j$.*
(ii) *If $W_i$ overlaps $W_j$ in a word $w \in A^*$, then either: $Z_i$ overlaps with $X_j$ or $X_i$ overlaps with $Z_j$.*
(iii) *If $Y_u = sY_u'$ for some $s, Y_u' \in A^*$ with $s \neq \varepsilon$, then $X_us$ is not a factor of any relation word other than $u$.*
(iv) *Suppose that $w = X_iY_iZ_iX_j''Y_jZ_j$ where $X_j = X_j'X_j''$ and $X_j'$ is a suffix of $Z_i$. If $W_k$ is a relation word in $R$ such that $w \equiv pW_kq$ for some $p, q \in A^*$, then $W_k$ equals a complement $\overline{W_i}$ of $W_i$, or a complement $\overline{W_j}$ of $W_j$.*

To prove that the word returned by **NormalForm** is the lexicographically least word equal to the input word, we establish the following theorem.

**Theorem 6.3.4.** *Suppose that $u, v \in A^*$ are such that $u \equiv v$, that $u = u_0Y_0 \cdots u_mY_mu_{m+1}$, and that $v = v_0\overline{Y_0} \cdots v_n\overline{Y_n}v_{n+1}$ where $Y_i$ and $\overline{Y_i}$ are the special middle words in $u$ and $v$, respectively. Then $u = v$ if and only if $m = n$ and $Y_0 = \overline{Y_0}, \dots, Y_m = \overline{Y_m}$.*

We establish the proof of Theorem 6.3.4 in a sequence of lemmas. We start by showing that if $u \equiv v$, then there is a 1-1 correspondence between the special middle words of $u$ and $v$. Using the properties in Lemma 6.3.3 we obtain the following lemma to show that $Y$ is a special middle word of $u$, if and only if some complement $\overline{Y}$ of $Y$ is a special middle word of $v$.

**Lemma 6.3.5.** *Let $u, v \in A^*$. Assume that $Y$ is a special middle word of $u$ such that $u = pYq$ for some $p, q \in A^*$. Then $u \equiv v$ if and only if one of the following holds:*

(i) *$v = p'Yq'$ such that $p' \equiv p$ and $q' \equiv q$; or*
(ii) *$u = pYq \equiv rXYZt \equiv r\overline{XYZ}t \equiv p'\overline{Y}q' = v$ and $p \equiv rX$, $q \equiv Zt$, $p' \equiv r\overline{X}$ and $q' \equiv \overline{Z}t$.*

**Proof.** Clearly, if (i) or (ii) hold then $u \equiv v$. It remains to show that if $u \equiv v$ then (i) or (ii) holds for $v$. Assume that $u = pYq$ and let $v \equiv u$. Then there exists a rewrite sequence $u = pYq \xrightarrow{*} v$. We will prove that no relation applied in this rewrite sequence can overlap $Y$ unless it is $XYZ$.

It is clear that since $Y$ is not a piece, $Y$ is not a factor of any relation word except $XYZ$ and no relation word is a factor of $Y$. We start by showing that no relation word in the rewrite sequence $pYq \xrightarrow{*} v$ overlaps with a proper suffix of $Y$. Since $Y$ is a special middle word of $u$, $u = pYq \equiv rXYZt$ for some $r, t \in A^*$ such that $p \equiv rX$ and $q \equiv Zt$. Since $q \equiv Zt$, it follows by Lemma 2.1.4 that there are two cases to consider: either $q$ has a clean overlap prefix $aX_1Y_1$ with $|a| \geq |Z|$ and $Z$ is a prefix of all words equivalent to $q$ or $q$ has a clean overlap prefix $aX_1Y_1$ with $|a| < |Z|$ and $Z$ is a prefix of $a\overline{X_1Y_1}$ for some complement $\overline{X_1Y_1}$ of $X_1Y_1$. If the former holds, then no relation word in the rewrite sequence can overlap with a suffix of $Y$ because that would imply that either a suffix of $XYZ$ longer than $Z$ is a factor of a different relation word or that a relation word is a factor of $YZ$. Both of these lead to contradictions. Assume that $q = aX_1Y_1t'$ for some $t' \in A^*$. By Lemma 2.1.4 every word equivalent to $q$ has either $aX_1Y_1$ or $a\overline{X_1Y_1}$ as a prefix. If a relation word in the rewrite sequence overlaps a suffix of $Y$, then one of the following holds:

- it is a factor of $Ya$ which is a contradiction since $Ya$ is a factor of $YZ$;
- it is a factor of $Ya\overline{X_1Y_1}$ for some complement $\overline{X_1Y_1}$ of $X_1Y_1$ which is a contradiction because it implies that the relation word can be written as a product of 2 pieces; or
- the relation word contains $\overline{X_1Y_1}$ as a factor which is clearly a contradiction.

It follows that no relation word in the rewrite sequence overlaps with a suffix of $Y$ unless it is $XYZ$ in the obvious place.

Similarly, we can prove that no relation word in the rewrite sequence can overlap with a prefix of $Y$. Similar to the previous case, since $p \equiv rX$ there are two cases to consider: either $X$ is a suffix of all words equivalent to $p$ or it follows by Lemma 6.3.3(i) and (ii) that $p \equiv r'X_2Y_2Z_2b$ for some $r', b \in A^*$ such that $|b| < |X|$ and there exists a word $r'\overline{X_2Y_yZ_2}b$ for $\overline{X_2Y_2Z_2}$ some complement of $X_2Y_2Z_2$, that has $X$ as a suffix. It follows by Lemma 6.3.3(ii) and (iv) that any word overlapping with $\overline{X_2Y_2Z_2}$ that is not $\overline{X_2Y_2Z_2}$ in a rewrite sequence either overlaps with a prefix of $\overline{X_2}$ or with a suffix of $\overline{Z_2}$ and is $XYZ$. It follows that no relation word in the rewrite sequence overlaps with a prefix of $Y$ unless it is $XYZ$ in the obvious place.

In conclusion, no relation word in the rewrite sequence $u \xrightarrow{*} v$ overlaps with $Y$ unless it is a complement of $XYZ$ and hence (i) or (ii) holds for every $v$ such that $v \equiv u$. □

It follows by Lemma 6.3.5 that if $u \equiv v$ and $Y_i$ is a special middle word of $u$, then some complement $\overline{Y_i}$ of $Y_i$ is a special middle word of $v$. Note that it also follows by Lemma 6.3.5(i) and (ii) that if $Y_i$, $Y_j$ are special middle words of $u$ and $Y_i$ occurs to the left of $Y_j$ in $u$, then the corresponding special middle word $\overline{Y_i}$ occurs to the left of $\overline{Y_j}$ in $v$. In particular, the following result holds as a corollary of Lemma 6.3.5.

**Corollary 6.3.6.** *If $u \equiv v$, then $(Y_0, Y_1, \dots Y_n)$ is the sequence of special middle words of $u$ if and only if the sequence of special middle words of $v$ is $(\overline{Y_0}, \overline{Y_1}, \dots \overline{Y_n})$ where $\overline{Y_i}$ is a complement of $Y_i$ for each $i$.*

The next lemma collects some basic facts about special middle words that follow more or less immediately from the definition of special middle words and the definition of clean overlap prefixes. We will use these properties in various results in the remainder of this section.

**Lemma 6.3.7.** *Let $w \in A^*$.*

(i) *If $Y$ is a special middle word of $w$ and $w = pYq$ for some $p, q \in A^*$, then $Z$ is a possible prefix of $q$ and* ***WpPrefix***$(q, q, Z)$*=Yes.*
(ii) *If $w = XYw'$ for some $w' \in A^*$ and* ***WpPrefix***$(w', w', Z)$*=Yes, then $XY$ is a clean overlap prefix of $w$.*
(iii) *If $Y$ is the middle word in line 18 of Algorithm 2 and the condition of line 19 is satisfied, then $Y$ is not a special middle word of $w = aXYw'$.*
(iv) *If $w = a_0X_0Y_0w'$ for some $w' \in A^*$ and $Y_0$ is the left most special middle word in $w$, then any word equivalent to $w$ has prefix $a_0$.*

We use some of the properties in Lemma 6.3.7 to prove the next lemma which refines the form of a word with respect to a C(4) monoid presentation.

**Lemma 6.3.8.** *Let $u \in A^*$. Assume that $(Y_0, Y_1, \ldots, Y_n)$, for some $n \geq 0$, is the sequence of special middle words of $u$. Then*

$$u = a_0X_0Y_0a_1X_1''Y_1a_2X_2''Y_2a_3 \ldots a_nX_n''Y_na_{n+1}$$

*where $a_i \in A^*$ and either $X_i'' = X_i$ or $X_i''$ is a proper suffix of $X_i$ and $a_i = Z_{i-1}$ for all $i$.*

**Proof.** Since $Y_0$ is the left most special middle word of $u$, $u = pY_0q$ and $p \equiv a_0X_0$ for some $a_0 \in A^*$. If a relation word $XYZ$ is a factor of $p$, then $Y$ is a factor of $p$ and hence it is a special middle word of $u$ occurring on the left of $Y_0$. This is a contradiction since $Y_0$ is the left most special middle word of $u$ and hence $p$ contains no relation words as factors. It follows that $p = a_0X_0$.

Let $Y_{k-1}, Y_k$ be special middle words of $u$. Then $u = rY_{k-1}b_kY_kt$ for some $r, b_k, t \in A^*$, by Lemma 6.3.2. We will prove that $b_k = a_kX_k''$ where either $X_k'' = X_k$ or $X_k''$ is a proper suffix of $X_k$ and $a_k = Z_{k-1}$. Since $Y_{k-1}$ is a special middle word of $u$, $b_kY_kt \equiv Z_{k-1}s$ for some $s \in A^*$. It follows that either $Z_{k-1}$ is a prefix of all words equivalent to $b_kY_kt$ or $b_kY_kt$ has a clean overlap prefix $cXY$ with $|c| < |Z_{k-1}|$ and $Z_{k-1}$ is a prefix of $c\overline{XY}$ for some complement $\overline{XY}$ of $XY$.

