Journal of Algebra 553 (2020) 268–318

Contents lists available at ScienceDirect

Journal of Algebra

www.elsevier.com/locate/jalgebra

The word problem of the Brin-Thompson group is coNP-complete

J.C. Birget

Dept. of Computer Science, and CCIB, Rutgers U., Camden, United States of America

ARTICLE INFO

Article history:
Received 21 February 2019
Available online 20 February 2020
Communicated by Derek Holt

Keywords:
Computational complexity
coNP-completeness
Word problem
Brin-Thompson group

ABSTRACT

We prove that the word problem of the Brin-Thompson group nV over a finite generating set is coNP-complete for every $n \geq 2$. It is known that $\{nV : n \geq 1\}$ is an infinite family of infinite, finitely presented, simple groups. We also prove that the word problem of the Thompson group V over a certain infinite set of generators, related to boolean circuits, is coNP-complete.

© 2020 Elsevier Inc. All rights reserved.

1. Introduction

The group nV was introduced by Brin [14] as an n-dimensional generalization of Richard Thompson's group V, for any positive integer n (with $1V = V$).

Brin proved that $2V$ is finitely generated and simple, that V is not isomorphic to $2V$ [14], that $2V$ is finitely presented [15], and that all nV are simple [16]. Hennig and Mattucci [24] show that all nV are finitely presented. Bleak and Lanoue [11] show that all nV are non-isomorphic. In short, the groups nV are an infinite family of infinite, finitely presented, simple groups.

The word problem of nV is decidable, as is easy to see from the definition of nV. The main result of the present paper is the following.

E-mail address: birget@camden.rutgers.edu.

https://doi.org/10.1016/j.jalgebra.2020.02.013
0021-8693/© 2020 Elsevier Inc. All rights reserved.

Theorem 1.1. *The word problem of nV over any finite generating set is* coNP*-complete, for all $n \geq 2$.*

Remarks on the theorem:

This is only the second example of a finitely presented group with coNP-complete word problem; the first example appeared in [6]. This is also the first "naturally occurring" example of a finitely presented group with either NP-complete or coNP-complete word problem. The proof of Theorem 1.1 strengthens the connection between acyclic circuits and finite group presentations; such a connection already played a crucial role in [6].

The Theorem implies that if $\mathsf{NP} \neq \mathsf{coNP}$ then the Dehn function of nV (for $n \geq 2$) has no polynomial upper bound; more strongly, nV cannot be embedded into a finitely presented group with polynomially bounded Dehn function (by [38,4]).

The Theorem implies that if $\mathsf{P} \neq \mathsf{NP}$ then $2V$ is not embeddable into V. It was proved recently [32, Coroll. 11.20] that $(n+1)V$ does not embed into nV for any $n \geq 1$.

The groups nV for $n \geq 2$ are the first examples of finitely presented simple groups whose word problem is harder than P (if $\mathsf{P} \neq \mathsf{NP}$).[1] Finitely presented infinite simple groups are related to the Boone-Higman theorem [13]. In [13] the authors ask whether their theorem can be strengthened as follows: *Does a finitely generated group G have a decidable word problem iff G is embeddable into a finitely presented simple group?* In contrast, it was observed in [6, Section 1] that all known finitely presented simple groups have a word problem of very low complexity; even coNP is a low complexity class on the scale of all decidable problems. The enormous gap between what is asked, and what has been observed so far motivates the following.

Question. Are the computational complexities of the word problems of all finitely presented simple groups unbounded?

More precisely, the negation of the question is: Is there a time-constructible total function t such that the word problems of all finitely presented simple groups belong to $\mathsf{DTime}(t)$? (See e.g. [26] for the definitions of "time-constructible" and "$\mathsf{DTime}(t)$".) In case of a negative answer, the Boone-Higman question also has a negative answer. If the answer is positive then there is a chance that the Boone-Higman question has a positive answer; in that case, the proof of the answer to the Question above might be easier than the proof of a strengthened Boone-Higman theorem, and could be a useful step along the way.

Overview: In section 2 we define the Higman-Thompson groups $G_{k,1}$ and the Brin-Thompson groups nV and $nG_{k,1}$ by (partial) actions on finite strings, or n-tuples of strings. For this, the concept of prefix code of strings is generalized to the concept of

[1] The Higman-Thompson groups $G_{k,s}$ have their word problem in P (in fact in coCFL, by Lehnert and Schweitzer [29]). For other currently known finitely presented infinite simple groups (Meier [34,35], Röver [37], Burger and Mozes [17], Lodha [30]), the complexity of the word problem has not been studied, but appears to be in P.

joinless code of n-tuples of strings. For the study of the computational complexity of the word problem, the string-based formalism is more convenient than the geometric approach. It follows fairly directly that the word problem of nV over a finite generating set belongs to coNP (section 3).

The proof of coNP-hardness is given in section 4. It goes through several steps, following the same strategy as the first half of [6] (where it was proved that a certain subgroup of $G_{3,1}$, over a certain infinite generating set, has a coNP-complete word problem. Based on this we show that the Thompson group V, over a certain infinite generating set, has a coNP-complete word problem. This infinite generating set of V consists of a finite generating set, together with all the bit-position transpositions $\tau_{i,i+1}$ (where $\tau_{i,i+1}$: $x_1 \dots x_{i-1}\, x_i\, x_{i+1}\, x_{i+2} \dots \longmapsto x_1 \dots x_{i-1}\, x_{i+1}\, x_i\, x_{i+2} \dots$). An alternative approach, based on bijective circuits and the work of Jordan [27], is described in subsection 4.5. Finally, we show that $\tau_{i,i+1}$ can be expressed by $\tau_{1,2}$ and the shift σ. This reduces the word problem of V, over an infinite generating set that includes position transpositions, to the word problem of $2V$ over a finite generating set (subsection 4.6).

