



Contents lists available at ScienceDirect

Journal of Algebra

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



The fully compressed subgroup membership problem

Marco Linton

University of Oxford, Mathematical Institute, Andrew Wiles building, Oxford,
OX26GG, United Kingdom

ARTICLE INFO

Article history:

Received 1 December 2021

Available online 5 April 2023

Communicated by Derek Holt

Keywords:

Free groups

Compression

Straightline programs

Involutive automata

Membership problem

Stallings folding

ABSTRACT

Suppose that F is a free group and k is a natural number. We show that the fully compressed membership problem for k -generated subgroups of F is solvable in polynomial time. In order to do this, we adapt the theory of Stallings' foldings to handle edges with compressed labels. This partially answers a question of Markus Lohrey.

Crown Copyright © 2023 Published by Elsevier Inc. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The rational subset membership problem for free monoids is a classic problem in formal language theory. This problem naturally extends to the world of group theory as follows. Let Σ be a finite set, let G be a group and $\pi : \Sigma^* \rightarrow G$ a surjective morphism. Then the *rational subset membership problem* for G is to decide, given as input a finite

E-mail address: marco.linton@maths.ox.ac.uk.

state automaton \mathcal{A} over Σ and a word $w \in \Sigma^*$, whether $\pi(w) \in \pi(L(\mathcal{A}))$. This problem is known to be solvable when G is free [3], abelian [5] or a right angled Artin group whose defining graph is a transitive forest [15]. The first two classes also admit polynomial time solutions. An important class of rational subsets of groups is that of finitely generated subgroups. The *subgroup membership problem* is the rational subset membership problem when restricted to this class. The full rational subset membership problem is strictly harder: the subgroup membership problem is solvable for nilpotent groups [2], but there are nilpotent groups in which the rational subset membership problem is undecidable [19].

Often, requiring inputs to be words over the generators may not be the most natural thing to do. Take, for instance, linear groups over \mathbb{Z} . Each element of a linear group over \mathbb{Z} can be represented by a matrix with binary integer entries, an exponentially more succinct representation. In [6], a variant of the subgroup membership problem was considered where the inputs could contain powers x^i where $x \in \Sigma$ and $i \in \mathbb{Z}$ is encoded in binary. The authors show that this variant remains solvable in polynomial time in the class of free groups. As a consequence, they also show that the subgroup membership problem for $\text{PSL}(2, \mathbb{Z})$ is solvable in polynomial time, even when the input matrix entries are encoded in binary. In [14], the subgroup membership problem for free groups was further generalised so that the input could contain power words w^i where $w \in \Sigma^*$ and $i \in \mathbb{Z}$ is encoded in binary. The polynomial time solution to this variant was similarly used to provide a polynomial time solution to the membership problem in $\text{GL}(2, \mathbb{Z})$, even when the input matrix entries are encoded in binary. Another version of the compressed membership problem can be found in [16], arising as an intermediate step to solve the word problem in polynomial time in the Baumslag–Gersten group.

A more general way of succinctly representing elements of Σ^* is given by *straight-line programs* (SLPs): context free grammars that generate exactly one word. In [17], it was asked whether the rational subset membership problem for free monoids was solvable in polynomial time when the transition labels of the automaton, as well as the input word that is tested for membership, are given by SLPs. In [8], Artur Jež settled this question in the affirmative. In [14] and [13], Markus Lohrey asked whether the membership problem for finitely generated subgroups of a free group is solvable in polynomial time, when all group elements are represented by SLPs. We call this the *fully compressed subgroup membership problem* for free groups. In the case that the number of input generators is fixed, we answer Markus Lohrey’s question in the affirmative.

Corollary 5.9. *The fully compressed membership problem for k -generated subgroups of a free group is in P .*

This result follows directly from our stronger result, Theorem 5.8. See Algorithm 1 for the sketch of our compressed Stallings’ folding algorithm. Note that by [1], the fully compressed membership problem for free groups is P-hard.

2. Preliminaries

We will fix Σ to be a finite alphabet and Σ^* to be the free monoid generated by Σ . A *letter* is an element of Σ ; a *word* is an element of Σ^* . The symbol ϵ will denote the *empty word*.

A *factorisation* of a word $w \in \Sigma^*$ is an equality $w = w_0 \cdot w_1 \cdot \dots \cdot w_n$ where $w_i \in \Sigma^*$ for all $0 \leq i \leq n$. Now let $w = w_0 \cdot w_1 \cdot \dots \cdot w_n \in \Sigma^*$ be the unique factorisation with $w_i \in \Sigma$. Given $0 \leq i \leq j \leq n$, we make the following definitions:

1. $w[i] = w_i$ is the $i + 1^{\text{th}}$ letter of w ,
2. $|w| = n + 1$ is the *length* of w ,
3. $w[i : j] = w_i \cdot w_{i+1} \cdot \dots \cdot w_{j-1}$ is a *subword*,
4. $w[: i] = w_0 \cdot w_1 \cdot \dots \cdot w_{i-1}$ is a *prefix*,
5. $w[j :] = w_j \cdot w_{j+1} \cdot \dots \cdot w_n$ is a *suffix*.

Additionally, for $0 < i \leq n + 1$, we define $w[-i] = w[n + 1 - i]$, the i^{th} letter of w from the end. We also define $w[: -i] = w[: n + 1 - i]$ and $w[-i :] = w[n + 1 - i :]$. A word w is *primitive* if it is not equal to a proper power $w = u^n$ where $n \geq 2$.

Let $x, w \in \Sigma^*$ be words, a *left w -factorisation of length k* of x is a factorisation of the form:

$$x = w[i :] \cdot w^n \cdot w[: j] \cdot z$$

such that $|w[i :] \cdot w^n \cdot w[: j]| = k$. We define *right w -factorisations* analogously. The following is Lemma 2.6 from [11].

Lemma 2.1. *Let $x, w \in \Sigma^*$ be two words and suppose that w is primitive. Then if x has a left (or right) w -factorisation of length $k \geq |w|$, then x has a unique maximal left (or right) w -factorisation.*

We write $\text{Arith}(j, k, l)$ for the arithmetic progression $\{i \cdot j + l \mid 0 \leq i \leq k\}$.

Let Σ^{-1} denote the set of formal inverses of Σ . The free group $F(\Sigma)$, freely generated by Σ , comes equipped with a natural surjective monoid homomorphism $\pi : (\Sigma \sqcup \Sigma^{-1})^* \rightarrow F(\Sigma)$. Here, the symbol \sqcup denotes the disjoint union of sets. The map π has a section $\text{red} : F(\Sigma) \rightarrow (\Sigma \sqcup \Sigma^{-1})^*$ mapping each element $g \in F(\Sigma)$ to the unique freely reduced word in $\pi^{-1}(g)$.

2.1. Compression

We refer the reader to [12] for an excellent survey on straight-line programs and compressed finite state automata. A *straight-line program*, or SLP, is a tuple $\mathbb{X} = \langle \Sigma, \mathcal{X}, X_n, \mathcal{P} \rangle$ consisting of the following:

1. Σ is a finite alphabet of *terminal* letters,
2. $\mathcal{X} = \{X_1, \dots, X_n\}$ is a finite alphabet of *non-terminal* letters,
3. X_n is the *root* non-terminal,
4. $\mathcal{P} = \{X_i \rightarrow W_i \mid i = 1, \dots, n\}$ is the set of *production rules* where $W_i \in (\Sigma \sqcup \{X_1, \dots, X_{i-1}\})^*$.

We inductively define $\text{word}(X_i) \in \Sigma^*$ to be the word obtained by replacing each non-terminal X_j appearing in W_i with $\text{word}(X_j)$. Then define $\text{word}(\mathbb{X}) = \text{word}(X_n)$. The *height* of a non-terminal $X_i \in \mathcal{X}$, denoted by $\|X_i\|$, is inductively defined as the maximum height of the symbols appearing in W_i plus one, where the height of a terminal is understood to be zero.

The *size* of the SLP \mathbb{X} is the quantity

$$|\mathbb{X}| = |\Sigma| + |\mathcal{X}| + \sum_{i=1}^n |W_i|.$$

If \mathbb{X} is part of an input to an algorithm, then \mathbb{X} contributes $|\mathbb{X}|$ to the input size.