In the latter case, since $b_kY_kt = cXYq \equiv c\overline{XY}q'$ for some $q, q' \in A^*$, it follows that $cXYq \equiv cXYZq''$ for some $q'' \in A^*$ and hence $Y$ is a special middle word of $cXYq$. Since $cXYq$ is a suffix of $u$ it follows by the definition of special middle words that $Y$ is a special middle word of $u$.

We will prove that $Y = Y_k$ and hence $b_k = cX_k = a_kX''_k$ where $X''_k = X_k$, as required. Clearly, if $|cXY| = |b_kY_k|$ and $Y \neq Y_k$ then either $Y$ is a suffix of $Y_k$ or $Y_k$ is a suffix of $Y$. This is a contradiction since middle words are not pieces and hence $Y = Y_k$. Assume that $|cXY| < |b_kY_k|$. Then $Y$ is a special middle word that occurs after $Y_{k-1}$ and before $Y_k$ as a factor of $u$ and this is a contradiction. Assume that $|cXY| > |b_kY_k|$. There are two cases to consider: either $Y_k$ is a factor of $XY$, which is clearly a contradiction, or $Y_k$ is a factor of $cXY$ that begins before the end of $c$. In this case, $Y_k$ must end before the start of $Y$ in $cXY$ because otherwise it would contain a factor of $XY$ longer than $X$. Since $Y_k$ is a special middle word, it follows that $b_kY_kt \equiv b_kY_kZ_kt'$ for some $t' \in A^*$. If $Z_k$ is a prefix of $t$ in this case, then either $Y$ is a factor of $X_kY_kZ_k$ or $XY$ contains a suffix of $X_kY_kZ_k$ longer than $Z_k$ and both of these lead to a contradiction. It follows that $Z_k$ is not a prefix of $t$, and hence $t$ has a clean overlap prefix $dX_*Y_*$ with $|d| < |Z_k|$. If $d$ ends after the end of $Y$ in $cXYq$ then $Y$ is a factor of $X_kY_kZ_k$, a contradiction. It follows that $d$ ends before the end of $Y$. But then $XY$ overlaps with $X_*Y_*$ which is a contradiction because $cXY$ is a clean overlap prefix of $cXYq$. It follows that $Y = Y_k$.

In the former case, $Z_{k-1}$ is a prefix of all words equivalent to $b_kY_kt$. If $b_kY_kt$ has a clean overlap prefix $cXY$ with $|c| < |Z_{k-1}|$ and $b_kY_kt \equiv c\overline{XY}q'$ for some $q' \in A^*$ and a proper complement $\overline{XY}$ of $XY$, then as in the previous paragraph $Y = Y_k$ and hence $b_k = a_kX_k$, as required. Assume that $b_kY_kt$ does not have a clean overlap prefix $cXY$ with $|c| < |Z_{k-1}|$ such that $b_kY_kt \equiv c\overline{XY}q'$ for some $q' \in A^*$ and a proper complement $\overline{XY}$ of $XY$. Then, since $Y_k$ is a special middle word of $u$, if $X_k$ is a suffix of $b_k$ then it follows by Lemma 6.3.7(ii) that $X_kY_k$ is a clean overlap prefix of $X_kY_kt$. If $b_kY_kt = b'X_kY_kt$ for some $b' \in A^*$ and there is not a clean overlap prefix contained in $b'$, then $b'X_kY_k$ is a clean overlap prefix of $b_kY_kt$ and $b_k = a_kX_k$, as required. If this is not the case, then either $X_k$ is a suffix of $b_k$ but $b_kY_kt$ has a clean overlap prefix $cXY$ with $|cXY| \leq |b_k|$ such that no equivalent of $b_kY_kt$ has prefix $c\overline{XY}$ for $\overline{XY}$ a proper complement of $XY$ or $X_k$ is not a suffix of $b_k$. In the first case, $b_k = a_kX_k$ and hence $X''_k = X_k$ as required. In the second case, if $X_k$ is not a suffix of $b_k$, then since no word equivalent to $u$ contains a relation word between $Y_{k-1}$ and $Y_k$, it follows that $X_k = X'_kX''_k$ and the proper prefix $X'_k$ of $X_k$ is a suffix of some complement $\overline{Z_{k-1}}$ of $Z_{k-1}$. It follows that $b_k = Z_{k-1}X''_k$, as required. □

At this point, we have proved results that explain the connection between the sequence of special middle words of a word $w$ and the form of $w$ or a word equivalent to $w$ with respect to this sequence. We would like to be able to compare words based on their sequences of special middle words. We utilize the algorithm **WpPrefix** from [11] to do this. The following results highlight the connection between special middle words in equivalent words $u$ and $v$ and recursive calls from within **WpPrefix**$(u, v, \varepsilon)$.

**Lemma 6.3.9.** *If **WpPrefix**$(u_j, v_j, p_j)$ is a recursive call from within **WpPrefix**$(u, v, \varepsilon)$ for $u, v \in A^*$ and $u \equiv v$, then $u_j$ is a suffix of a word equivalent to $u$ and $v_j$ is a suffix of a word equivalent to $v$. In addition, if $u_j = XYu'$ and $XY$ is a clean overlap prefix of*

*$u_j$, and $v_j = \overline{XY}v'$ and $\overline{XY}$ is a clean overlap prefix of $v_j$, then $Yu'$ is a suffix of $u$ and $\overline{Y}v'$ is a suffix of $v$.*

**Proof.** All line numbers in this proof refer to **WpPrefix** in [11]. Clearly, if **WpPrefix**$(u_j, v_j, p_j)$ is the initial call to **WpPrefix**$(u, v, \varepsilon)$, then $u_j = u, v_j = v$ and hence $u_j$ and $v_j$ are suffixes of $u$ and $v$, respectively. In addition if $u_j = u = XYu'$ and $XY$ is a clean overlap prefix of $u_j$, and $v_j = v = \overline{XY}v'$ and $\overline{XY}$ is a clean overlap prefix of $v_j$ then clearly $Yu'$ is a suffix of $u$ and $\overline{Y}v'$ is a suffix of $v$. Assume that the result holds for the $j$-th recursive call **WpPrefix**$(u_j, v_j, p_j)$ from within **WpPrefix**$(u, v, \varepsilon)$. We will show that the result holds for the recursive call **WpPrefix**$(u_{j+1}, v_{j+1}, p_{j+1})$ that occurs immediately after **WpPrefix**$(u_j, v_j, p_j)$. Since the result holds for $u_j$, there exists a word $w \equiv u$ with suffix $u_j$. If **WpPrefix**$(u_{j+1}, v_{j+1}, p_{j+1})$ occurs in one of lines 15, 25, 28, 29, 31 or 42, then $u_{j+1}$ is a suffix of $u_j$ and the result holds for $u_{j+1}$. If **WpPrefix**$(u_{j+1}, v_{j+1}, p_{j+1})$ occurs in line 24 or 33 then $u_j = XYZu''$ for some $u'' \in A^*$ and $u_{j+1} = \hat{Z}u''$ for $\hat{Z}$ a complement of $Z$. Since $u_j = XYZu'' \equiv \hat{X}\hat{Y}\hat{Z}u''$ there exists a word equivalent to $w$ with suffix $\hat{X}\hat{Y}\hat{Z}u''$ and hence $u_{j+1} = \hat{Z}u''$ is a suffix of a word equivalent to $u$. The proof that $v_{j+1}$ is a suffix of a word equivalent to $v$ is analogous.

It remains to show that if $u_j$ has clean overlap prefix $XY$ and $u_j = XYu'$ and $v_j$ has clean overlap prefix $\overline{XY}$ and $v_j = \overline{XY}v'$, then $Yu'$ is a suffix of $u$ and $\overline{Y}v'$ is a suffix of $v$. We will prove this for $u_j$, the proof for $v_j$ is analogous. As stated above, the result holds if **WpPrefix**$(u_j, v_j, p_j)$ is the initial call to **WpPrefix**$(u, v, \varepsilon)$ and $u$ has clean overlap prefix $XY$. If this is not the case, then the algorithm will make a number of calls in line 15 until a suffix of $u$ that has the form $XYu'$ with $XY$ a clean overlap prefix is found and hence the result holds for $Yu'$ in this case as well. We will assume that the result holds for **WpPrefix**$(u_j, v_j, p_j)$ such that $u_j$ has clean overlap prefix $X_jY_j$ and $u_j = X_jY_ju_j'$ and we will show that holds for the recursive call **WpPrefix**$(u_k, v_k, p_k)$ that is the first recursive call after **WpPrefix**$(u_j, v_j, p_j)$ such that $u_k$ has clean overlap prefix $X_kY_k$ and $u_k = X_kY_ku'$. Since $u_j$ has clean overlap prefix $X_jY_j$ and $u_j \equiv v_j$, the recursive call to **WpPrefix**$(u_{j+1}, v_{j+1}, p_{j+1})$ that occurs immediately after **WpPrefix**$(u_j, v_j, p_j)$ occurs in one of lines 24, 25, 28, 29, 31, 33 or 42. If it occurs in line 25, 28, 29, 31 or 42, then $u_{j+1}$ is a suffix of $u_j$ and since $Y_ju_j'$ is a suffix of $u$ it follows that $u_{j+1}$ is a suffix of $u$. Since **WpPrefix**$(u_k, v_k, p_k)$ is the first recursive call after **WpPrefix**$(u_j, v_j, p_j)$ such that $u_k$ has clean overlap prefix $X_kY_k$ and $u_k = X_kY_ku'$, it follows that any recursive call after **WpPrefix**$(u_{j+1}, v_{j+1}, p_{j+1})$ and before **WpPrefix**$(u_k, v_k, p_k)$ can only occur in line 15. It follows that $u_k$ is a suffix of $u_{j+1}$ and hence $Y_ku_k'$ is a suffix of $u$. If the recursive call to **WpPrefix**$(u_{j+1}, v_{j+1}, p_{j+1})$ occurs in line 24 or 33, then $u_j = X_jY_jZ_ju''$, $Y_jZ_ju''$ is a suffix of $u$ and $u_{j+1} = \hat{Z}_ju''$ for some complement $\hat{Z}_j$ of $Z_j$. Similar to the previous case, any recursive call after **WpPrefix**$(u_{j+1}, v_{j+1}, p_{j+1})$ and before **WpPrefix**$(u_k, v_k, p_k)$ can only occur in line 15. Hence $u_{j+1} = aX_kY_ku_k'$ for $aX_kY_k$ a clean overlap prefix of $u_{j+1}$. Since $\hat{Z}_j$ is a prefix of $u_{j+1}$, $|aX_k| \geq |\hat{Z}_j|$, otherwise a prefix of $X_kY_kZ_k$ longer than $X_k$ would be a factor of $\hat{Z}_j$. It follows that $Y_ku_k'$ is a suffix of $u''$ and hence a suffix of $u$. □

The following result holds as a direct consequence of Lemma 6.3.9 and the definition of special middle words and can be viewed as a tool to "identify" special middle words inside a word $w$ in some cases.