Summary of abbreviations and notations:

– The word *function* in this paper means partial function. The domain of a function $f : X \to Y$ is denoted by $\mathrm{Dom}(f) \subseteq X$, and the image by $\mathrm{Im}(f) \subseteq Y$. Most often, the sets X and Y will be free monoids A^*, or Cantor spaces A^ω, or their direct powers nA^* or nA^ω.
– A^*, the free monoid freely generated by A, a.k.a. the set of all strings over A;
– ε, the empty string;
– A^+, the free semigroup; $A^+ = A^* \setminus \{\varepsilon\}$;
– $|x|$, the length of the string $x \in A^*$;
– $x \leq_{\mathrm{pref}} y$, x $(\in A^*)$ is a prefix of y $(\in A^* \cup A^\omega)$;
– $x \parallel_{\mathrm{pref}} y$, x is prefix-comparable with y;
– nA^*, nA^ω, the n-fold cartesian product $\mathsf{X}_{i=1}^{n} A^*$, respectively $\mathsf{X}_{i=1}^{n} A^\omega$;
– $(\varepsilon)^n$, the n-tuple of empty strings;
– $A_{\varepsilon,n} = \bigcup_{1\leq i\leq n} \{\varepsilon\}^{i-1} \times A \times \{\varepsilon\}^{n-i}$, the unique minimum generating set of the monoid nA^*;
– $\ell(x)$, $\max\{|x_1|, \dots, |x_n|\}$ if $x = (x_1, \dots, x_n) \in nA^*$;
– $x \leq_{\mathrm{init}} y$, x $(\in nA^*)$ is an initial factor of y $(\in nA^* \cup nA^\omega)$;
– DAG, directed acyclic graph;
– $f|_M$, the restriction of a function f to a set M.

2. Definition of nV based on strings

The standard definitions in computational complexity require strings as inputs. Brin's original definition of nV uses geometric actions, but for the proof of coNP-completeness of the word problem of nV we also need a (partial) action of nV on n-tuples of strings. The groups nV are generalizations of V. We first look at V.

2.1. Definition of V based on strings

The group V can be defined in many ways; see e.g. [42,33,43,25,40,19]. We will mostly use two definitions of V from [5] (which are similar to [40], except that we use the terminology of prefix codes, right ideals, and right-ideal morphisms).

We recall some standard notions. An alphabet is any finite set, although we mostly use $\{0,1\}$ (the bits), and $\{0,1,\ldots,k-1\}$ for any integer $k \geq 2$. For an alphabet A and $m \in \mathbb{N}$, A^m denotes the set of sequences of length m over A (called set of *strings* of length m), and for $x \in A^m$ we say that $|x| = m$ (i.e., the *length* of x is m); $A^{\leq m}$ is the set of strings of length $\leq m$. The empty string is denoted by ε, and $|\varepsilon| = 0$. The set of all strings over A is denoted by A^*, and the set of all infinite strings indexed by the ordinal ω is denoted by A^ω. By default a "string" is finite; for infinite strings we explicitly say "infinite". For $x_1, x_2 \in A^*$ the *concatenation* is denoted by x_1x_2 or $x_1 \cdot x_2$; it has length $|x_1|+|x_2|$. For two subsets $S_1, S_2 \subseteq A^*$, we define the concatenation by $S_1 \cdot S_2 = \{x_1 \cdot x_2 : x_1 \in S_1 \text{ and } x_2 \in S_2\}$.

For $x, p \in A^*$ we say that p is a *prefix* of x iff $(\exists u \in A^*)\, x = pu$; this is denoted by $p \leq_{\text{pref}} x$. Two strings $x, y \in A^*$ are called *prefix-comparable* (denoted by $x \parallel_{\text{pref}} y$) iff $x \leq_{\text{pref}} y$ or $y \leq_{\text{pref}} x$. A *prefix code* (a.k.a. a prefix-free set) is any subset $P \subset A^*$ such that for all $p_1, p_2 \in P$: $p_1 \parallel_{\text{pref}} p_2$ implies $p_1 = p_2$. A *right ideal* of A^* is, by definition, any subset $R \subseteq A^*$ such that $R = R \cdot A^*$. A subset $C \subseteq R$ is said to generate R as a right ideal iff $R = C \cdot A^*$. It is easy to prove that every finitely generated right ideal is generated by a unique finite prefix code, and this prefix code is the minimum generating set of the right ideal (with respect to $\subseteq$). By definition, a *maximal prefix code* is a prefix code $P \subset A^*$ that is not a strict subset of any other prefix code of A^*. An *essential right ideal* is, by definition, a right ideal $R \subseteq A^*$ such that all right ideals of A^* intersect R (i.e., have a non-$\varnothing$ intersection with R). It is well known (see e.g. [5, Lemma 8.1]) that a right ideal $R \subseteq A^*$ is essential iff the unique prefix code that generates R is maximal.

A *right ideal morphism* of A^* is, by definition, a function $f : A^* \to A^*$ such that for all $x \in \mathrm{Dom}(f)$ and all $w \in A^*$: $f(xw) = f(x)\, w$. In that case, $\mathrm{Dom}(f)$ is a right ideal; one easily proves that $\mathrm{Im}(f)$ is also a right ideal. The prefix code that generates $\mathrm{Dom}(f)$ is denoted by $\mathrm{domC}(f)$, and is called the *domain code* of f; the prefix code that generates $\mathrm{Im}(f)$ is denoted by $\mathrm{imC}(f)$, and is called the *image code*. We are interested in the following monoid:

$$\mathcal{RI}_A^{\mathsf{fin}} = \{f : f \text{ is a right ideal morphism of } A^* \text{ such that } f \text{ is } \textit{injective}, \text{ and } \mathrm{domC}(f) \text{ and } \mathrm{imC}(f) \text{ are } \textit{finite maximal} \text{ prefix codes}\}.$$

We usually write $\mathcal{RI}^{\mathsf{fin}}$ for $\mathcal{RI}_A^{\mathsf{fin}}$ since we usually just deal with one alphabet A at a time. It is proved in [5, Prop. 2.1] that every $f \in \mathcal{RI}^{\mathsf{fin}}$ is contained in a unique $\subseteq$-maximum right ideal morphism in $\mathcal{RI}^{\mathsf{fin}}$; this is called the *maximum extension* of f. The Higman-Thompson group $G_{k,1}$ (where $k = |A|$) is a homomorphic image of $\mathcal{RI}^{\mathsf{fin}}$:

Definition 2.1 *(Thompson group V and Higman-Thompson groups $G_{k,1}$).* The Thompson group V, as a set, consists of the right ideal morphisms $f \in \mathcal{RI}^{\mathsf{fin}}_{\{0,1\}}$ that are maximum extensions in $\mathcal{RI}^{\mathsf{fin}}_{\{0,1\}}$. The multiplication in V consists of composition, followed by maximum extension.