If X is a non-terminal and i and j are positions in $\text{word}(X)$, then we may write $X[i : j]$; we call this a *truncated non-terminal*. The intention is that we have $\text{word}(X[i : j]) = \text{word}(X)[i : j]$. A *composition system* $\mathbb{X} = \langle \Sigma, \mathcal{X}, X_n, \mathcal{P} \rangle$ is defined in the same way as an SLP except that production rules can contain truncated non-terminals. The expressive power of SLPs and composition systems is virtually the same; this is because a composition system can be transformed into an equivalent SLP in quadratic time [7]. We define the size of a composition system in the same way as the size of an SLP.

We abuse notation and factorise SLPs just as we factorise words. That is, if W is a non-terminal, then we write $W = W_0 \cdot W_1 \cdot \dots \cdot W_n$ when we mean $\text{word}(W) = \text{word}(W_0) \cdot \text{word}(W_1) \cdot \dots \cdot \text{word}(W_n)$. Thus, if X and W are non-terminals, then a *left W -factorisation* of X is a factorisation of the form:

$$X = W[i :] \cdot W^n \cdot W[: j] \cdot Z$$

where Z is also a non-terminal.

Proposition 2.2. *There is a polynomial-time algorithm that, given as input two non-terminals X and W , decides if X has a left (or right) W -factorisation of length $k \geq |\text{word}(W)|$. If it does, then the algorithm also computes a maximal left (or right) W -factorisation of X .*

Proof. We prove the result for left W -factorisations. By Theorem 2 in [10], we may introduce a new non-terminal U and compute an integer n and a rule for U in polynomial-time such that $\text{word}(W) = \text{word}(U)^n$ and $\text{word}(U)$ is primitive. Now X has a left W -factorisation of length $k \geq |\text{word}(W)|$ if and only if X has a left U -factorisation

of length $k \geq |\text{word}(W)| \geq |\text{word}(U)|$. Since maximal left U -factorisations of X will correspond to maximal W -factorisations of X , it suffices to prove the result for U . Thus, we now assume that $\text{word}(W)$ is primitive.

First assume that $|\text{word}(X)| = |\text{word}(W)|$. Hence the problem becomes to decide if there is some integer i such that

$$X = W[i:] \cdot W[:i]. \quad (1)$$

This is the conjugacy problem and can be solved in polynomial time by Theorem 3.7 in [20]. This algorithm also produces a conjugating word. Equivalently, this algorithm produces an index i such that (1) holds. By Lemma 2.1, this is unique. So now suppose $|\text{word}(X)| > |\text{word}(W)|$. By the above, we can decide if $X[:|\text{word}(W)|]$ has a W -factorisation of length $|\text{word}(W)|$. Moreover, if so, we may also compute an index i such that $X[:i] = W[-i:]$. Since this i is unique, a maximal left W -factorisation must be an extension of this. Then by computing the largest prefix that $X[i:]$ and $W\left[\frac{|\text{word}(X)|}{|\text{word}(W)|}\right]$ have in common gives us the required factorisation. This may be done in polynomial time; see [4] or Theorem 2.9 in [20]. \square

Let $x, w \in \Sigma^*$ and let i be an integer. We will say x crosses w at i if there is an integer j such that $j \leq i < j + |x|$ and

$$w[j:j+|x|] = x.$$

Then we denote by $\text{Sub}(x, w, i)$ the set of such integers j . If X and W are non-terminals, then for ease of notation we write $\text{Sub}(X, W, i)$ when we mean $\text{Sub}(\text{word}(X), \text{word}(W), i)$. We shall need the following useful result, Theorem 2.17 in [11].

Theorem 2.3. *Let $v, x_1, \dots, x_n \in \Sigma^*$ be words with v primitive. There exist words $v_1, v_2, w \in \Sigma^*$ such that $v = v_1 \cdot v_2$, $w = v_2 \cdot v_1$ and:*

$$|\text{Sub}(x_i, w^2, |w|)| \leq 2$$

for all $i \leq n$.

We say a non-terminal X crosses the production $W \rightarrow U \cdot V$, if

$$X = U[-i_1:] \cdot V[:i_2]$$

for some $i_1 > 0$, $i_2 > 0$, or if X is a prefix of V . The following is Lemma 1 in [10].

Lemma 2.4. *There is a polynomial-time algorithm that, given as input non-terminals X and W and an integer $i \leq |\text{word}(W)|$, computes integers j, k, l such that $\text{Sub}(X, W, i) = \text{Arith}(j, k, l)$.*

2.2. Compressed automata

A *compressed non-deterministic finite state automaton*, or CNFA, is a tuple $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, \mathcal{P}, q_0, F)$, where:

1. Q is a finite set of *states*,
2. Σ is a finite alphabet of terminal letters,
3. $\mathcal{X} = \{X_1, \dots, X_n\}$ is a finite alphabet of non-terminal letters,
4. $\delta \subseteq Q \times \mathcal{X} \times Q$ is the set of *transitions*,
5. $\mathcal{P} = \{X_i \rightarrow W_i \mid i = 1, \dots, n\}$ is the set of *production rules* where $W_i \in (\Sigma \sqcup \{X_1, \dots, X_{i-1}\})^*$,
6. $q_0 \in Q$ is the *initial state*,
7. $F \subseteq Q$ is the set of *final states*.

A CNFA \mathcal{A} is a *compressed deterministic finite state automaton*, or CDFA, if $\text{word}(X) \neq \epsilon$ for all $X \in \mathcal{X}$ and if for each pair of transitions $\alpha = (p, X, q)$, $\beta = (p, Y, r) \in \delta$, $X[0] = Y[0]$ implies that $\alpha = \beta$. Often we will allow our productions to also contain truncated non-terminals. This only affects the run-time of any algorithm presented by a quadratic factor.

The *size* of the CNFA \mathcal{A} is the quantity

$$|\mathcal{A}| = |Q| + |\delta| + |\Sigma| + |\mathcal{X}| + \sum_{i=1}^n |W_i|.$$

If the CNFA \mathcal{A} is part of an input to an algorithm, then \mathcal{A} will contribute $|\mathcal{A}|$ to the input size.

A word $w \in \Sigma^*$ is *accepted* by \mathcal{A} if there is a sequence of states q_0, q_1, \dots, q_k with $q_k \in F$, such that $(q_{i-1}, X_i, q_i) \in \delta$ for all $1 \leq i \leq k$ and $w = \text{word}(X_1) \cdot \text{word}(X_2) \cdot \dots \cdot \text{word}(X_k)$. The *language* $L(\mathcal{A})$ of \mathcal{A} is the set of all words accepted by \mathcal{A} .

The *fully compressed membership problem for CDFA (respectively CNFA)* is the problem of deciding whether $\text{word}(\mathbb{X}) \in L(\mathcal{A})$ for a given SLP \mathbb{X} and CDFA (respectively CNFA) \mathcal{A} . The key result we will be using throughout this article is the main result from [8]:

Theorem 2.5. *The fully compressed membership problem for CDFA (respectively CNFA) is in P (respectively NP).*

We will say that a non-terminal X *determines a path* from $\alpha(i)$ to $\beta(j)$ in \mathcal{A} , where $\alpha, \beta \in \delta$, if there is a sequence of transitions

$$\alpha = (p_0, X_1, p_1), (p_1, X_2, p_2), \dots, (p_{n-1}, X_n, p_n) = \beta$$

such that

$$\text{word}(X) = \text{word}(X_1)[i:] \cdot \text{word}(X_2) \cdot \dots \cdot \text{word}(X_n[:j]).$$

An *involutive CNFA* is a CNFA $\mathcal{A} = (Q, \Sigma \sqcup \Sigma^{-1}, \mathcal{X}, \delta, \mathcal{P}, q_0, F)$ equipped with an involution $^{-1} : \delta \rightarrow \delta$ such that $(p, X, q)^{-1}$ is of the form (q, Y, p) with $\text{word}(X) = \text{word}(Y)^{-1}$ for all $(p, X, q) \in \delta$. An involutive CNFA \mathcal{A} is an *involutive CDFA* if it is a CDFA and if $\text{word}(X)$ is a freely reduced word for each $X \in \mathcal{X}$. Denote by $\partial\mathcal{A} \subseteq Q$ the set of states with at most one incoming and at most one outgoing transition.

A subset $H \subseteq F(\Sigma)$ of a free group is a *rational subset* if there exists a CNFA \mathcal{A} over $\Sigma \sqcup \Sigma^{-1}$ satisfying $H = \pi(L(\mathcal{A}))$. An important class of rational subsets of free groups are their finitely generated subgroups. The well developed theory of *Stallings automata* allows us to solve the membership problem for subgroups efficiently [21,9]. However, for our context, we will need to define a compressed analogue.