**Lemma 6.3.10.** *Let $u, v \in A^*$ be such that $u \equiv v$. Assume that there exists a recursive call* **WpPrefix**$(XYu', \overline{XY}v', p)$ *from within* **WpPrefix**$(u, v, \varepsilon)$ *for some $p, u', v' \in A^*$ such that $\overline{XY}$ is a proper complement of $XY$. Then $Y$ is a special middle word of $u$ and $\overline{Y}$ is the corresponding special middle word of $v$.*

**Proof.** Since $u \equiv v$ it follows that **WpPrefix**$(u, v, \varepsilon)$ returns Yes and hence since **WpPrefix**$(XYu', \overline{XY}v', p)$ is a recursive call from within **WpPrefix**$(u, v, \varepsilon)$, **WpPrefix**$(XYu', \overline{XY}v', p)$ returns Yes as well. It follows that $XYu' \equiv \overline{XY}v'$. Since $\overline{XY}$ is a proper complement of $XY$ it follows by Lemma 3 in [11] that $u' \equiv Zu''$ and $v' \equiv \overline{Z}v''$ for some $u'', v'' \in A^*$. In addition, it follows by Lemma 6.3.7 (ii) that $XY$ is a clean overlap prefix of $XYu'$ and $\overline{XY}$ is a clean overlap prefix of $\overline{XY}v'$. By Lemma 6.3.9, $XYu'$ is a suffix of a word equivalent to $u$ and hence there exists a word $w$ such that $w \equiv u$ and $XYZu''$ is a suffix of $w$. It follows that $Y$ is a special middle word of $w$ and by Corollary 6.3.6 some complements of $Y$ are the corresponding middle words in $u$ and $v$. By Lemma 6.3.9, $Yu'$ is a suffix of $u$ and $\overline{Y}v'$ is a suffix of $v$ and hence $Y$ and $\overline{Y}$ are the corresponding special middle words in $u$ and $v$, respectively. □

The next two results will be useful when comparing equivalent words based on the sequences of their special middle words.

**Lemma 6.3.11.** *Let $u, v \in A^*$ be such that $u \equiv v$. Assume that $Y$ is a special middle word of $u$ and let $\overline{Y}$ be the corresponding special middle word in $v$. Then there exists a recursive call* **WpPrefix**$(XYu', \overline{XY}v', p)$ *from within* **WpPrefix**$(u, v, \varepsilon)$ *for some $p, u', v' \in A^*$.*

**Proof.** All line numbers in this proof refer to **WpPrefix** in [11]. Assume that $(Y_0, Y_1, \dots, Y_n)$ is the sequence of special middle words of $u$. By Lemma 6.3.8, $u = a_0X_0Y_0a_1X_1''Y_1a_2X_2''Y_2a_3\dots a_nX_n''Y_na_{n+1}$ and $v = a_0\overline{X_0Y_0}b_1\overline{X_1''Y_1}b_2\overline{X_2''Y_2}b_3\dots b_n\overline{X_n''Y_n}b_{n+1}$ where $a_i, b_i \in A^*$ for all $i$ and $X_i''$ and $\overline{X_i''}$ are suffixes of $X_i$ and $\overline{X_i}$, respectively, for all $i$. We start by showing that the result holds for $Y_0$ and $\overline{Y_0}$. Since $u = a_0X_0Y_0u'$ for some $u' \in A^*$ and $Y_0$ is a special middle word of $u$, it follows by definition that $u' \equiv Z_0u''$ for some $u'' \in A^*$. It follows by Lemma 6.3.7(ii) applied to $X_0Y_0u'$ that $X_0Y_0$ is a clean overlap prefix of $X_0Y_0u'$. Since there are no relation words contained in $a_0$ and $a_0$ is also a prefix of $v$, **WpPrefix**$(u, v, \varepsilon)$ starts by making recursive calls in lines 15 and 28, until the prefix $a_0$ has been deleted and the recursive call to **WpPrefix**$(X_0Y_0u', \overline{X_0Y_0}v', p_0)$ occurs for some piece $p_0$.

We now assume that there exists a recursive call to **WpPrefix**$(X_kY_ku_k', \overline{X_kY_k}v_k', p_k)$ from within **WpPrefix**$(u, v, \varepsilon)$ for some $p_k, u_k', v_k' \in A^*$ for special middle words $Y_k$ and $\overline{Y_k}$, respectively, and we will prove that there exists a recursive call to

**WpPrefix**$(X_{k+1}Y_{k+1}u'_{k+1}, \overline{X_{k+1}Y_{k+1}}v'_{k+1}, p_{k+1})$ from within **WpPrefix**$(u, v, \varepsilon)$ for some $p_{k+1}, u'_{k+1}, v'_{k+1} \in A^*$ for special middle words $Y_{k+1}$ and $\overline{Y_{k+1}}$, respectively.

We will show this by examining the recursive calls that occur after **WpPrefix**$(X_kY_ku'_k, \overline{X_kY_k}v'_k, p_k)$. If $Z_k$ is not a prefix of $u'_k$ then the recursive call immediately after **WpPrefix**$(X_kY_ku'_k, \overline{X_kY_k}v'_k, p_k)$ occurs in one of lines 28, 29, 31 and 42. Since $Z_k$ is not a prefix of $u'_k$ and $Y_ku'_k$ is a suffix of $u$ by Lemma 6.3.9, it follows by Lemma 6.3.8 that $a_{k+1}X''_{k+1} = a_{k+1}X_{k+1}$. In addition, since $Y_k$ is a special middle word, $u'_k$ is equivalent to a word with prefix $Z_k$ and hence it has a clean overlap prefix $cXY$ with $|c| < |Z_k|$. Since $Z_k$ is not a prefix of $cX$ then it must be a prefix of $c\overline{X}$ for $\overline{X}$ a proper complement of $X$.

If the recursive call occurs in one of lines 28, 29 or 31 then the first argument is $u'_k$ which in this case has $a_{k+1}X_{k+1}Y_{k+1}$ as a prefix. If $a_{k+1}X_{k+1}Y_{k+1}$ is a clean overlap prefix of $u'_k$ the algorithm continues by making $|a_{k+1}|$ recursive calls in line 15. Then since $u \equiv v$ it makes the call **WpPrefix**$(X_{k+1}Y_{k+1}u'_{k+1}, V, p_{k+1})$ and $V = \overline{X_{k+1}Y_{k+1}}v'_{k+1}$ for some $p_{k+1}, u'_{k+1}, v'_{k+1} \in A^*$. In addition, since this call was proceeded by $|a_{k+1}|$ calls in line 15 it follows that $v'_k$ has the prefix $a_{k+1}\overline{X_{k+1}Y_{k+1}}$ and since there is no clean overlap prefix in $a_{k+1}$, it follows by Lemma 6.3.7(ii) that $\overline{Y_{k+1}}$ is the left most special middle word occurring to the right of $\overline{Y_k}$ in $v$ and hence it is the special middle word that corresponds to $Y_{k+1}$ in $u$. Assume that $a_{k+1}X_{k+1}Y_{k+1}$ is not a clean overlap prefix of $u'_k$. Since $Y_{k+1}$ is a special middle word of $u$, it follows by Lemma 6.3.7 (ii) that $X_{k+1}Y_{k+1}$ is a clean overlap prefix of a suffix of $u'_k$. It follows that the clean overlap prefix $cXY$ of $u'_k$ is such that $|cXY| \leq |a_{k+1}|$, otherwise either $cXY$ would overlap with $X_{k+1}Y_{k+1}$ or it would contain $X_{k+1}Y_{k+1}$ as a factor and both of these contradict the definition of a clean overlap prefix. It follows that $cXY$ is such that $|cXY| \leq |a_{k+1}|$ and since $Y$ is not a special middle word of $u$, $cXY$ is not followed by $Z$ and $u'_k$ is only equivalent to words that have $cXY$ as a prefix. It follows that in this case the algorithm makes recursive calls in lines 15 and 28 until $a_{k+1}$ gets deleted. Similar to the previous case, $a_{k+1}$ is a prefix of $v'_k$, the algorithm makes the recursive call **WpPrefix**$(X_{k+1}Y_{k+1}u'_{k+1}, \overline{X_{k+1}Y_{k+1}}v'_{k+1}, p_{k+1})$ for some $p_{k+1}, u'_{k+1}, v'_{k+1} \in A^*$ and $\overline{Y_{k+1}}$ is the special middle word of $v$ that corresponds to $Y_{k+1}$ in $u$.

If the recursive call occurs in line 42 then by the assumptions of this case $a_{k+1}$ has prefix $z_1$ and any clean overlap prefix $XY$ of $u'_k$ begins after the end of $z_1$. It follows that the result holds in this case following the same argument as in the case of lines 28, 29 and 31 applied to the suffix of $u'_k$ that follows $z_1$.

It remains to show that the result holds when $Z_k$ is a prefix of $u'_k$. In this case the recursive call immediately after **WpPrefix**$(X_kY_kZ_ku''_k, \overline{X_kY_k}v'_k, p_k)$ occurs in one of lines 24, 25 or 33. In case the call occurs in line 33 the result holds by symmetry with line 31. If the recursive call occurs in line 25, then $u''_k$ is not $\hat{Z}_{k+1}$-active for some complement $\hat{Z}_{k+1}$ of $Z_{k+1}$ and hence $a_{k+1}$ has the same form as in the case of lines 28, 29 and 31. Hence the same argument can show that the result holds in this case.