The same definition for $\mathcal{RI}^{\mathsf{fin}}_A$ with $A = \{0, 1, \ldots, k-1\}$ yields the Higman-Thompson group $G_{k,1}$ for every $k \geq 2$; $V = G_{2,1}$.

Every element $f \in \mathcal{RI}^{\mathsf{fin}}$ (and in particular, every $f \in G_{k,1}$) is determined by the restriction of f to $\mathrm{domC}(f)$. This restriction $f_{\mathrm{domC}(f)} : \mathrm{domC}(f) \to \mathrm{imC}(f)$ is a finite bijection, called the *table* of f [25]. Obviously, f ($\in \mathcal{RI}^{\mathsf{fin}}$) determines $\mathrm{domC}(f)$ and hence a unique table. When we use tables we do not always assume that f is a maximum extension. The well known tree representation of $G_{k,1}$ is obtained by using the *prefix trees* of $\mathrm{domC}(f)$ and $\mathrm{imC}(f)$.

Lemma 2.2. *Let $P, Q \subset A^*$ be finite maximal prefix codes. The right ideal morphism $f \in \mathcal{RI}^{\mathsf{fin}}$ determined by a table $F\colon P \to Q$ can be extended iff there exist $s, t \in A^*$ such that for every $\alpha \in A$: $s\alpha \in P$, $t\alpha \in Q$, and $F(s\alpha) = t\alpha$.*

In that case, f can be extended by defining $f(s) = t$. The table for this extension is obtained be replacing P by $(P \setminus sA) \cup \{s\}$, Q by $(Q \setminus qA) \cup \{q\}$, and $\{(s\alpha, t\alpha) : \alpha \in A\}$ by $\{(s, t)\}$.

This is called an extension step *of the table F.*

Proof. See [5, Lemma 2.2] and [25]. □

Since in an extension step the cardinality of $\mathrm{domC}(f)$ decreases, only finitely steps are needed to reach the maximum extension of f; the number of steps is $< |\mathrm{domC}(f)|$.

Based on the representation of the elements of V (and of $G_{k,1}$) by tables, one can show easily that the word problem of these groups is in P. A much stronger result is that the word problem is in coCFL (the set of languages whose complement is context-free) [29]; coCFL is a strict subclass of the parallel complexity class AC^1, which is a subclass of P (see e.g., [23]).

The A^ω definition of $G_{k,1}$: Maximality of finite prefix codes has the following characterization in terms of A^ω. A finite prefix code $P \subset A^*$ is maximal iff , $PA^\omega = A^\omega$. (This is not true for infinite prefix codes; a counter example is 0^*1.)

It follows that every element $f \in G_{k,1}$ determines a permutation of A^ω. Conversely, let $P \subset A^*$ be a finite maximal prefix code. Then for every $w \in A^\omega$ there exists a unique $p \in P$ and $v \in A^\omega$ such that $w = pv$. Let f be a permutation of A^ω for which there exists a table $F\colon P \to Q$ such f is defined by $f(pv) = F(p)\ v$ (for every $p \in P$ and $v \in A^\omega$). Then $f \in G_{k,1}$.

Thus, $G_{k,1}$ can be defined as a certain group of permutations of A^ω.

Lemma 2.3. *Let $F_1\colon P_1 \to Q_1$ and $F_2\colon P_2 \to Q_2$ be two tables that determine, respectively, the right ideal morphisms $f_1, f_2 \in \mathcal{RI}^{\mathsf{fin}}$. Then the following are equivalent:*

(1) *F_1 and F_2 determine the same element of $G_{k,1}$ (by maximum extension);*
(2) *f_1 and f_2 have the same maximum extension in $\mathcal{RI}^{\mathsf{fin}}$;*
(3) *f_1 and f_2 have a common restriction in $\mathcal{RI}^{\mathsf{fin}}$;*
(4) *f_1 and f_2 have a common restriction to an essential right ideal of A^*;*
(5) *F_1 and F_2 determine the same function on A^ω;*
(6) *f_1 and f_2 determine the same function on A^ω.*

Proof. (1) and (2) are equivalent by the definition of $G_{k,1}$. (2) implies (3) (which implies (4)): The intersection $f_1 \cap f_2$ is a common restriction; by [5, Lemma 8.3], $\mathrm{Dom}(f_1) \cap \mathrm{Dom}(f_2)$ is an essential right ideal. Moreover, $\mathrm{domC}(f_1 \cap f_2) \subset \mathrm{domC}(f_1) \cup \mathrm{domC}(f_2)$; hence $\mathrm{domC}(f_1 \cap f_2)$ is finite. (4) implies (2) by uniqueness of maximum extensions in $\mathcal{RI}^{\mathsf{fin}}$ (see [5, Lemma 2.1], which does not require finiteness of prefix codes). (3) implies (5) in an obvious way. And (5) implies (1), based on finiteness and uniqueness of maximum extension. (5) and (6) are obviously equivalent. □

The piecewise linear definition of V: Brin's definition of nV extends the definition of V as given in [19]; the latter is based on piecewise linear actions on the interval $[0,1] \subset \mathbb{R}$. We use half-open intervals, so neighboring intervals do not intersect; however, when the right boundary is 1, we use "1]". The boundary-points of the subintervals that appear are binary rational numbers (i.e., the denominator is a power of 2). A string $s = s_1 \dots s_m \in \{0,1\}^*$ with $m = |s|$ determines the half-open subinterval $[0.s,\ 0.s{+}2^{-|s|}[$; but if $s{+}2^{-|s|} = 1$ then we take $[0.s,\ 1]$, i.e., in that case we close the interval. Here, $0.s$ is a rational number written in fractional *binary representation*; i.e., $0.s = \sum_{i=1}^{m} s_i\, 2^{-i}$. E.g., 01100 (of length 5) determines the subinterval $[0.011,\ 0.011 + 2^{-5}[= [0.011,\ 0.01101[$.

2.2. Right ideals of nA^*

Here we completely develop the string description of nV, which is briefly alluded to in [14, subsection 4.3]. A hybrid string-geometric description was used in [11] (where some crucial concepts appear only in geometric form). Our description is entirely based on strings, but the correspondence with geometric concepts is often pointed out. The present subsection focuses on finitely generated right ideals of nA^*; in the next subsection, nV will be defined based on right-ideal morphisms of nA^*.