Definition 2.6. A *compressed Stallings automaton* for a subgroup $H \leq F(\Sigma)$ is an involutive CDFA \mathcal{A} such that $(\text{red} \circ \pi)(L(\mathcal{A})) = \text{red}(H)$ and such that the number of states and transitions are minimal.

3. Language intersections for CDFA

The classic Rabin–Scott construction [18] provides an algorithm to compute the intersection of two regular languages in polynomial time. In the compressed setting, there is no analogue. However, in this section, we show that for two special cases, given two CDFA \mathcal{A} and \mathcal{B} , we may compute $L(\mathcal{A}) \cap L(\mathcal{B})$ in polynomial time. We first consider the case that $\Sigma = \{a\}$.

Proposition 3.1. *There is a polynomial-time algorithm that, given as input CDFA \mathcal{A} and \mathcal{B} over a unary alphabet $\Sigma = \{a\}$, computes integers $m_1, \dots, m_k, n_0, \dots, n_l, n$ such that:*

$$L(\mathcal{A}) \cap L(\mathcal{B}) = (a^{m_1} \mid \dots \mid a^{m_k}) \mid a^{n_0} \cdot (a^n)^* \cdot (a^{n_1} \mid \dots \mid a^{n_l}).$$

Proof. A CDFA over a unary alphabet must have at most one outgoing transition from each state. Thus, there are integers $p_1, \dots, p_r, q_0, \dots, q_s, q$ such that:

$$L(\mathcal{A}) = (a^{p_1} \mid \dots \mid a^{p_r}) \mid a^{q_0} \cdot (a^q)^* \cdot (a^{q_1} \mid \dots \mid a^{q_s}),$$

where $p_1, \dots, p_r < q_0$ and $q_1, \dots, q_s < q$. Similarly for $L(\mathcal{B})$. By Lemma 2.7 in [20], for each SLP \mathbb{X} over $\{a\}$, we may compute an integer i such that $\text{word}(\mathbb{X}) = a^i$. Thus, we may compute the integers $p_1, \dots, p_r, q_0, \dots, q_s, q$ in polynomial time. Similarly, we may compute the integers that describe $L(\mathcal{B})$ in polynomial time. Now computing the integers $m_1, \dots, m_k, n_0, \dots, n_l, n$ required involves solving some systems of linear equations which may be done in polynomial time. \square

Just like the unary language case, regular sublanguages of languages of the form $u \cdot (v)^*$, where $u, v \in \Sigma^*$, have very convenient representations using integers, along with our original words u and v . The second case we consider is when $L(\mathcal{B})$ is of this form. First, we shall need a useful lemma.

Lemma 3.2. *Let $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, \mathcal{P}, q_0, F)$ be a CDFA and let $u, v \in \Sigma^*$ such that v is primitive and $|v| > |\text{word}(X)|$ for all $X \in \mathcal{X}$. If $L(\mathcal{A}) \cap u \cdot (v)^+ \neq \emptyset$, then $u \cdot v^k \in L(\mathcal{A})$ for some $1 \leq k \leq 2 \cdot (|\delta| + 1)$.*

Proof. Let $\mathcal{X} = \{X_1, \dots, X_n\}$. By Theorem 2.3 there are words $v_1, v_2, w \in \Sigma^*$ such that $v = v_1 \cdot v_2$, $w = v_2 \cdot v_1$ and $|\text{Sub}(\text{word}(X_i), w^2, |w|)| \leq 2$ for all i . We may assume that $v_2 \neq \epsilon$. This implies that for any given $\alpha = (p, X, q) \in \delta$, if $u \cdot v_1 \cdot w^m \cdot v_2 \in L(\mathcal{A})$, then the words $u \cdot v_1 \cdot w$, $u \cdot v_1 \cdot w^2$, \dots , $u \cdot v_1 \cdot w^m$ determine paths from q_0 to $\alpha(i)$ for at most two distinct values of i . So if $m > 2 \cdot |\delta|$, by the pigeonhole principle there must be some transition $\alpha = (p, X, q) \in \delta$ and integers $i < |\text{word}(X)|$, $j < l \leq 2 \cdot |\delta| + 1$ such that $u \cdot v_1 \cdot w^j$ and $u \cdot v_1 \cdot w^l$ determine a path from q_0 to $\alpha(i)$. As \mathcal{A} is a CDFA, there must be some $1 \leq k \leq l + 1 \leq 2 \cdot (|\delta| + 1)$, such that $u \cdot v_1 \cdot w^{k-1} \cdot v_2 \in L(\mathcal{A})$, otherwise $u \cdot (v)^+ \cap L(\mathcal{A}) = \emptyset$. Hence, $u \cdot v^k \in L(\mathcal{A})$. \square

Note that the proof of Lemma 3.2 does not use the fact that \mathcal{A} has compressed transition labels.

Theorem 3.3. *There exists a polynomial-time algorithm that, given as input a CDFA $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, \mathcal{P}, q_0, F)$ and SLPs \mathbb{U} and \mathbb{V} , computes a collection of integers $\{m_1, \dots, m_k, n_1, \dots, n_l, n\}$ with $k \leq |F|$ such that*

$$L(\mathcal{A}) \cap \text{word}(\mathbb{U}) \cdot (\text{word}(\mathbb{V}))^* = \text{word}(\mathbb{U}) \cdot ((\text{word}(\mathbb{V})^{m_1} \mid \dots \mid \text{word}(\mathbb{V})^{m_k}) \mid ((\text{word}(\mathbb{V})^n)^* \cdot (\text{word}(\mathbb{V})^{n_1} \mid \dots \mid \text{word}(\mathbb{V})^{n_l}))).$$

Proof. By Theorem 2 in [10], we may compute an SLP \mathbb{Y} and an integer $i \geq 1$ such that $\text{word}(\mathbb{Y})$ is primitive and $\text{word}(\mathbb{V}) = \text{word}(\mathbb{Y})^i$ in polynomial time. We may compute in polynomial time the largest integer $j \geq 0$ such that

$$\text{word}(\mathbb{U}) = \text{word}(\mathbb{U}) \left[: - \left| \text{word}(\mathbb{Y})^j \right| \right] \cdot \text{word}(\mathbb{Y})^j$$

by Theorem 2.9 in [20].

Now, if we can compute:

$$L(\mathcal{A}) \cap \text{word}(\mathbb{U}) \left[: - \left| \text{word}(\mathbb{Y})^j \right| \right] \cdot (\text{word}(\mathbb{Y}))^*$$

in polynomial time, then we can compute $L(\mathcal{A}) \cap \text{word}(\mathbb{U}) \cdot (\text{word}(\mathbb{V}))^*$ in polynomial time by Proposition 3.1. Hence, from now on we may assume that $j = 0$ and $i = 1$.

We now add auxiliary symbols to our alphabet, u and v , and will modify our automaton \mathcal{A} in five steps:

- Step 1** For each transition $(p, X, q) \in \delta$, we check using Proposition 2.2 if X has maximal right \mathbb{V} -factorisation of length greater than or equal to $|\text{word}(\mathbb{V})|$. If so, then let $X = X[:l] \cdot \mathbb{V}[-i:] \cdot \mathbb{V}^k \cdot \mathbb{V}[:j]$ be the maximal right \mathbb{V} -factorisation found. Note that this is unique as $\text{word}(\mathbb{V})$ is primitive. If $k \neq 0$, we now add two new states c and d and four new transitions: $(p, X[:l+i], c)$, (c, v^k, d) , $(c, X[l+i : |\text{word}(X)| - j], d)$ and $(d, X[-j:], q)$. If $k = 0$, we only add one new state c and two new transitions: $(p, X[:l+i], c)$ and $(c, X[-j:], q)$. Now we remove the transition (p, X, q) and denote by \mathcal{A}' the resulting CDFA. Note that $|\delta'| \leq 4 \cdot |\delta|$.
- Step 2** For every pair of states $(p, q) \in Q' \times Q'$, decide if \mathbb{V}^m determines a path from p to q for some $m \leq 2 \cdot (|\delta'| + 1)$ using Theorem 2.5. If so, then let m be the smallest such integer and add a transition (p, v^m, q) , if it doesn't exist already.
- Step 3** For every state $q \in Q'$, decide if $\mathbb{U} \cdot \mathbb{V}^m$ determines a path from q'_0 to q for some $m \leq 2 \cdot (|\delta'| + 1)$ using Theorem 2.5. If so, then let m be the smallest such integer and add a transition $(q'_0, u \cdot v^m, q)$.

Denote by \mathcal{A}'' the resulting CDFA and by $\delta_v \subseteq \delta''$ the transitions with label in $\{u, v\}^*$. The above steps may be done in polynomial time and the subautomaton on the transitions $\delta'' - \delta_v$ accepts precisely the same language as the automaton \mathcal{A} that we started off with.