If $X''_{k+1}$ is a proper suffix of $X_{k+1}$ then $u''_k$ is $\hat{Z}_{k+1}$-active for some complement $\hat{Z}_{k+1}$ of $Z_{k+1}$, otherwise no word equivalent to $u$ would contain $X_{k+1}Y_{k+1}$ as a factor in this position. It follows that in this case the algorithm makes a recursive call in line 24 and $\hat{Z}_{k+1}u''_k$

has the clean overlap prefix $bX_{k+1}Y_{k+1}$ for some $b \in A^*$ with $|b| < |\hat{Z}_{k+1}|$. It follows that the algorithm makes $|b|$ recursive calls in line 15 and the result holds for $Y_{k+1}$. If $u_k''$ is $\hat{Z}_{k+1}$-active but the clean overlap prefix of $\hat{Z}_{k+1}u_k''$ is $cXY$ with $Y \neq Y_{k+1}$, then $a_{k+1}$ has the same form as in the cases of lines 25, 28, 29 and 31 and the result holds in this case. □

The following result holds as a corollary of Lemma 6.3.11.

**Corollary 6.3.12.** *Let $u, v \in A^*$ be such that $u \equiv v$. Assume that $Y_k$ and $Y_{k+1}$ are consecutive special middle words of $u$. Assume that $u_k, u_{k+1}, v_k, v_{k+1}$ are such that* ***WpPrefix***$(u_k, v_k, p_k)$*,* ***WpPrefix***$(u_{k+1}, v_{k+1}, p_{k+1})$ *are the recursive calls corresponding to $Y_k, Y_{k+1}$ from within* ***WpPrefix***$(u, v, \varepsilon)$ *from Lemma 6.3.11. If* ***WpPrefix***$(u_j, v_j, p_j)$ *is a recursive call from within* ***WpPrefix***$(u, v, \varepsilon)$ *that occurs after* ***WpPrefix***$(u_k, v_k, p_k)$ *and before* ***WpPrefix***$(u_{k+1}, v_{k+1}, p_{k+1})$ *and it is not the recursive call that occurs immediately after* ***WpPrefix*** $(u_k, v_k, p_k)$*, then* ***WpPrefix***$(u_j, v_j, p_j)$ *occurs either in line 15 or line 28 of* ***WpPrefix****.*

**Proof.** All line numbers in this proof refer to **WpPrefix** in [11]. Assume that **WpPrefix**$(u_j, v_j, p_j)$ is the recursive call from within **WpPrefix**$(u, v, \varepsilon)$ as described in the statement of this lemma. If **WpPrefix**$(u_j, v_j, p_j)$ occurs in one of lines 31, 33 or 42 then $u_j = XYu'$ where $XY$ is a clean overlap prefix of $u_j$ and $v_j = \overline{XY}v'$ where $\overline{XY}$ is a clean overlap prefix of $v_j$ and $u', v' \in A^*$ and $\overline{XY}$ is a proper complement of $XY$. It follows by Lemma 6.3.10 that there exists a special middle word in a word equivalent to $u$ that occurs between $Y_k$ and $Y_{k+1}$, a contradiction. In addition, if **WpPrefix**$(u_j, v_j, p_j)$ occurs in lines 24, 25 or 29 then either $u_j = XYZu''$ for some relation word $XYZ$ and some $u'' \in A^*$ or $u_j = XYu'$ and $Z$ is a possible prefix of $u'$. By Lemma 6.3.9 there exists a word equivalent to $u$ containing a relation word $XYZ$ as a factor and hence $Y$ is a special middle word. It follows that there exists a special middle word in a word equivalent to $u$ that occurs between $Y_k$ and $Y_{k+1}$, a contradiction. It follows that **WpPrefix**$(u_j, v_j, p_j)$ can only occur in line 15 or 28. □

We are now ready to prove the following lemma. Theorem 6.3.4 will hold as a corollary of Lemma 6.3.13. In addition, Lemma 6.3.13 will be used as a tool to compare prefixes of equivalent words when proving the correctness of the algorithm.

**Lemma 6.3.13.** *Suppose that $u, v \in A^*$ are such that $u \equiv v$, that $u = u_0Y_0 \cdots u_mY_mu_{m+1}$, and that $v = v_0\overline{Y_0} \cdots v_n\overline{Y_m}v_{m-1}$ where $Y_i$ are the special middle words in $u$ and $\overline{Y_i}$ are the special middle words in $v$. If $Y_0 = \overline{Y_0}, \ldots, Y_k = \overline{Y_k}$ for some $k$, then $u_0Y_0 \cdots u_kY_k = v_0\overline{Y_0} \cdots v_k\overline{Y_k}$.*

**Proof.** All line numbers in this proof refer to **WpPrefix** in [11]. It follows by Lemma 6.3.7(iv) and Lemma 6.3.8, that $u_0Y_0 = a_0X_0Y_0$ and $a_0$ is a prefix of every word equivalent to $u$. Since $Y_0 = \overline{Y_0}$, $u_0Y_0 = v_0\overline{Y_0}$ by [11, Lemma 2].

Assume that the result holds for $Y_0 = \overline{Y_0}, \ldots, Y_{k-1} = \overline{Y_{k-1}}$ for some $k \geq 1$. Since $u \equiv v$ and $Y_k$ is a special middle word of $u$, it follows by Lemma 6.3.11 that there exist recursive calls to **WpPrefix**$(u_{k-1}, v_{k-1}, p_{k-1})$ and **WpPrefix**$(u_k, v_k, p_k)$ from within **WpPrefix**$(u, v, \varepsilon)$ such that $u_{k-1} = X_{k-1}Y_{k-1}u'_{k-1}$, that $v_{k-1} = \overline{X_{k-1}Y_{k-1}}v'_{k-1}$, $u_k = X_kY_ku'_k$ and $v_k = \overline{X_kY_k}v'_k$. Since $Y_{k-1} = \overline{Y_{k-1}}$ and $Y_k = \overline{Y_k}$ the next recursive call to **WpPrefix** within **WpPrefix**$(u_{k-1}, v_{k-1}, p_{k-1})$ must occur in one of lines 24, 25, 28, or 29 (in the other cases the prefix of $u_{k-1}$ is a proper complement of the prefix of $v_{k-1}$). The only subsequent type of recursive call that can occur between **WpPrefix**$(u_{k-1}, v_{k-1}, p_{k-1})$ and **WpPrefix**$(u_k, v_k, p_k)$ is in lines 15 and 28 by Corollary 6.3.12. Hence $u_k = v_k$ since in the recursive calls of lines 15 and 28 equal prefixes of the first two arguments are deleted. It follows that $u_0Y_0 \cdots u_kY_k = v_0\overline{Y_0} \cdots v_k\overline{Y_k}$. □

Having established Theorem 6.3.4, in the next 2 lemmas we consider how the special middle words $u, v \in A^*$ such that $u \equiv v$ interact with the lexicographic order.

**Lemma 6.3.14.** *Suppose that $u, v \in A^*$, that $u \equiv v$, and that $u = pY_{k-1}Z_{k-1}X''_kY_kq$ for special middle words $Y_{k-1}$ and $Y_k$ in $u$, some $p, q \in A^*$ and some proper suffix $X''_k$ of $X_k$. If $Y_{k-1}$ and a proper complement $\overline{Y_k}$ of $Y_k$ are the corresponding middle words in $v$, then there is a suffix $X'_k$ of a complement $\overline{Z_{k-1}}$ of $Z_{k-1}$ such that $X_k = X'_kX''_k$ and $X'_k$ is also a prefix of $\overline{X_k}$.*

**Proof.** Since $Y_{k-1}, Y_k$ are special middle words of $u$, there is a rewrite sequence $u \xrightarrow{*} p'X_{k-1}Y_{k-1}Z_{k-1}X''_kY_kq \to p'\overline{X_{k-1}Y_{k-1}Z_{k-1}}X''_kY_kq$ where $\overline{Z_{k-1}} = zX'_k$ for some $z \in A^*$. Hence

$$p'\overline{X_{k-1}Y_{k-1}Z_{k-1}}X''_kY_kZ_kq_0 = p'\overline{X_{k-1}Y_{k-1}}zX_kY_kZ_kq_0 \to p'\overline{X_{k-1}Y_{k-1}}z\overline{X_kY_kZ_k}q_0$$

for some $q_0 \in A^*$. Since $Z_{k-1}$ is a factor of $v$, but not of $p'\overline{X_{k-1}Y_{k-1}}z\overline{X_kY_kZ_k}q_0$ by the assumption of this case,

$$p'\overline{X_{k-1}Y_{k-1}}z\overline{X_kY_kZ_k}q_0 \xrightarrow{*} p'\overline{X_{k-1}Y_{k-1}Z_{k-1}}\,\overline{X''_kY_k}q_1 \to p'X_sY_sZ_{k-1}\overline{X''_kY_k}q_1 \xrightarrow{*} v$$

where $X_sY_sZ_{k-1}$ is a complement of $X_{k-1}Y_{k-1}Z_{k-1}$ with suffix $Z_{k-1}$ and $v = p_0Z_{k-1}\overline{X''_kY_k}q'$. In particular, since

$$p'\overline{X_{k-1}Y_{k-1}}z\overline{X_kY_kZ_k}q_0 \xrightarrow{*} p'\overline{X_{k-1}Y_{k-1}Z_{k-1}}\,\overline{X''_kY_k}q_1,$$

$\overline{X_k} = X'_k\overline{X_k}''$ and so $X'_k = \overline{X_k}'$ is the required common prefix. □

In the next lemma, we make use of the following observation: if $u, v \in A^*$ are such that $u < v$ and $u$ is not a prefix of $v$, then $uw < vw'$ for all $w, w' \in A^*$.