As before, let A be a finite alphabet of cardinality $k \geq 1$, usually denoted by $\{0, \dots, k-1\}$ or $\{a_0, \dots, a_{k-1}\}$. The n-fold cartesian product $\mathsf{X}_{i=1}^{n} A^*$ will be denoted by nA^*; we choose this notation in analogy with the notation nV, and also in order to avoid confusion with n-fold concatenation (of the form $S^n = \{s_1 \cdot \ldots \cdot s_n : s_1, \dots, s_n \in S\} \subseteq A^*$). Similarly, nA^ω denotes the n-fold cartesian product $\mathsf{X}_{i=1}^{n} A^\omega$. Multiplication in nA^* is done coordinatewise, i.e., nA^* is the direct product of n copies of the free monoid A^*. For $u \in nA^*$ we denote the coordinates of u by $u_i \in A^*$, for $1 \leq i \leq n$; i.e., $u = (u_1, \dots, u_n)$.

Geometrically: $x = (x_1, \dots, x_n) \in n\{0,1\}^*$ represents the hyperrectangle $\mathsf{X}_{i=1}^{n} [0.x_i,\ 0.x_i{+}2^{-|x_i|}[$ (except that "$0.x_i{+}2^{-|x_i|}[$" is replaced by "1]" if $0.x_i{+}2^{-|x_i|} = 1$).

The measure of this hyperrectangle is $2^{-(|x_1| + \cdots + |x_n|)}$. In particular, $(\varepsilon)^n$ represents $[0,1]^n$ and has measure 1.

The concept of prefix is similar to the one in A^*, but in order to avoid confusion we will use the phrase "initial factor". So the *initial factor order* on nA^* is defined for $u, v \in nA^*$ by $u \leq_{\text{init}} v$ iff there exists $x \in nA^*$ such that $ux = v$. In a similar way we have the concepts of comparability (denoted by $\|_{\text{init}}$), right ideal, generating set of a right ideal, and essential right ideal. It is easy to prove that $u \leq_{\text{init}} v$ in nA^* iff $u_i \leq_{\text{pref}} v_i$ for all $i = 1, \ldots, n$. For any $u, v \in nA^*$ there exists a unique $\leq_{\text{init}}$-maximum common initial factor, denoted by $u \wedge v$. In terms of coordinates, $(u \wedge v)_i = u_i \wedge_{\text{pref}} v_i$, where $u_i \wedge_{\text{pref}} v_i$ is the longest common prefix of the strings u_i and v_i.

An *initial factor code* is a set $S \subset nA^*$ such that no two different elements of S are $\leq_{\text{init}}$-comparable.

As we shall see, a crucial way in which nA^* with $n \geq 2$ differs from A^* concerns the *join* operation with respect to $\leq_{\text{init}}$. For all n, the join of $u, v \in nA^*$ is defined by $u \vee v = \min_{\leq_{\text{init}}}\{z \in nA^* \colon u \leq_{\text{init}} z \text{ and } v \leq_{\text{init}} z\}$. Of course, $u \vee v$ does not always exist.

Definition 2.4. A set $S \subset nA^*$ is *joinless* iff no two elements of S have a join with respect to $\leq_{\text{init}}$ in nA^*. Joinless sets will be called *joinless codes*, since they are necessarily initial factor codes.

A set $S \subset nA^*$ is a *maximal* joinless code iff S is $\subseteq$-maximal among the joinless codes of nA^*. (In other words, adding a new element to a maximal joinless code S results in a set, some of whose elements have joins.)

A right ideal $R \subseteq nA^*$ is called *joinless generated* iff R is generated, as a right ideal, by a joinless code.

(About the grammar: "Joinless**ly** generated" would not make sense since it is not the generating process that is joinless.)

Examples: Not every initial factor code is joinless; e.g., $\{(\varepsilon, 0),\ (0, \varepsilon)\}$ is an initial factor code where $(\varepsilon, 0) \vee (0, \varepsilon) = (0, 0)$. An example of a maximal joinless code is $\{(\varepsilon, 0),\ (0, 1),\ (1, 1)\}$. A maximal joinless code is usually not maximal as an initial factor code; for example, in $\{(\varepsilon, 0),\ (0, 1),\ (1, 1)\}$ one could add $(00, \varepsilon)$; the result would be a initial factor code (that is not joinless). The only maximal joinless code that is also maximal as an initial factor code is $\{(\varepsilon, \varepsilon)\}$.

From here on, a joinless code will be called maximal if it is maximal as a joinless code.

Connection with the geometric description: For $u, v \in nA^*$ we have $v \leq_{\text{init}} u$ iff the hyperrectangle u is contained in the hyperrectangle v (i.e., $\leq_{\text{init}}$ corresponds to $\supseteq$); note that "shorter" n-tuples correspond to "larger" hyperrectangles. The join $u \vee v$ represents the hyperrectangle obtained by intersecting the hyperrectangles u and v (so $\vee$ corresponds to $\cap$). Note that $u \vee v$ does not exist iff the intersection is the empty set (since the empty set is not a hyperrectangle). The meet $u \wedge v$ (which always exists) does not represent the union, nor the smallest hyperrectangle that contains u and v, but

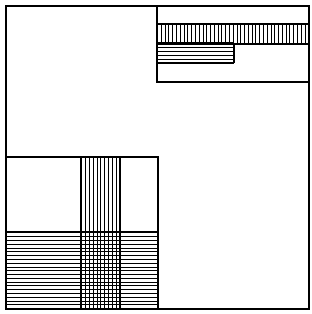


Fig. 1. A few elements of $2\{0,1\}^*$.

the smallest hyperrectangle representable by an n-tuple in nA^* that contains u and v. Joinlessness of a code means that any two hyperrectangles in the chosen subdivision of $[0,1]^n$ are *disjoint* as sets. A joinless code is maximal iff its hyperrectangles form a *tiling* of $[0,1]^n$. In an initial factor code, $\leq_{\text{init}}$-incomparability means that no hyperrectangle in the code is contained in another one.

Examples (for the correspondence between strings and geometry): Fig. 1 shows a few elements of $2\{0,1\}^*$. The large square $[0,1]\times[0,1]$ is represented by $(\varepsilon,\varepsilon)$. The numbers use fractional binary representation; e.g., $0.1101 = \frac{1}{2}+\frac{1}{4}+\frac{1}{16}$.