Claim 1. \mathbb{V}^m determines a path between states p and q if and only if v^m does.

One direction is by construction so we show the other direction by induction on m . The base case when $m = 0$ is trivial. So suppose the claim holds for all $k < m$. Suppose that \mathbb{V}^m determines a path between states p and q traversing a transition $(c, X, d) \in \delta'' - \delta_v$ satisfying $|\text{word}(X)| \geq |\text{word}(\mathbb{V})|$. Then X has a \mathbb{V} -factorisation of length at least $|\text{word}(\mathbb{V})|$ and so by Step 1, $X = \mathbb{V}^k$ for some $k \geq 1$. By uniqueness of \mathbb{V} -factorisations, it follows that there is some $j < m$ such that \mathbb{V}^j determines a path between p and c . Hence \mathbb{V}^{m-j-k} also determines a path between d and q . By induction the claim follows. So now assume that \mathbb{V}^m traverses only transitions (c, X, d) satisfying $|\text{word}(X)| < |\text{word}(\mathbb{V})|$. By Lemma 3.2, there is some integer $k \leq m, 2 \cdot (|\delta'' - \delta_v| + 1)$ and a state c such that \mathbb{V}^k determines a path from p to c . By Step 3, there is a transition (p, v^k, c) . If $k = m$, then $c = q$ and we are done. If $k < m$, then we proceed by induction and the claim is proven.

Claim 2. $\mathbb{U} \cdot \mathbb{V}^m$ determines a path between states q_0 and q if and only if $u \cdot v^m$ does.

We also prove this claim by induction on m . Suppose that $\mathbb{U} \cdot \mathbb{V}^m$ determines a path between q_0 and q . If \mathbb{U} determines a path between q_0 and some state, then we are done.

So suppose \mathbb{U} determines a path between q_0 and $\alpha(i)$ for some $\alpha \in \delta'' - \delta_v$ and some $i \geq 1$. As before, we first assume that \mathbb{V}^m determines a path from $\alpha(i)$ to q traversing a transition $(c, X, d) \in \delta'' - \delta_v$ satisfying $|\text{word}(X)| \geq |\text{word}(\mathbb{V})|$. Note that \mathbb{V}^m must traverse a transition by Step 1. Then just as in the proof of Claim 1, we may use induction to see that $u \cdot v^m$ must determine a path from q_0 to q . Now suppose that \mathbb{V}^m traverses only transitions (c, X, d) satisfying $|\text{word}(X)| < |\text{word}(\mathbb{V})|$. Then by Lemma 3.2 and the fact that $|\alpha[i:]| < |\text{word}(\mathbb{V})|$, there is some integer $k \leq m, 2 \cdot (|\delta'' - \delta_v| + 1)$ and a state c such that $\mathbb{U} \cdot \mathbb{V}^k$ determines a path from p to c . By Step 3, there is a transition $(q_0, u \cdot v^k, c)$. If $k = m$, then $c = q$ and we are done. If $k < m$, then the claim follows from Claim 1.

Since \mathcal{A} was assumed deterministic, for every pair of transitions $(p, v^i, q), (p, v^j, r)$ with $i \leq j$, if we remove (p, v^j, r) , the accepted language remains unmodified. Similarly for transitions with label $u \cdot v^i$. Hence, if we keep doing this until there is at most one outgoing transition with label in $(v)^*$ or $u \cdot (v)^*$ for each state $p \in Q$, we will be left with a deterministic automaton. Finally, the subautomaton on the transitions δ_v gives us the required language. By Proposition 3.1 we may compute the required integers. \square

4. Maximal prefix membership

In [8], the fully compressed membership problem was shown to be in P for CDFA and in NP for CNFA. A natural extension of this problem is the *fully compressed prefix membership problem*: given as input a CNFA \mathcal{A} and an SLP \mathbb{X} , decide if any prefix of $\text{word}(\mathbb{X})$ is in the language $L(\mathcal{A})$. If \mathcal{A} is not a CDFA, then the problem is clearly still in NP. In order to extend Theorem 2.5 to deterministically solve this problem for CDFA, it turns out that the missing ingredient is Theorem 3.3.

The following proposition will serve as a base for a binary search approach to the fully compressed prefix membership problem.

Proposition 4.1. *There is a polynomial-time algorithm that, given as input a CDFA $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, \mathcal{P}, q_0, F)$, a transition $\alpha = (p, X, q) \in \delta$ and a production $W \rightarrow U \cdot V$, decides if there is some integer $i \leq |\text{word}(X)|$ such that $U \cdot V[:i]$ determines a path from q_0 to q , traversing α last. If such an integer exists, then the algorithm also computes such an i .*

Proof. If such an i exists, then there is a crossing appearance of X in $W \rightarrow U \cdot V$. By Lemma 2.4 we may decide in polynomial time if $W \rightarrow U \cdot V$ has a crossing appearance of X . If it does, then the algorithm also produces a triple (j, k, l) such that $\text{Arith}(j, k, l)$ encodes all the crossing appearances of X in W . Now if such an i exists, then there must be some $m \in \text{Arith}(j, k, l)$ such that $W[:m]$ determines a path from q_0 to p . In particular, we have $W[l:m] \in (W[l:l+j])^*$. So the problem now becomes to find m as then $i = m + |\text{word}(X)| - |\text{word}(U)|$. Note that since \mathcal{A} is a CDFA, if m exists, it is unique. Let \mathcal{A}' be a CDFA identical to \mathcal{A} , except with p as its only final state. If such

an m exists, then we must have $L(\mathcal{A}') \cap W[:l] \cdot (W[l:l+j])^* \neq \emptyset$. By Theorem 3.3, we may decide in polynomial time if this intersection is non-empty. If it is non-empty then we may also compute non-negative integers n, n_1 such that:

$$L(\mathcal{A}') \cap W[:l] \cdot (W[l:l+j])^* = W[:l] \cdot W[l:l+j]^{n_1} \cdot (W[l:l+j]^n)^*.$$

The above language is of a simpler form to that given in Theorem 3.3 as now we only have a single final state. In particular, we can have $n = 0$. Note that $X = W[l:l+j]^r \cdot W[l:l+j']$ for some $r \geq k$ and $j' < j$. Now $W[l:l+j]^n$ determines a path from p to p . If $n \neq 0$, then this path traverses α first and so we must have $r \leq n$. Since either $n = 0$ or $r \leq n$, then the required m exists if and only if $n_1 \leq k$. If this condition is satisfied, then:

$$m = l + n_1 \cdot j$$

and so:

$$i = l + n_1 \cdot j + |\text{word}(X)| - |\text{word}(U)|$$

is the required integer. \square

The following Theorem puts the fully compressed prefix membership problem for CDFA in P. Given a positive answer to an instance of the fully compressed prefix membership problem, it also provides a maximal prefix. Finding the maximal prefix is key to our involutive CNFA to CDFA conversion algorithm in Section 5.

Theorem 4.2. *There is a polynomial-time algorithm that, given as input a CDFA $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, \mathcal{P}, q_0, F)$ and an SLP \mathbb{W} , decides if there exists an integer $i \geq 0$ such that $\mathbb{W}[:i] \in L(\mathcal{A})$. If such an i exists, the algorithm also computes the largest such i .*

Proof. Let \mathcal{X} be the non-terminals of \mathcal{A} . If $|\text{word}(\mathbb{W})| < \min\{|\text{word}(X)| \mid X \in \mathcal{X}\}$ then $\mathbb{W}[:i]$ cannot traverse any transition for any $i > 0$. If $q_0 \in F$, then $i = 0$, otherwise there is no such i and we are done.

Now suppose that $|\text{word}(\mathbb{W})| \geq \min\{|\text{word}(X)| \mid X \in \mathcal{X}\}$. The proof is by induction on height. When $\|\mathbb{W}\| = 0$ the result is clear. For the inductive hypothesis, suppose the theorem holds for SLPs of height strictly less than $\|\mathbb{W}\|$. For each transition $\alpha = (p, X, q) \in \delta$, using Proposition 4.1 we may determine whether there is some integer $j \leq |\text{word}(X)|$ such that $U \cdot V[:j]$ determines a path from q_0 to q and traversing α last, where $W \rightarrow U \cdot V$ is the root production of \mathbb{W} .