**Lemma 6.3.15.** *Suppose that $u, v \in A^*$ are such that $u \equiv v$. If $u < v$ and $Y_k$ is the left most special middle word in $u$ such that the corresponding middle word $\overline{Y_k}$ in $v$ is a proper complement of $Y_k$, then $X_kY_kZ_k < \overline{X_kY_kZ_k}$.*

**Proof.** Let $u := w_0, w_1, \ldots, w_n := v$ be any rewrite sequence. By Lemmas 6.3.8 and 6.3.13, there exist $a_k, b_k, p, q, q' \in A^*$ such that $u = pX''_{k-1}Y_{k-1}a_kX''_kY_kq$ and $v = pX''_{k-1}Y_{k-1}b_k\overline{X_k}''\overline{Y_k}q'$ where $X''_k$ and $\overline{X_k}''$ are (not necessarily proper) suffixes of $X_k$ and $\overline{X_k}$, respectively. Since $Y_k$ is a special middle word of $u$, it follows by Lemma 6.3.5 (ii) that there exists $j \in \{0, \ldots, n\}$ such that $X_kY_kZ_k$ is a factor of $w_j$ and $\overline{X_kY_kZ_k}$ is a factor of $w_{j+1}$. It remains to show that $X_kY_kZ_k < \overline{X_kY_kZ_k}$.

It follows by Lemma 6.3.9 and Lemma 6.3.11 that there exists a recursive call to **WpPrefix**$(X_{k-1}Y_{k-1}a_k\ X''_kY_kq,\ X_{k-1}Y_{k-1}b_k\overline{X_k}''\overline{Y_k}q', t)$ for some piece $t$, from within **WpPrefix**$(u, v, \varepsilon)$. In addition, $X_{k-1}Y_{k-1}$ is a clean overlap prefix of the first two arguments of this call by Lemma 6.3.7 (i) and (ii). Since the first two arguments of this call begin with the clean overlap prefix $X_{k-1}Y_{k-1}$, the recursive call occurring immediately after **WpPrefix**$(X_{k-1}Y_{k-1}a_kX''_kY_kq,\ X_{k-1}Y_{k-1}b_k\overline{X_k}''\overline{Y_k}q', t)$ must occur in one of lines 24, 25, 28, 29 and the only possible recursive calls that can occur before **WpPrefix**$(X_kY_kq, \overline{X_kY_k}q', t')$ for some piece $t'$, are those in lines 15 or 28 by Corollary 6.3.12. Since in the recursive calls of lines 15 and 28 equal prefixes of the first two arguments are deleted, it follows that $a_k = b_k$ and $v = pX_{k-1}Y_{k-1}a_k\overline{X''_kY_k}q'$. Since $u < v$ and $X_kY_k \neq \overline{X_kY_k}$, it follows that $X''_kY_k < \overline{X''_kY_k}$.

We will show that this implies that $X_kY_k < \overline{X_kY_k}$ and hence $X_kY_kZ_k < \overline{X_kY_kZ_k}$. There are two cases to consider: when $X''_k = X_k$ and when $X''_k$ is a proper suffix of $X_k$. If $X''_k = X_k$ then it follows that $\overline{X''_k} = \overline{X_k}$, since otherwise there would not be a recursive call to **WpPrefix**$(X_kY_kq, \overline{X_kY_k}q', t')$ from within **WpPrefix**$(u, v, \varepsilon)$, which contradicts Lemma 6.3.11. It follows that $X_kY_k < \overline{X_kY_k}$. But $X_kY_k$ is not a prefix of $\overline{X_kY_k}$ because $Y_k$ is not a piece, and so $X_kY_kZ_k < \overline{X_kY_kZ_k}$.

Suppose that $X''_k$ is a proper suffix of $X_k$. It follows by Lemma 6.3.8 that $a_k = b_k = Z_{k-1}$, and so $u = pX''_{k-1}Y_{k-1}Z_{k-1}X''_kY_kq$ and $v = pX''_{k-1}Y_{k-1}Z_{k-1}\overline{X''_kY_k}q'$. In this case, as in the previous case, it follows that $X''_kY_k < \overline{X''_kY_k}$. By Lemma 6.3.14, if $X'_k$ and $\overline{X'_k}$ are such that $X_k = X'_kX''_k$ and $\overline{X_k} = \overline{X'_kX''_k}$, then $X'_k = \overline{X'_k}$. Hence $X_kY_kZ_k = X'_kX''_kY_kZ_k < X'_k\overline{X''_kY_kZ_k} = \overline{X_kY_kZ_k}$. □

We are now ready to prove the correctness of the **NormalForm** algorithm. We have already shown that the output of **NormalForm**$(w_0)$ is a word equivalent to $w_0$ in Corollary 6.2.2. By Theorem 6.3.4 it suffices to show that if $v_n$ is the output of **NormalForm**$(w_0)$, then the sequences of special middle words of $v_n$ and $\min w_0$ are identical. We accomplish this in the next two lemmas.

**Lemma 6.3.16.** *Let $w_0$ be the input to **NormalForm**$(w_0)$ and let $v_i$ with $i \geq 0$ be the value of $v$ after the $i$-th iteration of the while loop starting in line 2. Then for every special middle word $Y_k$ in $w_0$ there exists an $i$ such that $v_i$ contains a complement of $Y_k$.*

**Proof.** All line numbers in this proof refer to **NormalForm** in Algorithm 2. We will show that there exists an $i$ such that during the $i$-th iteration of the while loop in **NormalForm** one of the following holds:

(i) $\overline{Y_k} = Y_s$ in line 3 and all conditions of line 3 are satisfied; or
(ii) $\overline{Y_k} = Y$ in line 18.

If (i) holds, then $v_{i+1}$ is assigned in line 11 or 15 and $v_{i+1}$ contains a complement of $Y_k$. If (ii) holds, then $v_{i+1}$ is assigned in line 21 or 25 and $v_{i+1}$ contains a complement of $Y_k$.

We proceed by induction on the number of special middle words in $w_0$. By Theorem 6.3.4, if there are no special middle words in $w_0$, then the only word equivalent to $w_0$ is itself. In particular, if there are no special middle words in $w_0$, then neither $w_0$ nor any word equivalent to $w_0$ contains a relation word as a factor, by Lemma 6.3.5. It follows that either $w_0$ does not have a clean overlap prefix or if $w_0$ has a clean overlap prefix $cXY$ for some $c \in A^*$ such that $w_0 = cXYw'$ for $w' \in A^*$, then **WpPrefix**$(w', w', Z)$=No because otherwise there exists a word equivalent to $w_0$ that contains $XYZ$ as a factor in the obvious place. This is a contradiction since there are no special middle words in $w_0$. It follows that in every iteration of the while loop in line 2 the conditions of line 3 are not satisfied. If the conditions of line 18 are satisfied, then the condition of line 19 is satisfied as well and if the conditions of line 18 are not satisfied then we have the case of lines 29-30. In particular, in every iteration of the while loop in line 2 $v_iw_i = w_0$ and hence the algorithm returns $w_0$, as required.

Suppose that $w_0$ contains at least one special middle word. Let $w_0 = a_0X_0Y_0q$ where $Y_0$ is the left most special middle word of $w_0$ and $a_0, q \in A^*$. Since $Y_0$ is a special middle word, **WpPrefix**$(q, q, Z_0)$=Yes by definition and hence $X_0Y_0$ is a clean overlap prefix of $X_0Y_0q$ by Lemma 6.3.7 (ii). Since $Y_0$ is the left most special middle word of $w_0$, the algorithm begins by finding the clean overlap prefix $aXY$ of $w_0$ such that $w_0 = aXYw'$. If $aXY = a_0X_0Y_0$, then (ii) holds for $Y_0$. If $aXY \neq a_0X_0Y_0$, then $|aXY| < |a_0X_0Y_0|$ because otherwise $X_0Y_0$ would be a factor of $aXY$, which contradicts the fact that $X_0Y_0$ is a clean overlap prefix of $X_0Y_0q$. In addition, $X_0Y_0$ does not overlap with a suffix of $aXY$ because $aXY$ is clean. In this case, $aXY$ satisfies the conditions of line 18 and 19 and $w_1 = w'$. Since $W$ is assigned to be equal to $\varepsilon$ in line 20 and since no clean overlap prefix of $w'$ overlaps with $X_0Y_0$, the same steps as in the first iteration of the while loop are repeated until $w_i = bX_0Y_0w_i'$ for some $b, w_i'$ such that (ii) is satisfied for $Y_0$.

We assume that (i) or (ii) holds for the special middle words $Y_0, \dots, Y_{k-1}$ of $w_0$ for some $k \geq 1$. We will show that (i) or (ii) holds for $Y_k$ also.

Let $v_jw_j$ be the word equivalent to $w_0$ after the $j$-th iteration of the while loop in line 2.

Suppose that either (i) or (ii) was satisfied for $Y_{k-1}$ during the $j$-th iteration of the while loop. In this case, $v_j$ is defined in one of lines 11, 15, or 25, and in any of these cases:

$$v_j = p\overline{X''_{k-1}Y_{k-1}} \text{ and } w_j = b_k\overline{X''_kY_k}q$$

by Lemma 6.3.8 for some $p, b_k, q \in A^*$, $\overline{X''_{k-1}}$ and $\overline{X''_k}$ are suffixes of $\overline{X_{k-1}}$ and $\overline{X_k}$, respectively, and $\overline{Z_{k-1}}$ is a prefix of $b_k$ since $w_j$ was assigned in one of lines 12, 16 or 26. Since $\overline{Y_k}$ is a special middle word of $v_jw_j$ it follows by definition that **WpPrefix**$(q, q, \overline{Z_k})$ =Yes. In addition, by Lemma 6.3.8, either $\overline{X''_k} = \overline{X_k}$; or $\overline{X''_k}$ is a proper suffix of $\overline{X_k}$ and $b_k = \overline{Z_{k-1}}$. If $\overline{X''_k}$ is a proper suffix of $\overline{X_k}$ and $b_k = \overline{Z_{k-1}}$, then, since $\overline{Y_k}$ is a special middle word of $v_jw_j$, $\overline{X''_kY_k}q$ is either $\overline{Z_{k-1}}$-active or $Z$-active for some proper complement $Z$ of $\overline{Z_{k-1}}$ since there exists a word equivalent to $v_jw_j$ containing $\overline{X_kY_kZ_k}$ as a factor in the obvious place. If the latter holds, then the conditions of line 3 are satisfied. In particular, we have that $W = \overline{X_{k-1}Y_{k-1}Z_{k-1}}$ and that $\overline{Z_{k-1}}$ is a prefix of $w_j$ since $v_j$ was assigned in one of lines 11, 15 or 25. In addition **WpPrefix**$(q, q, \overline{Z_k}) =$ Yes since $\overline{Y_k}$ is a special middle word. Hence (i) holds for $\overline{Y_k}$. If the former holds, then the conditions of line 3 are not satisfied and since $\overline{X''_kY_k}q$ is $\overline{Z_{k-1}}$-active it follows by definition that $z\overline{X_kY_k}$ is a clean overlap prefix of $w_j = b_k\overline{X''_kY_k}q$ for some $z \in A^*$ with $|z| < |b_k|$. Hence (ii) holds for $\overline{Y_k}$.