$(0,00) \in 2\{0,1\}^*$ represents the rectangle $[0,\ 0.1[\times[0,\ 0.01[$ (horizontally hashed); and $(010,0)$ represents $[0.01,\ 0.011[\times[0,\ 0.1[$ (vertically hashed). The join $(010,00) = (0,\ 00)\vee(010,\ 0)$ represents $[0.01,\ 0.011[\times[0,\ 0.01[$ (doubly hashed). $(0,0) = (0,00)\wedge(010,0)$ represents $[0,0.1[\times[0,0.1[$.

The rectangle $[0.1,\ 0.11[\times[0.1101,\ 0.111[$ is represented by $(10,1101)$ (horizontally hashed). And $[0.1,\ 1]\times[0.111,\ 0.1111]$ is represented by $(1,1110)$ (vertically hashed). Here, $(10,1101)\vee(1,1110)$ does not exist, and the meet $(10,1101)\wedge(1,1110) = (1,11)$ represents $[0.1,\ 1]\times[0.11,\ 1]$.

For $u,v\in A^*$, $u\vee_{\text{pref}} v$ exists in A^* iff u and v have a common upper bound for $\leq_{\text{pref}}$. This holds iff $u \parallel_{\text{pref}} v$; in that case, $u\vee_{\text{pref}} v = u$ if $v\leq_{\text{pref}} u$, and $u\vee_{\text{pref}} v = v$ if $u\leq_{\text{pref}} v$. Hence in A^*, prefix codes are the same thing as joinless codes. This is not the case for nA^* with $n\geq 2$; here, joinless codes are a special case of initial factor codes, and the join is characterized as follows:

Lemma 2.5 *(Join for $\leq_{\text{init}}$ in nA^*). For all $u=(u_1,\ldots,u_n)$, $v=(v_1,\ldots,v_n)\in nA^*$, the following are equivalent:*

(1) *the join $u\vee v$ (with respect to $\leq_{\text{init}}$) exists;*
(2) *u and v have a common upper bound for $\leq_{\text{init}}$, i.e., $(\exists z)\,[\,u\leq_{\text{init}} z$ and $v\leq_{\text{init}} z\,]$;*
(3) *for all $i=1,\ldots,n$: $u_i \parallel_{\text{pref}} v_i$ in A^*.*

Moreover, if $u\vee v = ((u\vee v)_i : i=1,\ldots,n)$ exists, then

$$(u\vee v)_i \;=\; \begin{cases} u_i & \text{if } v_i\leq_{\text{pref}} u_i \text{ (in } A^*),\\ v_i & \text{if } u_i\leq_{\text{pref}} v_i \text{ (in } A^*).\end{cases}$$

In other words, if $u \vee v$ exists then $(u \vee v)_i = \max_{\leq_{\mathrm{pref}}}\{u_i, v_i\}$, and $|(u \vee v)_i| = \max\{|u_i|, |v_i|\}$.

So, in nA^* the relation $\|_{\mathrm{init}}$ is not equivalent to coordinatewise $\|_{\mathrm{pref}}$; the latter is equivalent to the existence of a join; $\|_{\mathrm{init}}$ implies (but is not equivalent to) existence of a join.

Proof. [(1) ⇒ (2)] is obvious. [(2) ⇒ (3)] is straightforward: If $u \leq_{\mathrm{init}} z$ and $v \leq_{\mathrm{init}} z$ for some $z \in nA^*$ then $ur = vt = z$ for some $s, t, z \in nA^*$. Hence, $u_i s_i = v_i t_i = z_i$, so $u_i \|_{\mathrm{pref}} v_i$ in A^*.
[(3) ⇒ (1)] Suppose $u_i \|_{\mathrm{pref}} v_i$ for all i. Then $u_i \leq_{\mathrm{pref}} v_i$ for some i, and $v_i \leq_{\mathrm{pref}} u_i$ for the other i. Hence, $(u \vee v)_i = u_i$ if $v_i \leq_{\mathrm{pref}} u_i$ (in A^*), and $(u \vee v)_i = v_i$ otherwise; so $u \vee v$ exists. □

Notation 2.6. Let $A_{\varepsilon,n} = \bigcup_{1 \leq i \leq n} \{\varepsilon\}^{i-1} \times A \times \{\varepsilon\}^{n-i}$.

Note that $A_{\varepsilon,n}$ is the unique minimum generating set of nA^* as a monoid; the cardinality is $|A_{\varepsilon,n}| = n\,|A|$.

Lemma 2.7.
(1) *Every right ideal $R \subseteq nA^*$ is generated, as a right ideal, by a unique initial factor code. (Finiteness of generating sets is not assumed here.)*
(2) *If a right ideal $R \subseteq nA^*$ is generated by a* joinless *code then the unique initial factor code that generates R is joinless.*

Proof. Let $P = R \setminus R \cdot A_{\varepsilon,n}$. We claim that P is an initial factor code that generates R, and that P is the unique such initial factor code. (We closely follow the proof of [8, Lemma 8.1(1)].)

Let us show that P generates R. Obviously, since $P \subset R$, we have $P(nA^*) \subseteq R(nA^*) = R$. Conversely, to show that $R \subseteq P(nA^*)$, consider any $r \in R$. In nA^*, r has only finitely many initial factors, hence there exists a (not necessarily unique) $p \in R$ which is an initial factor of r and is $\leq_{\mathrm{init}}$-minimal in R. So $r = px$ for some $x \in nA^*$. And $p \notin R\,A_{\varepsilon,n}$, otherwise there would exist $p = r'a$ for some $r' \in R, a \in A_{\varepsilon,n}$, which would contradict that p is $\leq_{\mathrm{init}}$-minimal in R. Hence $p \in R \setminus R\,A_{\varepsilon,n}$.

To show that P is an initial factor code, let $p, p' \in P$ and suppose $p = p'x$ for some $x \in nA^*$. If $x \neq (\varepsilon)^n$ then $p \in RA_{\varepsilon,n}$, contradicting the assumption that $p \in P$ $(= R \setminus RA_{\varepsilon,n})$. So, $p = p'$.

To prove uniqueness of the initial factor code that generates R, we generalize the proof of [8, Lemma 8.1(1')]. If $P_1(nA^*) = P_2(nA^*)$ for two initial factor codes P_1, P_2, then for every $p_1 \in P_1$ there exists $p_2 \in P_2$ such that $p_1 = p_2 x$ (for some $x \in nA^*$). Also, there is $p_1' \in P_1$ such that $p_2 = p_1' y$ (for some $y \in nA^*$). Hence $p_1 = p_1' xy$, which implies $x = y = (\varepsilon)^n$, since P_1 is an initial factor code. Thus, $p_1 = p_2 \in P_2$. Therefore, $P_1 \subseteq P_2$. Similarly we have $P_2 \subseteq P_1$, so $P_1 = P_2$.