If there is no such transition, then, if i exists, we must have $i \leq |\text{word}(U)|$. Since $\|U\| < \|\mathbb{W}\|$, by the inductive hypothesis we may decide if there is some i such that $U[:i] \in L(\mathcal{A})$ in polynomial time. Furthermore, if it exists, we may also compute the

maximal such i . Thus we have decided if there is some i such that $\mathbb{W}[: i] \in L(\mathcal{A})$ and computed the maximal such i in polynomial time.

If there is such a transition $\alpha = (p, X, q)$, then $i > |\text{word}(U)|$ and U determines a path from q_0 to $\alpha(-j)$. Now we modify our automaton \mathcal{A} in the following way: remove the transition α , add a new state c and two new transitions $(p, X[: -j], c)$ and $(c, X[-j :], q)$. After changing the initial state to c , let \mathcal{A}' be the resulting CDFA. We have that U determines a path from q_0 to c and so

$$L(\mathcal{A}) \cap \text{word}(U) \cdot \Sigma^* = \text{word}(U) \cdot L(\mathcal{A}').$$

Since $\|V\| < \|W\|$, by the inductive hypothesis we may determine the maximal index l such that $V[: l] \in L(\mathcal{A}')$ in polynomial time. By construction, $V[: l] \in L(\mathcal{A}')$ if and only if $\mathbb{W}[: |\text{word}(U)| + l] \in L(\mathcal{A})$. \square

Corollary 4.3. *The fully compressed prefix membership problem for CDFA (respectively CNFA) is in P (respectively NP).*

5. Converting an involutive CNFA to an involutive CDFA

Any NFA \mathcal{A} can be converted to a DFA \mathcal{A}' such that $L(\mathcal{A}) = L(\mathcal{A}')$. However, even if \mathcal{A}' is minimal, it may have exponentially many more states and transitions than \mathcal{A} . The same holds true for CNFAs. Fortunately, the involutive NFA and CNFA case is different as we explain below. In this section we present an algorithm to solve the following problem:

Input An involutive CNFA \mathcal{A} .

Output An involutive CDFA \mathcal{A}' such that $\pi(L(\mathcal{A})) = \pi(L(\mathcal{A}'))$.

We call this *involutive CNFA to CDFA conversion*. In the classic Stallings algorithm [21], we are given an involutive NFA as input and we identify pairs of adjacent transitions with the same label until we obtain an involutive DFA which accepts the same language. Each such identification is called a *fold*. Since each fold decreases the number of transitions by one, the number of folds to be performed is bounded above by the number of transitions we started off with.

Since our input will have compressed transition labels, the number of folds to be performed may be exponential in the input size. In order to overcome this problem, we would like to perform many folds at once. We may do this by what we call *transition folding*.

5.1. Transition folding

Let $\mathcal{A} = (Q, \Sigma \sqcup \Sigma^{-1}, \mathcal{X}, \delta, \mathcal{P}, q_0, F)$ be an involutive CNFA. Let $\alpha = (p, X, q) \in \delta$ be a transition. We say a CNFA \mathcal{A}' is obtained from \mathcal{A} by a *transition fold*, or *folding* α , if

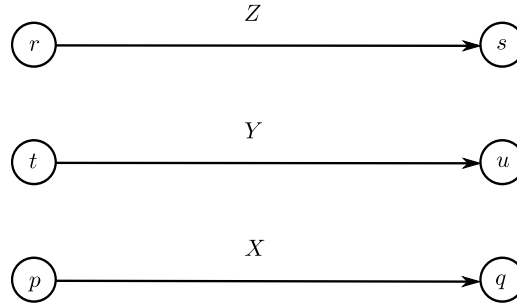


Fig. 1. Transitions β, γ and α .

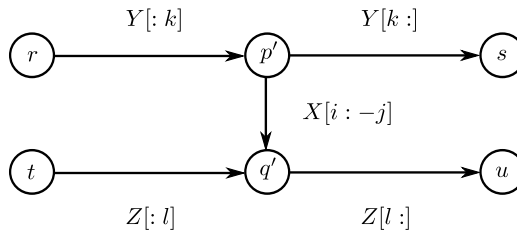


Fig. 2. Result of folding α as in Case 1.

\mathcal{A}' is obtained in the following way. Let i and j be largest integers such that $X[:i]$ and $X^{-1}[:j]$ determine paths in \mathcal{A} from p and q respectively, not traversing any part of α or α^{-1} , and subject to the constraint that $i + j \leq |\text{word}(X)|$. Suppose that $X[:i]$ and $X^{-1}[:j]$ determine paths to $\beta(k)$ and $\gamma(l)$ respectively in \mathcal{A} , with $\beta, \gamma \neq \alpha, \alpha^{-1}$. Let $\beta = (r, Y, s)$ and $\gamma = (t, Z, u)$. We have two cases to consider:

Case 1: $\beta \neq \gamma, \gamma^{-1}$.

Then we remove the transitions α, β, γ and their inverses and we add states p' and q' along with the following transitions:

$$\begin{aligned} (r, Y[:k], p'), (p', Y[k:], s), \\ (t, Z[:l], q'), (q', Z[l:], u), \\ (p', X[i:-j], q'), \end{aligned}$$

and their inverses. See Figs. 1 and 2. If any new transition has empty label, then we remove it, along with its inverse, and identify the corresponding states.

Case 2: $\beta = \gamma$ or $\beta = \gamma^{-1}$.

If $\beta = \gamma^{-1}$, then we may replace γ with γ^{-1} and replace l with $|\text{word}(Z)| - l$ so that $\beta = \gamma$. If $k > l$, then we may swap α with α^{-1} , β with γ and k with l so that $k \leq l$. Now we have two subcases to consider:

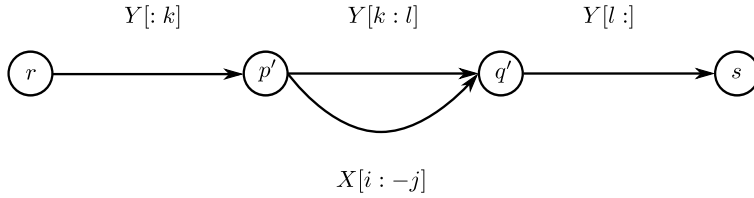


Fig. 3. Result of folding α as in Subcase 2.1.

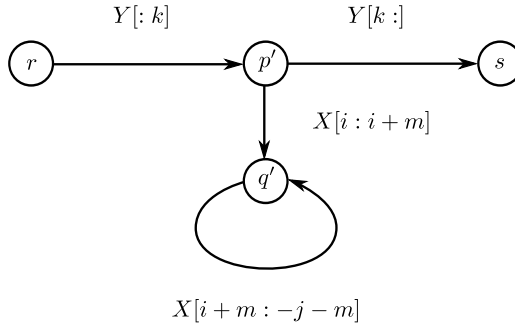


Fig. 4. Result of folding α as in Subcase 2.2.

Subcase 2.1: $k < l$.

Then we remove the transitions α, β and their inverses and add states p', q' and transitions:

$$(r, Y[: k], p'), (p', Y[k : l], q'), (q', Y[l :], s), \\ (p', X[i : -j], q'),$$

and their inverses. See Fig. 3. As before, if any new transition has empty label, then we remove it, along with its inverse, and identify the corresponding states.

Subcase 2.2: $k = l$.

Let m be the largest integer such that $\text{word}(X)[i : i + m] = \text{word}(X)^{-1}[j : j + m]$. Then we remove the transitions α, β and their inverses and add states p', q' and transitions:

$$(r, Y[: k], p'), (p', Y[k :], s), \\ (p', X[i : i + m], q'), (q', X[i + m : -j - m], q'),$$

and their inverses. See Fig. 4. Once again, if any new transition has empty label, then we remove it, along with its inverse, and identify the corresponding states.

No states are removed during a transition fold and so the initial and final states remain unchanged.

We will write $\mathcal{A} \rightarrow \mathcal{A}'$ to denote that \mathcal{A}' is obtained from \mathcal{A} by a transition fold and $\mathcal{A} \xrightarrow{\alpha} \mathcal{A}'$ to denote that \mathcal{A}' is obtained from \mathcal{A} by folding the transition α . We will call a transition fold $\mathcal{A} \rightarrow \mathcal{A}'$ *complete* if $i + j = |\text{word}(X)|$, *incomplete* otherwise. Note that if \mathcal{A} without the transitions α and α^{-1} is not deterministic, then there may be more than one CNFA that is obtained from \mathcal{A} by folding the transition α .

The following two lemmas are by definition of a transition fold.