In the case that $X''_k = X_k$ then $w_j = \overline{Z_{k-1}}c_k\overline{X_kY_k}w'$ for some $c_k \in A^*$. If the conditions of line 3 are satisfied for $w_j$ then $c_k\overline{X_kY_k}w'$ is $Z$-active for some proper complement $Z$ of $\overline{Z_{k-1}}$, $c_k\overline{X_kY_k}w' = X''_sY_sw''$ for some $w'' \in A^*$ and **WpPrefix**$(w'', w'', Z_s)$= Yes and hence there exist $p, q \in A^*$ such that $v_jw_j = pY_sq$ and a word $p'X_sY_sZ_sq'$ with $p'X_s \equiv p, Z_sq' \equiv q$. But this implies that $Y_s$ is a special middle word of $w_0$ that occurs between $Y_{k-1}$ and $Y_k$, a contradiction. It follows that in this case the conditions of line 3 cannot be satisfied. Since $w_j = \overline{Z_{k-1}}c_k\overline{X_kY_k}w'$ and $\overline{Y_k}$ is a special middle word of $v_jw_j$, **WpPrefix**$(w', w', \overline{Z_k}) =$ Yes and hence $\overline{X_kY_k}$ is a clean overlap prefix of $\overline{X_kY_k}w'$ by Lemma 6.3.7 (ii). It follows by the same argument applied to prove the base case that either $c_k\overline{X_kY_k}$ is a clean overlap prefix of $w_j$ or $w_j$ has a clean overlap prefix $cXY$ such that $|cXY| \leq c_k$ and hence (ii) holds for $Y_k$. □

**Lemma 6.3.17.** *Let $v_i$, $w_i$ and $W_i$ be the values of $v$, $w$ and $W$ after the $i$-th iteration of the while loop of line 2 in Algorithm 2. If $W_i = XYZ$ then $Y$ is a special middle word of $v_iw_i$.*

**Proof.** All line numbers in this proof refer to **NormalForm** in Algorithm 2. The value of $W_i$ gets assigned in one of lines 10, 14, 20 and 24. In line 24, $W \leftarrow \varepsilon$ and hence it suffices to examine the cases of lines 10, 14 and 24. In each of these cases, $W_i \leftarrow XYZ$ for some relation word $XYZ$ and $v_i \leftarrow v'Y$ for some $v' \in A^*$. It follows that $Y$ is a factor of $v_iw_i$.

We prove that $Y$ is a special middle word of $v_iw_i$ by induction. Assume that the $k$-th iteration of the while loop of line 2 is the first iteration of **NormalForm**$(w_0)$ such that a value not equal to $\varepsilon$ gets assigned to $W_k$. It follows that $W_{k-1}$ did not satisfy the condition of line 3 and the values of $W_k, v_k, w_k$ get assigned in lines 24, 25 and 26, respectively. Hence $w_{k-1}$ has a clean overlap prefix $aXY$ for some $a \in A^*$, $w_{k-1} = aXYw'$ and $Z$ is a possible prefix of $w'$. It follows that $Y$ is a factor of $v_{k-1}w_{k-1} \equiv v_{k-1}aXYZq \equiv$

$v_{k-1}aX'Y'Z'q = v_k w_k$ for $X'Y'Z'$ the lexicographically minimal equivalent of $XYZ$ and for some $q \in A^*$. Hence, by definition, $Y'$ is a special middle word of $v_k w_k$.

We now assume that the result holds for the first $j$ iterations of the while loop of line 2 and assume that $m \in \mathbb{N}$ is such that the $(j+m)$-th iteration of the while loop of line 2 is the first iteration after the $j$-th iteration of **NormalForm**$(w_0)$ such that a value not equal to $\varepsilon$ gets assigned to $W_{j+m}$. Then the value of $W_{j+m}$ gets assigned in one of lines 10, 14 or 24. In the cases of lines 10 and 14, $W_{j+m-1} \neq \varepsilon$ and hence $W_{j+m-1} = W_{j-1}$ and $W_{j+m} = W_j$. It follows that $W_{j-1} = X_r Y_r Z_r$ and $v_{j-1}w_{j-1} = pY_r q$ for some $p, q \in A^*$ and $v_{j-1}w_{j-1} \equiv p'X_r Y_r Z_r q'$ for some $p', q' \in A^*$. Since the value of $W_j$ gets assigned in one of lines 10 and 14, $w_{j-1}$ satisfies the conditions of line 3. In particular, $w_{j-1} = Z_r w'$, $w'$ is $\overline{Z_r}$-active, for $\overline{Z_r}$ a proper complement of $Z_r$ and $\overline{Z_r}w' = aX_s Y_s w''$ and $Z_s$ is a possible prefix of $w''$. It follows that $v_{j-1}w_{j-1} \equiv p'\overline{X_r Y_r Z_r}q' = p'\overline{X_r Y_r}aX_s Y_s w''$. Since $Z_s$ is a possible prefix of $w''$, it follows that $X_s Y_s Z_s$ is a factor of a word equivalent to $v_{j-1}w_{j-1}$ and hence $Y_s$ is a special middle word of $v_{j-1}w_{j-1}$. If the values of $W_j, v_j, w_j$ get assigned in lines 14, 15 and 16, respectively, then $v_{j-1}w_{j-1} = v_j w_j$, $W_j = X_s Y_s Z_s$ and hence $Y_s$ is a special middle word of $v_j w_j$. If the values of $W_j, v_j, w_j$ get assigned in lines 10, 11 and 12, respectively, then $W \leftarrow X_t Y_t Z_t$ is a complement of $X_s Y_s Z_s$ and since $Y_s$ is a special middle word of $v_{j-1}w_{j-1}$, it follows by Lemma 6.3.5 that $Y_t$ is a special middle word of $v_j w_j$.

In the case of line 24 the result follows by an argument that is identical to the argument in the proof of the base case of this proof. □

**Lemma 6.3.18.** *Let $w_0$ be the input to **NormalForm**$(w_0)$ and let $v_i$ with $i \geq 0$ be the value of $v$ after the $i$-th iteration of the while loop in line 2. Then $v_i$ is a prefix of the lexicographically minimal word* $\min w_0$ *equivalent to $w_0$ for every $i$.*

**Proof.** All line numbers in this proof refer to **NormalForm** in Algorithm 2. Certainly, $v_0 = \varepsilon$ is a prefix of the lexicographically least word $\min w_0$ equivalent to $w_0$. Assume for $j \geq 1$ that $v_{j-1}$ is a prefix of $\min w_0$. We will show that $v_j$ is also a prefix of $\min w_0$. Since $v_{j-1}$ is a prefix of $\min w_0$, the special middle words in $v_{j-1}$ are the initial $k$ special middle words in $\min w_0$ for some $k$. The value of $v_j$ is assigned in one of lines 11, 15, 21, 25, and 29 and in every case $v_{j-1}$ is a prefix of $v_j$. We consider each of these cases separately.

**line 11:** If $v_j$ is defined in line 11, then the conditions of lines 3, 4 and 9 are satisfied. Since the conditions of line 3 are satisfied, $W_{j-1} = X_r Y_r Z_r$ and by Lemma 6.3.17, $Y_r$ is a special middle word of $v_{j-1}w_{j-1}$. Since the value of $W_{j-1}$ is assigned such that $Y_r$ is a suffix of $v_{j-1}$, it follows that $Y_r = Y_{k-1}$. In addition, since the value of $W_j$ gets assigned in line 10, it follows by Lemma 6.3.17 that $Y_s = Y_k$ in line 3 and $Y_t = \overline{Y_k}$ in line 11. It suffices by Lemma 6.3.13 to prove that the special middle word $\overline{Y_k}$ in $v_j$ is equal to the complement of $\overline{Y_k}$ in $\min w_0$.

Since $\min w_0 \equiv v_j w_j$, it follows that $\min w_0 \leq v_j w_j$. If the $k+1$-th special middle word $\overline{\overline{Y_k}}$ in $\min w_0$ is not $\overline{Y_k}$, then $\min w_0 < v_j w_j$, and so, by Lemma 6.3.15, $\overline{\overline{X_k Y_k Z_k}} <$

$\overline{X_kY_kZ_k}$. If $a$ is the suffix of $\overline{Z_{k-1}}$ given in line 3, then $\overline{X_kY_kZ_k}$ is chosen in line 5 to be the least complement of $X_kY_kZ_k$ with prefix $a$. Seeking a contradiction we will show that $\overline{\overline{X_kY_kZ_k}}$ also has a prefix $a$. In order to accomplish this, we show that $\min w_0$ and $v_jw_j$ satisfy the assumption of Lemma 6.3.14. In other words, we will show that $v_jw_j = pY_{k-1}Z_{k-1}\overline{X''_kY_k}q$ for some $p, q \in A^*$ and some suffix $\overline{X''_k}$ of $\overline{X_k}$.

The word $v_{j-1}$ was defined to be $v'Y_{k-1}$ for some $v'$. Since the condition in line 3 holds, $w_{j-1} = Z_{k-1}w'$ for some $w'$, and $\overline{Z_{k-1}}w' = bX_kY_kw''$ for some $b \in A^*$ such that $|b| < |\overline{Z_{k-1}}|$. This implies that $X_k = aX''_k$ where $a$ is the suffix of $\overline{Z_{k-1}}$ given in line 3, and $X''_k$ is a prefix of $w'$. Hence, since $w'$ is $Z_{k-1}$-active, $w' = X''_kY_kw''$ and so

$$v_{j-1}w_{j-1} = v'Y_{k-1}Z_{k-1}w' = v'Y_{k-1}Z_{k-1}X''_kY_kw'',$$

and

$$v_jw_j = v'Y_{k-1}Z_{k-1}\overline{X''_kY_k}q$$

for some $q \in A^*$. Hence, by Lemma 6.3.14, the word $a$ is a prefix of both $\overline{X_k}$ and $\overline{\overline{X_k}}$, giving the required contradiction.