Part (2) follows immediately from the uniqueness of the initial factor code that generates R. □

Lemma 2.8. *Let $P \subset nA^*$ be a finite maximal joinless code. Then every $w \in nA^\omega$ has a unique initial factor in P; i.e., $(\forall w \in nA^\omega)(\exists!\, p \in P,\ u \in nA^\omega)\,[\, w = pu\,]$.*

Proof. If there were two different initial factors p, q of w in P then p and q would be initial factors of a finite initial factor of w; hence p and q would have a join, contradicting that P is joinless. This shows uniqueness.

Let us show existence. Since P is a maximal joinless code, every initial factor v of w has a join with some element of P. Let us pick v so that its coordinates (in A^*) are longer than all the coordinates of the elements of P. Then the element of P that has a join with v is an initial factor of v. □

Lemma 2.9. *Let $P \subset nA^*$ be any finite* joinless *code, and let $R = P \cdot (nA^*)$ be the right ideal generated. (Recall that by Lemma 2.7, P is uniquely determined by R.) Then the following are equivalent:*

(1) *R is an* essential *right ideal;*
(2) *P is* maximal *as a joinless code;*
(3) *$P \cdot (nA^\omega) = nA^\omega$;*
(4) *$R \cdot (nA^\omega) = nA^\omega$.*

Proof. [(1) ⇔ (2)] Suppose P is a finite joinless code. Then P is maximal joinless iff every $v \in nA^*$ has a join with some element of P (as follows directly from the definition of maximality). This is equivalent to the property that every monogenic right-ideal of nA^* intersects $P\,(nA^*)$; i.e., $P\,(nA^*)$ is essential.

[(3) ⇒ (1)] If $P\,(nA^\omega) = nA^\omega$, then every $w \in nA^\omega$ has an initial factor in P. It follows that for every right ideal $R \subset nA^*$, $R\,(nA^\omega) \subseteq P\,(nA^\omega)$. Hence R intersects $P\,(nA^*)$. So, $P\,(nA^*)$ is essential.

[(2) ⇒ (3)] Suppose P is a finite maximal joinless code. Let $w \in nA^\omega$, and for any $(i_1, \ldots, i_n) \in \mathbb{N}^n$, let $w^{(i_1,\ldots,i_n)}$ be the initial factor of w in $A^{i_1} \times \ \ldots \ \times A^{i_n}$. Then $w^{(i_1,\ldots,i_n)}$ has a join with some $p \in P$. Since P is finite, p is an initial factor of $w^{(i_1,\ldots,i_n)}$ if each of $i_1, \ldots, i_n$ is larger than $\max\{|p_i| : p \in P,\ i \in \{1, \ldots, n\}\}$. Hence, p is an initial factor of w, so $w \in P\,(nA^\omega)$. Since for every $w \in nA^\omega$ such a $p \in P$ exists (by Lemma 2.8), we conclude that $nA^\omega \subseteq P\ (nA^\omega)$.

The equivalence of (3) and (4) is obvious since $nA^* \cdot nA^\omega = nA^\omega$, so $R \cdot (nA^\omega) = P \cdot (nA^*) \cdot (nA^\omega) = P \cdot (nA^\omega)$. □

Remark. Lemma 2.9 only talks about joinless generated right ideals. Indeed, an essential finitely generated right ideal in nA^* is not necessarily joinless generated. An example for $A = \{0, 1\}$ is

$$R = \{(\varepsilon, 0), (0, \varepsilon), (1, 1)\} \cdot (2A^*).$$

It is easy to prove that R is essential, and that $\{(\varepsilon,0),\ (0,\varepsilon),\ (1,1)\}$ is an initial-factor code that is not joinless (since $(\varepsilon,0)\vee(0,\varepsilon)=(0,0)$ exists). By Lemma 2.7, this initial factor code is unique, i.e., R is not generated by any other initial-factor code; hence R is not joinless generated.

Section 5 of version 1 of [10] gives a detailed proof (independently of Lemma 2.7) that R is essential in $2A^*$, and that R is not generated (as a right ideal) by any finite joinless code.

DAGs and nA^*: The following generalizes the well known concepts of the tree of A^* and the tree of a prefix code. We abbreviate *directed acyclic graph* by DAG. A few definitions: The *leaves* of a DAG are the vertices of out-degree 0; all the other vertices are *interior vertices*. For a DAG D, the sub-DAG spanned by the interior vertices of D is called the *interior* DAG of D. The *sources* of a DAG are the vertices of in-degree 0; if there is only one source, and all vertices are reachable from this source, this source is called the *root*, and the DAG is then called *rooted*. The *depth* of a vertex v in a rooted DAG is defined to be the length of the shortest path from the root to v; by "path" we will always mean a directed path.

• *The* DAG *of* nA^* is the infinite rooted DAG with vertex set nA^* and root $(\varepsilon)^n$; the edges are the ordered pairs $(s,t)\in(nA^*)\times(nA^*)$ such that there exists $i\in\{1,\ldots,n\}$ and $a\in A$ with $t=(s_1,\ldots,s_{i-1},\ s_ia,\ s_{i+1},\ \ldots,s_n)$ (where $s=(s_1,\ldots,s_{i-1},\ s_i,\ s_{i+1},\ \ldots,s_n)$). Hence every vertex has $|A_{\varepsilon,n}|$ $(=n\cdot|A|)$ children; see Notation 2.6. And $u\leq_{\text{init}} v$ iff there exists a directed path from u to v in the DAG. It is easy to show that the depth of a vertex $v=(v_1,\ldots,v_n)$ in the DAG of nA^* is $\sum_{i=1}^n|v_i|$.

The DAG of nA^* is the *right Cayley graph* of the monoid nA^* over the generating set $A_{\varepsilon,n}$.

• For any finite subset $P\subset nA^*$ we define the *initial factor* DAG *of* P (also called the P-DAG): This is a finite rooted subDAG of the DAG of nA^*; the root of the P-DAG is the root of the DAG of nA^*; the vertices and edges are those vertices, respectively edges, of the DAG of nA^* that appear on any path from the root to any vertex in P. Hence the vertices of the P-DAG are all the initial factors of the elements of P (so the P-DAG is uniquely determined by P). The set of leaves of the P-DAG is P iff P is an initial factor code.