Lemma 5.1. *If $\mathcal{A} \rightarrow \mathcal{A}'$, then:*

1. $\pi(L(\mathcal{A}')) = \pi(L(\mathcal{A}))$,
2. $|\delta'|/2 - |Q'| \leq |\delta|/2 - |Q|$ and $\mathcal{X}' = \mathcal{X}$,
3. *if each transition label of \mathcal{A} is freely reduced, then so is each transition label of \mathcal{A}' .*

Lemma 5.2. *Suppose that the subautomaton of \mathcal{A} without the transitions α and α^{-1} is a CDFA. Then if $\mathcal{A} \xrightarrow{\alpha} \mathcal{A}'$ is an incomplete transition fold, then \mathcal{A}' is a CDFA.*

The following theorem follows directly from Theorem 4.2 and the definition of a transition fold.

Theorem 5.3. *There is a polynomial-time algorithm that, given as input an involutive CNFA $\mathcal{A} = (Q, \Sigma \sqcup \Sigma^{-1}, \mathcal{X}, \delta, \mathcal{P}, q_0, F)$ and a transition $\alpha \in \delta$ such that the subautomaton of \mathcal{A} without the transitions α and α^{-1} is a CDFA, computes an involutive CNFA \mathcal{A}' such that $\mathcal{A} \xrightarrow{\alpha} \mathcal{A}'$.*

5.2. Minimalistic involutive CNFAs

Throughout our algorithm, we will require that our CNFAs be of a particular form. An involutive CNFA $\mathcal{A} = (Q, \Sigma \sqcup \Sigma^{-1}, \mathcal{X}, \delta, \mathcal{P}, q_0, F)$ is *minimalistic* if the following holds:

1. For every transition $(p, X, q) \in \delta$, we have that $\text{word}(X)$ is freely reduced and non-empty.
2. For every state $p \in Q$ such that $p \neq q_0$, $p \notin F$, we have that p has at least three incoming transitions and at least three outgoing transitions.

We remark that the definition of a minimalistic automaton allows for isolated states, so long as they are either the initial state or a final state. It will be useful for the running of Algorithm 1 to not get rid of such states.

Every CNFA can be transformed into an equivalent minimalistic one. The following lemma says that we can do this efficiently and without adding any complexity.

Lemma 5.4. *There is a polynomial-time algorithm that, given as input an involutive CNFA $\mathcal{A} = (Q, \Sigma \sqcup \Sigma^{-1}, \mathcal{X}, \delta, \mathcal{P}, q_0, F)$, computes an involutive CNFA $\mathcal{A}' = (Q', \Sigma \sqcup \Sigma^{-1}, \mathcal{X}', \delta', \mathcal{P}', q'_0, F')$ such that:*

1. \mathcal{A}' is minimalistic,
2. $\pi(L(\mathcal{A}')) = \pi(L(\mathcal{A}))$,
3. $|\delta'| \leq |\delta|$ and $|\delta'| + |\mathcal{X}'| \leq |\delta| + |\mathcal{X}|$,
4. $\max\{|W'_1|, |W'_2|, \dots\} \leq \max\{2, |W_1|, |W_2|, \dots\}$,
5. if $\partial\mathcal{A} \subseteq \{q_0\} \cup F$, then $|\delta'|/2 - |Q'| \leq |\delta|/2 - |Q|$.

Proof. Let \mathcal{A} be the input involutive CNFA. We first freely reduce each transition label. This may be done in polynomial time and does not change the number of non-terminals by Theorem 3.3 in [20]. Then, for each state $q \in Q \setminus (\{q_0\} \cup F)$ such that q has no incoming or outgoing transition, remove q . For each state $q \in Q \setminus (\{q_0\} \cup F)$ such that q has only one incoming transition (p, X, q) and one outgoing transition (q, X^{-1}, p) , remove q and the two transitions. For each state $q \in Q \setminus (\{q_0\} \cup F)$ such that q has two incoming transitions (q, X, q) , (q, X^{-1}, q) and two outgoing transitions (q, X, q) , (q, X^{-1}, q) , remove these transitions and the state q . Then, for each state $q \in Q \setminus (\{q_0\} \cup F)$ such that q has two incoming transitions (p_1, X, q) , (p_2, Y, q) (that are not inverses of each other) and two outgoing transitions (q, X^{-1}, p_1) , (q, Y^{-1}, p_2) , remove these transitions and the state q and add two non-terminals $Z \rightarrow X[-i] \cdot Y^{-1}[i:]$ and $Z^{-1} \rightarrow Y[-i] \cdot X^{-1}[i:]$, where i is the largest integer so that $X[-i:] = Y[-i:]$, and two new transitions (p_1, Z, p_2) and (p_2, Z^{-1}, p_1) . This may be performed in polynomial time by Theorem 2.9 in [20]. Finally, for each transition (p, X, q) with $\text{word}(X) = \epsilon$, remove (p, X, q) and (q, X^{-1}, p) and identify p with q .

The language is clearly preserved under these operations and the output \mathcal{A}' will be minimalistic by construction. This establishes (1) and (2). The number of transitions never increases and whenever the number of non-terminals increases by one, the number of transitions decreases by one. This establishes (3). Each new production introduced has size two and hence (4) holds. If $\partial\mathcal{A} \subseteq \{q_0\} \cup F$, then whenever the number of states decreases by one, the number of transitions decreases by two. This establishes (5). \square

The following lemma will be used for our inductive hypothesis when bounding the number of transition folds to be performed in Algorithm 1.

Lemma 5.5. *Let $\mathcal{A} = (Q, \Sigma \sqcup \Sigma^{-1}, \mathcal{X}, \delta, \mathcal{P}, q_0, F)$ be a minimalistic involutive CNFA. Then:*

$$\begin{aligned} \frac{|\delta|}{2} &\leq 3 \cdot \left(\frac{|\delta|}{2} - |Q| \right) + |\{q_0\} \cup F| + |\partial\mathcal{A}| \leq \frac{3}{2} \cdot |\delta| - |Q|, \\ |Q| &\leq 2 \cdot \left(\frac{|\delta|}{2} - |Q| \right) + |\{q_0\} \cup F| + |\partial\mathcal{A}| \leq |\delta|. \end{aligned}$$

Proof. Since \mathcal{A} is minimalistic, we have $\partial\mathcal{A} \subseteq \{q_0\} \cup F$. Denote by $M = (\{q_0\} \cup F) - \partial\mathcal{A}$. If $\deg(q)$ denotes the number of incoming and outgoing transitions from the state q , then we have $\deg(q) \geq 6$ for all $q \in Q - (M \sqcup \partial\mathcal{A})$. Thus:

$$\begin{aligned} \frac{|\delta|}{2} &= \sum_{q \in Q} \frac{\deg(q)}{4} \\ &\geq \frac{3}{2} \cdot (|Q| - |M| - |\partial\mathcal{A}|) + |M| + \frac{1}{2} \cdot |\partial\mathcal{A}| \\ &= \frac{3}{2} \cdot |Q| - \frac{1}{2} \cdot |M| - |\partial\mathcal{A}|. \end{aligned}$$

Since $|M| + |\partial\mathcal{A}| = |\{q_0\} \cup F|$, by subtracting $\frac{3}{4} \cdot |\delta|$ from both sides and multiplying by -2 , we obtain the first inequalities. Similarly, by subtracting $|Q|$ from both sides and multiplying by 2 we obtain the second inequalities. \square

5.3. The algorithm

We are now ready to present our involutive CNFA to involutive CDFA conversion algorithm. See Algorithm 1 for the outline. Denote by

$$k(\mathcal{A}) = 5 \cdot \left(\frac{|\delta|}{2} - |Q| \right) + 2 \cdot |\{q_0\} \cup F|$$

and

$$s(\mathcal{A}) = \sum_{(p, X, q) \in \delta} \frac{|\text{word}(X)|}{2}.$$

The following proposition shows that the algorithm solves the involutive CNFA to CDFA conversion problem and also bounds the number of transition folds performed in terms of $k(\mathcal{A})$ and $s(\mathcal{A})$. The main idea is that, by our choice of the order in which the transition folds are performed, each complete transition fold gets rid of a definite proportion of the CNFA. Note that the size of the input to Algorithm 1 is bounded below by $\log_2(s(\mathcal{A}))$.

Algorithm 1 Involutive CNFA to CDFA conversion.

1. Make \mathcal{A} minimalistic. Let $i = 0$ and $\mathcal{A} = \mathcal{A}_0$.
2. Let $\alpha_i = (p, X, q) \in \delta_i$ such that $|\text{word}(X)|$ is maximal over all transition labels. If

$$\mathcal{A}'_i = (Q, \Sigma \sqcup \Sigma^{-1}, \mathcal{X}, \delta - \{\alpha_i, \alpha_i^{-1}\}, \mathcal{P}, q_0, F \cup \{p, q\})$$

is not a CDFA, proceed to 3. If \mathcal{A}'_i is a CDFA, set $\mathcal{A}''_i = \mathcal{A}'_i$ and proceed to 4.