**line 15:** Similar to the case of line 11, $W_{j-1} = X_{k-1}Y_{k-1Z_{k-1}}$, $v_j = v_{j-1}Z_{k-1}X''_kY_k$, and we must show that $Y_k$ is in $\min w_0$. If the conditions of line 3 are satisfied but the conditions of line 4 are not satisfied, then $X_kY_kZ_k$ is the lexicographically minimum relation word with prefix $a$. If $\min w_0$ does not contain $Y_k$, then it contains a proper complement $\overline{Y_k}$. As in the previous case, $v_jw_j = v'Y_{k-1}Z_{k-1}\overline{X''_kY_k}q$, and so by Lemma 6.3.14 (applied to $v_jw_j$ and $\min w_0$) the word $a$ is a prefix of both $X_k$ and $\overline{X_k}$. Thus it follows by Lemma 6.3.15 that $\overline{X_kY_kZ_k} < X_kY_kZ_k$. But in this case $\overline{X_kY_kZ_k}$ is a proper complement of $X_kY_kZ_k$ that has prefix $a$ and the condition of line 4 is satisfied, a contradiction.

If the conditions of lines 3 and 4 are satisfied but the condition of line 9 is not satisfied, then **WpPrefix**$(w_0, v_{j-1}Z_{k-1}\overline{X''_kY_kZ_k}t, \varepsilon)$ = No. Hence no word equivalent to $w_0$ contains $Y_{k-1}$ and a proper complement $\overline{Y_k}$ of $Y_k$ where $\overline{X_kY_kZ_k}$ has prefix $a$. But, by Lemma 6.3.14, every word equivalent to $w_0$ that contains $Y_{k-1}$ and a proper complement of $Y_k$, has the property that the proper complement of $Y_k$ is the middle word of a relation word with prefix $a$. Hence no word equivalent to $w_0$ contains both $Y_{k-1}$ and a proper complement of $Y_k$. In particular, $\min w_0$ contains $Y_k$, as required.

**line 21:** In this case, $v_j = v_{j-1}aXY$, $w_{j-1} = aXYw'$ and **WpPrefix**$(w', w', Z)$=No. It follows that $Y$ is not a special middle word of $v_{j-1}w_{j-1}$. By assumption $v_{j-1}$ is a prefix of $\min w_0$ and by Lemma 2.1.4 $aXY$ is a prefix of all words equivalent to $w_{j-1}$. It follows that $v_j = v_{j-1}aXY$ is a prefix of $\min w_0$.

**line 25:** In this case, $v_j = v_{j-1}a\overline{XY}$, $w_{j-1} = aXYw'$ and **WpPrefix**$(w', w', Z)$ = Yes. It follows that there exists a word equivalent to $v_{j-1}w_{j-1}$ containing $XYZ$ as a factor and hence $Y$ is a special middle word of $v_{j-1}w_{j-1}$. Since $v_{j-1}$ contains the initial $k$ special middle words of $\min w_0$ and $Y$ is a special middle word occurring after $Y_{k-1}$, it follows that $Y = Y_k$. In this case, $v_j = v_{j-1}a\overline{X_k Y_k}$ and by Lemma 6.3.13 it suffices to show that $\overline{Y_k}$ is a factor of $\min w_0$. In this case, $w_{j-1} = b_k X_k Y_k w'$ and $Z_k$ is a possible prefix of $w'$. By Lemma 6.3.15, $\min w_0$ must contain the middle word in $\min X_k Y_k Z_k$. Hence $\min w_0$ contains $\overline{Y_k}$, as required.

**line 29:** In this case, neither of the conditions in lines 3 or 18 are satisfied. Since the condition of line 18 is not satisfied, $w_{j-1}$ contains no relation words as factors and it is only equivalent to itself. Hence $v_j = v_{j-1}w_{j-1}$ is the normal form of $w_0$. □

The proof of the correctness of **NormalForm** is concluded in the following proposition.

**Proposition 6.3.19.** *If $w_0 \in A^*$ is arbitrary, then the word $v$ returned by **NormalForm**$(w_0)$ is the lexicographical least word equivalent to $w_0$.*

**Proof.** All line numbers in this proof refer to **NormalForm** in Algorithm 2. In Corollary 6.2.2 it is shown that the word returned by **NormalForm** is equivalent to $w_0$. We will use the same notation as in the proof of Corollary 6.2.2; $v_i, w_i$ will be used to denote the values of $v$ and $w$ after the $i$-th iteration of the while loop in line 2.

In Lemma 6.3.16, we showed that for every special middle word $Y_k$ in $w_0$ there exists an $i$ such that $v_i$ contains a complement of $Y_k$. Since $v_i$ is a proper prefix of $v_{i+1}$ for every $i$, it follows that eventually $v_i$ contains a complement of every special middle word in $w_0$. In Lemma 6.3.18, we showed that $v_i$ is a prefix of $\min w_0$ for all $i$. Together these two statements imply that when **NormalForm** terminates, the middle words in $v_i$ coincide with the special middle words in $\min w_0$ and hence $v_i$ is the lexicographically least word equivalent to $w_0$ by Theorem 6.3.4. □

## *6.4. Complexity*

In this section we analyze the complexity of **NormalForm**. Throughout this section we suppose that the maximal piece prefix $X$, suffix $Z$, and middle word $Y$ has been computed already for every relation word in the given presentation $\langle A|R\rangle$. The time complexity for doing this is discussed in Section 3. As such we do not include the complexity of determining that the presentation satisfies $C(4)$, nor that of finding the $X$, $Y$, and $Z$, in the statements in this section. We start with two results regarding the complexity of finding a clean overlap prefix for a word $w$ and deciding if a word $w$ is $p$-active for a piece $p$. Finally, we show that for a given C(4) presentation $\langle A \mid R\rangle$, the complexity of **NormalForm**$(w)$ is $O(|w|^2)$ where $w \in A^*$ is the input.

**Lemma 6.4.1.** *If $w \in A^*$ is arbitrary, then the clean overlap prefix of $w$, if any, can be found in time linear to the length of $w$.*

**Proof.** Let $M$ denote the number of relation words and let $\delta$ be the length of the longest relation word in our $C(4)$ presentation. According to Lemma 7 in [8] to check if a word

$v$ has a clean overlap prefix of the form $X_iY_i$ where $X_iY_iZ_i = W_i$, $1 \leq i \leq r$ it suffices to check if $v'$ has a clean overlap prefix of this form, where $v'$ is a prefix of $v$ such that $|v| = 2\delta$.

Hence, in order to find the clean overlap prefix of $w$ that has the form $sX_iY_i$ for $s \in A^*$ it suffices to check at most $|w|$ suffixes of $w$ for clean overlap prefixes of the form $X_iY_i$. This can be done in $O(|w|)$ time. □

**Lemma 6.4.2.** *If $w \in A^*$ is arbitrary and $p$ is a piece, then deciding if $w$ is $p$-active takes constant time.*

**Proof.** Again, let $M$ denote the number of relation words and let $\delta$ be the length of the longest relation word in our $C(4)$ presentation. According to Lemma 7 in [8] it suffices to check if $w'$ is $p$-active, where $w'$ is a prefix of $w$ of length $2\delta$. Since $p$ is a piece, then clearly $|p| < \delta$. A string searching algorithm, such as, for example, Boyer-Moore-Horspool [5], can check if there exists some $i$, $1 \leq i \leq M$ such that the factor $X_iY_i$ occurs in $pw'$ before the end of $p$. This takes $O(M\delta|pw'|) = O(3M\delta^2)$ time. □

**Proposition 6.4.3.** *The complexity of **NormalForm** is $O(|w_0|^2)$ where $w_0 \in A^*$ is the input, given that the decompositions of the relation words in the presentation into $XYZ$ are known.*

**Proof.** Let $\langle A \mid R \rangle$ be the presentation, let $M$ be the number of distinct relation words in $R$ and let $\delta$ be the length of the longest relation word in $R$. We have already shown that the while loop of line 2 will be repeated at most $|w_0|$ times. We analyze the complexity of each step of the procedure in the loop.

In line 3 the algorithm tests if the word $w'$ is $\overline{Z_r}$-active for $\overline{Z_r}$ some complement of $Z_r$ in constant time. In addition, checking if $Z_r \neq \overline{Z_r}$ requires comparing at most $\delta$ characters. Finding the suffix $a$ of $\overline{Z_r}$ such that $aw' = X_sY_sw''$ also requires checking at most $\delta$ characters, hence these checks can be performed in constant time.

In lines 3,9 and 19, **WpPrefix**$(u, v, p)$ is called. According to [8], the algorithm can be implemented with execution time bounded above by a linear function of the length of the shortest of the words $u$ and $v$. Since every time **WpPrefix**$(u, v, p)$ is called either $u = w_0$ or $u$ is a suffix of some word equivalent to $w_0$, this step can be executed in $O(\delta|w_0|)$ time.

In lines 4-5 we search for proper complements of $X_sY_sZ_s$ that have the prefix $a$. Since $a$ is a piece, this step also requires constant time.

In line 9 the algorithm finds the suffix $b$ of $\overline{X_s}$ such that $\overline{X_s} = ab$ and the suffix $t$ of **ReplacePrefix**$(w'', Z_s)$ that follows $Z_s$. This is also done in constant time since $a$ and $Z_s$ are pieces.

In lines 9, 16 and 26 Algorithm 1 is called. Each time, Algorithm 1 takes as input a suffix of some word $s$ equivalent to $w_0$. Since $|s| < \delta|w_0|$, this step can be completed in $O(\delta|w_0|)$ time.