Note that the trees and DAGs considered here are not ordered trees or DAGs; i.e., the children of a vertex are defined as a set, not a sequence; similarly, the leaves form a set, not a sequence.

Lemma 2.10. *Let $P\subset nA^*$ be a finite maximal joinless code such that $P\neq\{\varepsilon\}^n$. Let $v=(v_1,\ldots,v_n)$ be any leaf of the interior* DAG *of the* DAG *of P, and let v_+ be the set of children of v in the P-*DAG*; so $v_+=v\cdot A_{\varepsilon,n}\cap P$, and v_+ is non-empty (since v is an interior vertex).*

(0) *Then v_+ satisfies*

$$v_+ \subseteq \{(v_1, \ldots, v_{i-1}, v_i a, v_{i+1}, \ldots, v_n) : a \in A\} = v \cdot (\{\varepsilon\}^{i-1} \times A \times \{\varepsilon\}^{n-i}),$$

for some $i \in \{1, \ldots, n\}$; and i is unique (for a given v and P).

(1) *For $n = 1$, part* (0) *holds with equality for every leaf v of the interior* DAG*: $v_+ = \{va : a \in A\}$.*

(2) (Lawson and Vdovina [28, Thm. 12.11], but with a different formalism.) *For $n = 2$ and $|A| = 2$, part* (0) *holds with equality for some maximum-depth leaf v of the interior* DAG *of P:*

$$v_+ = \{(v_1 a, v_2) : a \in A\} \quad \text{or} \quad v_+ = \{(v_1, v_2 a) : a \in A\}.$$

However, equality does not necessarily hold for every interior leaf, not even for every interior leaf of maximum depth.

(3) (Lawson and Vdovina [28, Ex. 12.8]) *For $n \geq 3$ there exist finite maximal joinless codes $P \subset n\{0,1\}^*$ for which the inclusion in part* (0) *is strict. I.e., for every leaf v of the interior* DAG *and for every $i \in \{1, \ldots, n\}$:*

$$v_+ \neq \{(v_1, \ldots, v_{i-1}, v_i a, v_{i+1}, \ldots, v_n) : a \in A\}.$$

Proof. (0) Since v is interior without having interior children, it contains a least one child in P, of the form $(v_1, \ldots, v_{i-1}, v_i a, v_{i+1}, \ldots, v_n)$, for some $a \in A$, $i \in \{1, \ldots, n\}$.

Any possible child of v belongs to $v \cdot A_{\varepsilon,n}$. If, in addition to $(v_1, \ldots, v_{i-1}, v_i a, v_{i+1}, \ldots, v_n)$, v had an additional child of the form $(v_1, \ldots, v_{j-1}, v_j b, v_{j+1}, \ldots, v_n)$ with $i \neq j$ (for any $b \in A$), then P would not be joinless. Indeed, these two children have the join $(v_1, \ldots, v_j b, \ldots, v_i a, \ldots, v_n)$ (if $j < i$), or $(v_1, \ldots, v_i a, \ldots, v_j b, \ldots, v_n)$ (if $i < j$). This shows that all children of v belong to $\{(v_1, \ldots, v_{i-1}, v_i a, v_{i+1}, \ldots, v_n) : a \in A\}$ for one particular i (depending on v).

(1) For $n = 1$ the Lemma is folklore knowledge.

(2) (This result is equivalent to [28, Thm. 12.11], but the proof given here is rather different.)

Here $A = \{0, 1\}$. Let $v = (v_1, v_2)$ be a maximum-depth leaf of the interior DAG of P. Since v is an interior leaf, at least one of its children is in P. Hence either $(v_1 a, v_2) \in P$ or $(v_1, v_2 a) \in P$, for some $a \in A$.

Let us assume that $a = 0$ and that $(v_1 0, v_2) \in P$; the other cases are very similar. Since (v_1, v_2) has maximum depth in the interior DAG, $(v_1 0, v_2)$ has maximum depth in P.

If it is also the case that $(v_1 1, v_2) \in P$, then $\{(v_1 0, v_2), (v_1 1, v_2)\} \subseteq P$, and the Lemma holds. Therefore, from here on we only consider the situation where $(v_1 1, v_2) \notin P$ (but $(v_1 0, v_2) \in P$). Then there exists $(u_1, u_2) \in P \setminus \{(v_1 0, v_2)\}$ with $(u_1, u_2) \neq (v_1 1, v_2)$, such that (u_1, u_2) has a join with $(v_1 1, v_2)$. By Proposition 2.5, this is equivalent to $u_1 \parallel_{\text{pref}} v_1 1$ and $u_2 \parallel_{\text{pref}} v_2$.

This leads to four cases.

Case 1: $v_1 1 \leq_{\text{pref}} u_1$ and $v_2 \leq_{\text{pref}} u_2$.

Then $v_1 <_{\text{pref}} u_1$ and $v_2 \leq_{\text{pref}} u_2$. Since (v_1, v_2) is a leaf of the interior DAG of P, and $(u_1, u_2) \in P$, it follows that (u_1, u_2) is a child of (v_1, v_2). Since $(u_1, u_2) \notin \{(v_1 0, v_2), (v_1 1, v_2)\}$, it follows that (u_1, u_2) is of the form $(v_1, v_2 c)$ for some $c \in A$. But then (u_1, u_2) $(= (v_1, v_2 c))$ has a join with $(v_1 0, v_2) \in P$, contradicting the fact that P is joinless. So, case 1 is ruled out.

Case 2: $v_1 1 \geq_{\text{pref}} u_1$ and $v_2 \geq_{\text{pref}} u_2$; since $(u_1, u_2) \neq (v_1 1, v_2)$, at least one of these $\geq_{\text{pref}}$ is strict.

Case 2.1: $v_1 1 >_{\text{pref}} u_1$ and $v_2 \geq_{\text{pref}} u_2$:

Then (u_1, u_2) is interior, since (v_1, v_2) is interior. But (u_1, u_2) being an interior vertex contradicts the assumption that $(u_1, u_2) \in P$. So case 2.1 is ruled out.