3. Convert \mathcal{A}'_i into a CDFA and let \mathcal{A}''_i be the output.
 4. Let \mathcal{A}'''_i be obtained from \mathcal{A}''_i by adding α_i and α_i^{-1} to its transition set and removing p and q from its final states if they were not final states in \mathcal{A}_i .
 5. Fold α_i in \mathcal{A}'''_i and make the resulting CNFA minimalistic. Let \mathcal{A}_{i+1} be the output. If \mathcal{A}_{i+1} is a CDFA then terminate with output \mathcal{A}_{i+1} . If not, then let $i := i + 1$ and go back to 2.
-

Proposition 5.6. *Let \mathcal{A} be an involutive CNFA with $\partial\mathcal{A} \subseteq \{q_0\} \cup F$ and let $k = k(\mathcal{A})$ and $s = s(\mathcal{A}) \geq 2$. Then Algorithm 1 performs Step 5 at most $(k \cdot \log(s))^k$ times and outputs a CDFA \mathcal{A}_m such that $k(\mathcal{A}_m) \leq k(\mathcal{A})$ and $\pi(L(\mathcal{A}_m)) = \pi(L(\mathcal{A}))$.*

Proof. Since $\partial\mathcal{A} \subseteq \{q_0\} \cup F$, we first note that making \mathcal{A} minimalistic does not increase the quantities $k(\mathcal{A})$ or $s(\mathcal{A})$ by Lemma 5.4. We also note that for all $i \geq 2$:

$$\frac{1}{\log\left(\frac{i}{i-1}\right)} \leq i$$

so that it suffices to show that Algorithm 1 performs Step 5 at most $\left(\log_{\frac{k}{k-1}}(s)\right)^k$ many times.

The proof is by induction on n . In the base case, \mathcal{A} has only one pair of inverse transitions. Then Algorithm 1 performs Step 5 at most one time. So now suppose that \mathcal{A} has at least two pairs of inverse transitions and suppose that the result is true for all minimalistic involutive CNFAs \mathcal{A}' with $s(\mathcal{A}') < s(\mathcal{A})$.

Denote by F' and F the final states of \mathcal{A}'_0 and \mathcal{A}_0 respectively. Let $\alpha_0 = (p, X, q)$ be the transition found in Step 2. We have $i = |\{q_0\} \cup F'| - |\{q_0\} \cup F| \in \{0, 1, 2\}$ and $k(\mathcal{A}'_0) = k(\mathcal{A}_0) + 2 \cdot i - 5 \leq k(\mathcal{A}_0) - 1$. Moreover, by our choice of α_0 , we have

$$s(\mathcal{A}'_0) \leq \frac{|\delta| - 2}{|\delta|} \cdot s(\mathcal{A}_0) \leq \frac{k-1}{k} \cdot s$$

by Lemma 5.5. We now assume that \mathcal{A}'_0 is not a CDFA. The other case is simpler and so we omit it. Note that if \mathcal{A}'_0 is not a CDFA, then we have $s(\mathcal{A}'_0) \geq 2$. Hence, the inductive hypothesis applies and so Step 5 is performed at most

$$\left(\log_{\frac{k-1}{k-2}}\left(\frac{k-1}{k} \cdot s\right)\right)^{k-1}$$

many times when Algorithm 1 is run with input \mathcal{A}'_0 . Once Step 5 is performed, either \mathcal{A}_1 is obtained from \mathcal{A}'''_0 by an incomplete transition fold, or it is obtained by a complete transition fold. By induction, we have $k(\mathcal{A}''_0) \leq k(\mathcal{A}'_0)$. Arguing as before, we then have

$$k(\mathcal{A}'''_0) = k(\mathcal{A}''_0) + 5 - 2 \cdot i \leq k(\mathcal{A}'_0) + 5 - 2 \cdot i \leq k(\mathcal{A}_0).$$

Then by Lemmas 5.1 and 5.4, we have that also $k(\mathcal{A}_1) \leq k(\mathcal{A}_0)$.

Now if \mathcal{A}_1 is a CDFA, the algorithm terminates having performed Step 5 at most:

$$\begin{aligned} \left(\log_{\frac{k-1}{k-2}}\left(\frac{k-1}{k} \cdot s\right)\right)^{k-1} + 1 &\leq \left(\log_{\frac{k}{k-1}}(s) - 1\right)^{k-1} + 1 \\ &\leq \left(\log_{\frac{k}{k-1}}(s)\right)^k \end{aligned}$$

many times. If \mathcal{A}_1 is not a CDFA, then it is obtained from \mathcal{A}_0''' by a complete transition fold by Lemma 5.2. We have that $2 \leq s(\mathcal{A}_1) < s(\mathcal{A}_0)$ and

$$s(\mathcal{A}_1) \leq \frac{k-1}{k} \cdot s.$$

Once again, the inductive hypothesis applies and Algorithm 1 performs Step 5 at most

$$\left(\log_{\frac{k}{k-1}} \left(\frac{k-1}{k} \cdot s \right) \right)^k$$

many times on input \mathcal{A}_1 . In particular, $k(\mathcal{A}_m) \leq k(\mathcal{A}_1) \leq k(\mathcal{A})$. Putting everything together, Algorithm 1 performs Step 5 at most:

$$\begin{aligned} & \left(\log_{\frac{k}{k-1}} \left(\frac{k-1}{k} \cdot s \right) \right)^k + \left(\log_{\frac{k-1}{k-2}} \left(\frac{k-1}{k} \cdot s \right) \right)^{k-1} + 1 \\ & \leq \left(\log_{\frac{k}{k-1}}(s) - 1 \right)^{k-1} \cdot \left(\left(\log_{\frac{k}{k-1}}(s) - 1 \right) + 1 \right) + 1 \\ & \leq \left(\log_{\frac{k}{k-1}}(s) \right)^k - \left(\log_{\frac{k}{k-1}}(s) \right) + 1 \\ & \leq \left(\log_{\frac{k}{k-1}}(s) \right)^k \end{aligned}$$

many times on input \mathcal{A} as required.

The fact that \mathcal{A}_m is a CDFA follows from the algorithm. The fact that $\pi(L(\mathcal{A}_m)) = \pi(L(\mathcal{A}))$ follows from Lemmas 5.1 and 5.4. Thus, the algorithm is correct. \square

Finally, putting everything together, we may provide complexity bounds for the compressed involutive CNFA to CDFA conversion problem.

Theorem 5.7. *There is an algorithm that, given as input an involutive CNFA $\mathcal{A} = (Q, \Sigma \sqcup \Sigma^{-1}, \mathcal{X}, \delta, \mathcal{P}, q_0, F)$, computes an involutive CDFA \mathcal{A}' such that $\pi(L(\mathcal{A}')) = \pi(L(\mathcal{A}))$ in $O(n^{O(|\delta|)})$ time.*

Proof. We are going to analyse the complexity of Algorithm 1. By combining Lemmas 5.1, 5.4 and 5.5, we see that the number of transitions, and hence also the number of states, of any involutive CNFA appearing in the running of Algorithm 1 is at most $\frac{3}{2} \cdot |\delta|$. Moreover, the only new non-terminals introduced in the running of Algorithm 1 come from Lemma 5.4, and each such non-terminal has production size two. Hence, the size of any involutive CNFA appearing in the running of Algorithm 1 is at most polynomial in the size of \mathcal{A} by Lemmas 5.1 and 5.4.

By Lemma 5.5 and Proposition 5.6, Algorithm 1 performs Step 5 at most $O(n^{O(|\delta|)})$ many times. Therefore, each step is performed at most $O(n^{O(|\delta|)})$ many times. By Lemma 5.4, Step 1 requires polynomial time. Step 2 requires at most polynomial time

also. Step 3 simply makes a call to Algorithm 1 and so its runtime is already accounted for by the runtime of the other steps. Step 4 clearly requires at most polynomial time. By Lemma 5.2, Theorem 5.3 and Lemma 5.4, Step 5 requires time polynomial in input size. So overall, Algorithm 1 is in $O(n^{O(|\delta|)})$. \square

Applying the main results from each section to the context of free groups, we prove our main theorem.