In lines 5 and 25 we search for the lexicographically minimal complement of some relation word $X_iY_iZ_i$. Clearly, this check can be done by comparing at most $\delta$ characters $M$ times, hence it is constant for a given presentation.

In line 18 the algorithm finds the clean overlap prefix of $w$. As shown in Lemma 6.4.1, this can be done in $O(|w|)$ time. Since $w$ is always a suffix of some word equivalent to $w_0$, this step can be executed in $O(\delta|w_0|)$ time.

In lines 11-12, 15, 21-22, 25-26 and 29 Algorithm 2 concatenates $v$ with a word of length at most $\delta|w_0|$ and deletes a prefix of length at most $\delta|w_0|$ from $w$. Hence these steps require at most $2\delta|w_0|$ time. $\square$

We end the paper with an example of the application of **NormalForm** to specific $C(4)$ presentation.

**Example 6.4.4.** Let

$$\langle a, b, c, d \mid ab^3a = cdc \rangle$$

be the presentation and let $w_0 = cdcdcab^3ab^3ab^2cd$. The set of relation words of the presentation is $\{ab^3a, cdc\}$ and each relation word has a single proper complement. The set of pieces of $\mathcal{P}$ is $P = \{\varepsilon, a, b, c, b^2\}$. Let $W_0 = ab^3a$, $W_1 = cdc$. Clearly, $X_{W_0} = a$, $Y_{W_0} = b^3$, $Z_{W_0} = a$ and $X_{W_1} = c$, $Y_{W_1} = d$, $Z_{W_1} = c$.

Algorithm 2 begins with $v \leftarrow \varepsilon$, $W \leftarrow \varepsilon$ and $w \leftarrow cdcdcab^3ab^3ab^2cd$. Since $u = \varepsilon$ the conditions of line 3 are not satisfied. The word $w$ has a clean overlap prefix $X_{W_1}Y_{W_1} = cd$ followed by $Z_{W_1} = c$ hence **WpPrefix**$(cdcab^3ab^3ab^2cd, cdcab^3ab^3ab^2cd, c)$ returns Yes and **ReplacePrefix**$(cdcab^3ab^3ab^2cd, c)$ returns $cdcab^3ab^3ab^2cd$. Since $W_0 < W_1$, $v \leftarrow X_{W_0}Y_{W_0} = ab^3$, $w \leftarrow adcab^3ab^3ab^2cd$ and $W \leftarrow ab^3a$ in lines 24-26.

Now $W = ab^3a$, $w$ begins with $Z_{W_0} = a$, $w' = dcab^3ab^3ab^2cd$ is $\overline{Z_{W_1}}$-active and the prefix $Y_{W_1} = d$ of $w'$ is followed by $Z_{W_1} = c$, hence the conditions in line 3 are satisfied. In addition, $ab^3a < cdc$ but $X_{W_0}$ and $X_{W_1}$ do not have a common prefix. Hence, in lines 14-16 $v \leftarrow ad$, $w \leftarrow$ **ReplacePrefix**$(cab^3ab^3ab^2cd, c) = cab^3ab^3ab^2cd$ and $W \leftarrow cdc$.

At this point, $W = cdc$ and $w = cab^3ab^3ab^2cd$ begins with $Z_{W_1} = c$ but $ab^3ab^3ab^2cd$ is not $Z_{W_0}$-active. The word $w$ has the clean overlap prefix $cab^3$ that is followed by $a$, hence in lines 24-25, $v \leftarrow vcab^3$, $w \leftarrow$ **ReplacePrefix**$(ab^3ab^2cd, a) = ab^3ab^2cd$ and $W \leftarrow ab^3a$.

Next, $W = ab^3a$ and $w = ab^3ab^2cd$ begins with $Z_{W_0}$ and $b^3ab^2cd$ is not $Z_{W_1}$-active but it has the clean overlap prefix $ab^3$. The clean overlap prefix is followed by $a$, hence in lines 24-25 $v \leftarrow vab^3$, $w \leftarrow ab^2cd$ and $W \leftarrow ab^3a$.

Finally, $W = ab^3a$, $w = ab^2cd$ begins with $Z_{W_0}$ but $b^2cd$ is not $Z_{W_1}$-active. Now $w$ has the clean overlap prefix $ab^2cd$ that is followed by $\varepsilon$, hence in line 19 **WpPrefix**$(\varepsilon, \varepsilon, c)$ returns No and in lines 20-22 $v \leftarrow vab^2cd$, $W \leftarrow \varepsilon$ and $w \leftarrow \varepsilon$. Since $w = \varepsilon$, the algorithm returns $v = ab^3adcab^3ab^3ab^2cd$.

Next, we will apply **NormalForm**, to find the normal form of $w_0 = cdab^3cdc$. We begin with $v \leftarrow \varepsilon$, $W \leftarrow \varepsilon$ and $w \leftarrow cdab^3cdc$. Since $W = \varepsilon$ we do not have the case of line 3. The word $w$ has the clean overlap prefix $cd$ of the form $X_{W_1}Y_{W_1}$ and **WpPrefix**$(ab^3cdc, ab^3cdc, c)$=Yes, and **ReplacePrefix**$(ab^3cdc,\ c)$ returns $cdcb^3a$. In lines 24-26 $v \leftarrow ab^3$, $w \leftarrow adcb^3a$ and $W \leftarrow ab^3a$.

For this iteration, $u = ab^3a$ and $w$ begins with $Z_{W_0} = a$, $dcb^3a$ is $Z_{W_1}$-active and clearly **WpPrefix**$(cb^3a, cb^3a, c)$=Yes. In addition, $ab^3a < cdc$ but $X_{W_0}$ and $X_{W_1}$ do not have a common prefix, hence in lines 14-16 $v \leftarrow vad$, $w \leftarrow$ **ReplacePrefix**$(cb^3a, c)$= $cb^3a$ and $W \leftarrow cdc$.

At this stage, $W = cdc$, $w = cb^3a$ begins with $Z_{W_1} = c$ and $b^3a$ is $Z_{W_0}$-active but $X_{W_0}$ and $X_{W_1}$ do not have a common prefix hence $v \leftarrow cb^3$, $w \leftarrow$**ReplacePrefix**$(a,\ a)$ $= a$, $W \leftarrow ab^3a$ in lines 14-16.

Finally, $a$ begins with $Z_{W_0}$ but clearly $\varepsilon$ is not $Z_{W_1}$-active. Also $a$ does not have a clean overlap prefix and in lines 29-30 $v \leftarrow va$, $w \leftarrow \varepsilon$ and the algorithm returns $v = ab^3adcb^3a$.

## Data availability

No data was used for the research described in the article.

## Acknowledgments

The second author would like to thank the School of Mathematics and Statistics of the University of St Andrews and the Cyprus State Scholarship Foundation for their financial support. The authors would also like to thank Mark Kambites for his correspondence on the topic of this paper, and the anonymous referee for their comments that helped to improve the paper.

## Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jalgebra.2023.04.019.

## References

[1] J. Berstel, Transductions and Context-Free Languages, Teubner-Verlag, 1979.
[2] Robert H. Gilman, Presentations of groups and monoids, J. Algebra 57 (2) (1979) 544–554, https://doi.org/10.1016/0021-8693(79)90238-2.
[3] Dan Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, 1997.
[4] P.M. Higgins, Techniques of Semigroup Theory, Oxford Science Publications, Oxford University Press, Incorporated, 1992.
[5] R. Nigel Horspool, Practical fast searching in strings, Softw. Pract. Exp. 10 (6) (1980) 501–506, https://doi.org/10.1002/spe.4380100608.
[6] J.H. Johnson, Do rational equivalence relations have regular cross-sections?, in: Wilfried Brauer (Ed.), Automata, Languages and Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 1985, pp. 300–309.

[7] J. Howard Johnson, Rational equivalence relations, in: Laurent Kott (Ed.), Automata, Languages and Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 1986, pp. 167–176.
[8] Mark Kambites, Small overlap monoids I: the word problem, J. Algebra 321 (2009) 2187, https://doi.org/10.1016/j.jalgebra.2008.09.038.
[9] Mark Kambites, Small overlap monoids II: automatic structures and normal forms, J. Algebra 321 (2009) 2302, https://doi.org/10.1016/j.jalgebra.2008.12.028.
[10] Mark Kambites, Generic complexity of finitely presented monoids and semigroups, Comput. Complex. 20 (1) (March 2011) 21–50, https://doi.org/10.1007/s00037-011-0005-5.
[11] Mark Kambites, A note on the definition of small overlap monoids, Semigroup Forum 83 (2011) 499, https://doi.org/10.1007/s00233-011-9350-6.
[12] Donald E. Knuth, Peter B. Bendix, Simple word problems in universal algebras, in: John Leech (Ed.), Computational Problems in Abstract Algebra, Pergamon, 1970, pp. 263–297.
[13] R.C. Lyndon, P.E. Schupp, Combinatorial Group Theory, Classics in Mathematics, vol. 89, Springer-Verlag, 1977.
[14] J.D. Mitchell, et al., libsemigroups - C++ library for semigroups and monoids, Version 2.7.1, https://doi.org/10.5281/zenodo.1437752, Mar 2023.
[15] A.A. Markov, On the impossibility of certain algorithms in the theory of associative systems, C. R. (Dokl.) Acad. Sci. URSS 55 (1947) 587–590.
[16] Emil L. Post, Recursive unsolvability of a problem of Thue, J. Symb. Log. 12 (1) (1947) 1–11, https://doi.org/10.2307/2267170.
[17] John Hermann Remmers, Some Algorithmic Problems for Semigroups: A Geometric Approach, PhD thesis, University of Michigan, USA, 1971, AAI7123856.
[18] Charles C. Sims, Computation with Finitely Presented Groups, Encyclopedia of Mathematics and Its Applications, Cambridge University Press, 1994.
[19] J.A. Todd, H.S.M. Coxeter, A practical method for enumerating cosets of a finite abstract group, Proc. Edinb. Math. Soc. 5 (1) (1936) 26–34, https://doi.org/10.1017/S0013091500008221.
[20] E. Ukkonen, On-line construction of suffix trees, Algorithmica 14 (3) (September 1995) 249–260, https://doi.org/10.1007/BF01206331.