Theorem 5.8. *Given as input a collection $\mathbb{W}, \mathbb{W}_1, \dots, \mathbb{W}_k$ of SLPs over $\Sigma \sqcup \Sigma^{-1}$, we may do the following in $O(n^{O(k)})$ time:*

1. *Compute a compressed Stallings automaton for*

$$H = \langle \pi(\text{word}(\mathbb{W}_1)), \dots, \pi(\text{word}(\mathbb{W}_k)) \rangle \leq F(\Sigma).$$

2. *Compute an SLP \mathbb{X} such that*

$$\pi(\text{word}(\mathbb{W})) \in H \cdot \pi(\text{word}(\mathbb{X}))$$

and $|\text{word}(\mathbb{X})|$ is smallest possible.

3. *Compute an SLP \mathbb{X} such that*

$$\langle \pi(\text{word}(\mathbb{W})) \rangle \cap H = \langle \pi(\text{word}(\mathbb{X})) \rangle.$$

Proof. Let W be the root non-terminal for \mathbb{W} and W_i the root non-terminals for \mathbb{W}_i for each i . Let \mathcal{X} be the union of all the non-terminals appearing in each \mathbb{W}_i and \mathcal{P} the union of all the productions. Consider the CNFA

$$\mathcal{A} = (\{q_0\}, \Sigma \sqcup \Sigma^{-1}, \mathcal{X}, \{(q_0, W_i, q_0), (q_0, W_i^{-1}, q_0)\}_{i=1}^k, \mathcal{P}, q_0, \{q_0\}).$$

We have $\pi(L(\mathcal{A})) = H$. By Theorem 5.7 we may compute a minimalistic involutive CDFA \mathcal{A}' such that $\pi(L(\mathcal{A}')) = \pi(L(\mathcal{A}))$ in $O(n^{O(k)})$ time. At no point in the algorithm can an unreachable state be created. Combined with the fact that \mathcal{A}' is minimalistic, it follows that \mathcal{A}' is a compressed Stallings automaton for the subgroup H .

By Theorem 3.3 in [20], we may compute an SLP \mathbb{W}' such that $\text{word}(\mathbb{W}')$ is freely reduced and $\pi(\text{word}(\mathbb{W}')) = \pi(\text{word}(\mathbb{W}))$. Let i be the maximal integer such that $\mathbb{W}'[i]$ determines a path from the start state of \mathcal{A}' to any other state. Let p be this state. By Theorem 4.2, we may compute p and i in polynomial time. We may also compute the shortest path from the start state to p in polynomial time. Let $X_1 \dots X_m$ be the sequence of labels determined by this path. Then if \mathbb{X} is an SLP with $\text{word}(\mathbb{X}) = \text{word}(X_1) \dots \text{word}(X_m) \cdot \text{word}(\mathbb{W}')[i:]$, then $\pi(\text{word}(\mathbb{W})) \in H \cdot \pi(\text{word}(X_1 \dots X_m \cdot \mathbb{W}'[i:]))$ and $|\text{word}(\mathbb{X})|$ is minimal possible.

By Corollary 3.6 in [20], we may compute in polynomial time SLPs \mathbb{U} and \mathbb{V} such that $\text{word}(\mathbb{U}) \cdot \text{word}(\mathbb{V}) \cdot \text{word}(\mathbb{U}^{-1}) = \text{word}(\mathbb{W}')$ with $|\text{word}(\mathbb{U})|$ maximal possible. By Theorem 4.2, we may decide if there is a transition $\alpha \in \delta'$ and an integer i such that $\text{word}(\mathbb{U})$ determines a path from the start state of \mathcal{A}' to $\alpha(i)$. Furthermore, if it does, we may compute $\alpha = (p, X, q)$ and i in polynomial time. Then we modify \mathcal{A}' as follows: add a new state p' and new transitions $(p, X[:i], p')$, $(p', X^{-1}[-i:], p)$, $(p', X[i:], q)$ and $(q, X^{-1}[:i], p')$. Remove the transitions α and α^{-1} and make p' the initial and final state. Let \mathcal{A}'' be the resulting CDFA. Now $\langle \pi(\text{word}(\mathbb{W})) \rangle \cap H = \langle \pi(\text{word}(\mathbb{W}))^m \rangle$ if and only if $(\text{word}(\mathbb{V}))^* \cap L(\mathcal{A}'') = (\text{word}(\mathbb{V}))^m$. Finally, we may compute m in polynomial time by Theorem 3.3. \square

Corollary 5.9. *The fully compressed membership problem for k -generated subgroups of a free group is in P .*

We conclude with a question.

Question 5.10. Is the runtime of Algorithm 1 actually in P ?

Data availability

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

Acknowledgments

We would like to thank Saul Schleimer for suggesting the problem and for the many helpful mathematical discussions. We would also like to thank the anonymous reviewer for the valuable comments. This work was carried out during the author's PhD studies at the University of Warwick.

References

- [1] J. Avenhaus, K. Madlener, The Nielsen reduction and P-complete problems in free groups, *Theor. Comput. Sci.* 32 (1–2) (1984) 61–76.
- [2] J. Avenhaus, D. Wißmann, Using rewriting techniques to solve the generalized word problem in polycyclic groups, in: *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation, ISSAC '89*, Association for Computing Machinery, New York, NY, USA, 1989, pp. 322–337.
- [3] M. Benoist, Parties rationnelles du groupe libre, *C. R. Acad. Sci. Paris, Sér. A-B* 269 (1969) A1188–A1190.
- [4] L. Gasieniec, M. Karpinski, W. Plandowski, W. Rytter, Efficient algorithms for Lempel-Ziv encoding, in: R. Karlsson, A. Lingas (Eds.), *Algorithm Theory — SWAT'96*, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 392–403.
- [5] Z. Grunschlag, *Algorithms in geometric group theory*, ProQuest LLC, Ann Arbor, MI, 1999, Thesis (Ph.D.)—University of California, Berkeley.
- [6] Y. Gurevich, P. Schupp, Membership problem for the modular group, *SIAM J. Comput.* 37 (2) (2007) 425–459.

- [7] C. Hagenah, Gleichungen mit regulären Randbedingungen über freien Gruppen, Thesis (Ph.D.), University of Stuttgart, Germany, 2000.
- [8] A. Jež, The complexity of compressed membership problems for finite automata, *Theory Comput. Syst.* 55 (4) (2014) 685–718.
- [9] I. Kapovich, A. Myasnikov, Stallings foldings and subgroups of free groups, *J. Algebra* 248 (2) (2002) 608–668.
- [10] Y. Lifshits, Processing compressed texts: a tractability border, in: B. Ma, K. Zhang (Eds.), *Combinatorial Pattern Matching*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 228–240.
- [11] M. Linton, On the intersections of finitely generated subgroups of free groups: reduced rank to full rank, [arXiv:2108.10814](https://arxiv.org/abs/2108.10814), 2021.
- [12] M. Lohrey, Algorithms on SLP-compressed strings: a survey, *Groups Complex. Cryptol.* 4 (2) (2012) 241–299.
- [13] M. Lohrey, Compression techniques in group theory, in: *Connecting with Computability*, in: *Lecture Notes in Comput. Sci.*, vol. 12813, Springer, Cham, 2021, pp. 330–341.
- [14] M. Lohrey, Subgroup membership in $GL(2, \mathbb{Z})$, in: 38th International Symposium on Theoretical Aspects of Computer Science, in: *LIPIcs. Leibniz Int. Proc. Inform.*, vol. 187, Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2021, 51.
- [15] M. Lohrey, B. Steinberg, The submonoid and rational subset membership problems for graph groups, *J. Algebra* 320 (2) (2008) 728–755.
- [16] A. Myasnikov, A. Ushakov, D.W. Won, The word problem in the Baumslag group with a non-elementary Dehn function is polynomial time decidable, *J. Algebra* 345 (2011) 324–342.
- [17] W. Plandowski, W. Rytter, Complexity of language recognition problems for compressed words, in: *Jewels Are Forever*, Springer, Berlin, 1999, pp. 262–272.
- [18] M.O. Rabin, D. Scott, Finite automata and their decision problems, *IBM J. Res. Dev.* 3 (1959) 114–125.
- [19] V. Roman'kov, On the occurrence problem for rational subsets of a group, in: V. Roman'kov (Ed.), *International Conference on Combinatorial and Computational Methods in Mathematics*, 1999, pp. 76–81.
- [20] S. Schleimer, Polynomial-time word problems, *Comment. Math. Helv.* 83 (4) (2008) 741–765.
- [21] J.R. Stallings, Topology of finite graphs, *Invent. Math.* 71 (3) (1983) 551–565.