Case 2.2: $v_1 1 = u_1$ and $v_2 >_{\text{pref}} u_2$:

Then $u_2 = v_2 cz$, for some $c \in A$ and $z \in A^*$. But now $(u_1, u_2) = (v_1 1, v_2 cz)$ has greater depth than $(v_1 0, v_2)$, which has maximum depth in P. So case 2.2 is ruled out.

Case 3: $v_1 1 \geq_{\text{pref}} u_1$ and $v_2 \leq_{\text{pref}} u_2$; since $(u_1, u_2) \neq (v_1 1, v_2)$, at least one of $\geq_{\text{pref}}$ or $\leq_{\text{pref}}$ is strict.

Case 3.1: $v_1 1 >_{\text{pref}} u_1$ and $v_2 \leq_{\text{pref}} u_2$.

Then $v_1 1 = u_1 x 1$, and $u_2 = v_2 y$ for some $x, y \in A^*$; so $v_1 = u_1 x$. But then $(u_1, u_2) \vee (v_1 0, v_2) = (u_1, v_2 y) \vee (u_1 x 0, v_2) = (u_1 x 0, v_2 y)$ exists, contradicting the fact that $\{(u_1, u_2), (v_1 0, v_2)\} \subseteq P$. So case 3.1 is ruled out.

Case 3.2: $v_1 1 = u_1$ and $v_2 <_{\text{pref}} u_2$.

Then (u_1, u_2) has greater depth than $(v_1 0, v_2)$, contradicting the fact that $(v_1 0, v_2)$ has maximum depth in P. So case 3.2 is ruled out.

Case 4: $v_1 1 \leq_{\text{pref}} u_1$ and $v_2 \geq_{\text{pref}} u_2$.

Then $u_1 = v_1 1 x$ and $v_2 = u_2 y$ for some $x, y \in A^*$. Since $(v_1 0, v_2)$ has maximum depth in P we have $|u_1| + |u_2| \leq |v_1 0| + |v_2|$, hence $|v_1| + 1 + |x| + |u_2| \leq |v_1| + 1 + |u_2| + |y|$, hence $|x| \leq |y|$. Moreover, $y \neq \varepsilon$, otherwise $|x| = 0$, hence $x = \varepsilon$, hence $(u_1, u_2) = (v_1 1, v_2)$, which would imply $(v_1 1, v_2) \in P$. In summary this proves:

$$|x| \leq |y| \neq 0 \quad \text{and} \quad v_2 >_{\text{pref}} u_2.$$

Notation (used in the remainder of the proof): For any $z \in \{0,1\}^+$, let z^- denote the bitstring obtained by complementing the right-most bit of z. And $z\{0,1\}^{-1}$ denotes the bitstring obtained by removing the right-most bit of z.

Note that since $(v_1 0, v_2) \in P$, if we prove that $(v_1 0, v_2^-) \in P$ then the Lemma holds for the interior vertex $(v_1 0,\ v_2\{0,1\}^{-1})$.

Claim. $(v_1 0, v_2^-) \in P$.

Proof of the Claim. Assume by contradiction that there exists $(w_1, w_2) \in P$ such that $(w_1, w_2) \neq (v_1 0, v_2^-)$, and (w_1, w_2) has a join with $(v_1 0, v_2^-)$. The existence of this join is equivalent to $w_1 \parallel_{\text{pref}} v_1 0$ and $w_2 \parallel_{\text{pref}} v_2^-$.

This leads to four cases.

Case 4.1: $w_1 \leq_{\text{pref}} v_1 0$ and $w_2 \leq_{\text{pref}} v_2^-$. At least one of the $\leq_{\text{pref}}$ is strict.
Case 4.1.1: $w_1 \leq_{\text{pref}} v_1 0$ and $w_2 \leq_{\text{pref}} v_2^- A^{-1} = v_2 A^{-1}$.

Then the join $(w_1, w_2) \vee (v_1 0, v_2) = (v_1 0, v_2)$ exists, contradicting the fact that (w_1, w_2) and $(v_1 0, v_2)$ belong to P. So case 4.1.1 is ruled out.
Case 4.1.2: $w_1 \leq_{\text{pref}} v_1$ and $w_2 \leq_{\text{pref}} v_2^-$.

Then $v_1 = w_1\alpha$ and $v_2^- = u_2 y^- = w_2\beta$ for some $\alpha, \beta \in A^*$. The latter equality implies that $w_2 \parallel_{\text{pref}} u_2$. Recall that in case 4, $u_1 = v_1 1x$; this and $v_1 = w_1\alpha$ imply that $u_1 = w_1\alpha 1x$, hence $u_1 \parallel_{\text{pref}} w_1$. Now, since $u_1 \parallel_{\text{pref}} w_1$ and $w_2 \parallel_{\text{pref}} u_2$, the join $(w_1, w_2) \vee (u_1, u_2)$ exists, which contradicts the fact that (w_1, w_2) and (u_1, u_2) belong to P. So case 4.1.2 is ruled out.

Case 4.2: $w_1 \geq_{\text{pref}} v_1 0$ and $w_2 \geq_{\text{pref}} v_2^-$.

Since $(v_1 0, v_2)$ has maximum depth in P, and $(v_1 0, v_2^-)$ has the same depth, it follows that $(w_1, w_2) = (v_1 0, v_2^-)$. This contradicts the assumption $(w_1, w_2) \neq (v_1 0, v_2^-)$. So case 4.2 is ruled out.

Case 4.3: $w_1 \leq_{\text{pref}} v_1 0$ and $w_2 \geq_{\text{pref}} v_2^-$.
Case 4.3.1: $w_1 = v_1 0$ and $w_2 >_{\text{pref}} v_2^-$ (since $(w_1, w_2) \neq (v_1 0, v_2^-)$, equality in the first coordinate implies strictness in the second).

Then $|w_1| + |w_2| > |v_1 0| + |v_2^-| = |v_1 0| + |v_2|$, i.e., (w_1, w_2) has greater depth than $(v_1 0, v_2)$, which contradicts the fact that $(v_1 0, v_2)$ has maximum depth in P. So case 4.3.1 is ruled out.
Case 4.3.2: $w_1 <_{\text{pref}} v_1 0$ and $w_2 \geq_{\text{pref}} v_2^-$.

Then $w_1 \leq_{\text{pref}} v_1 = w_1 s$, and $w_2 = v_2^- t = u_2 y^- t$, for some $s, t \in A^*$. Recall that $y \neq \varepsilon$ in case 4. Then $(w_1, w_2) \vee (u_1, u_2) = (w_1, u_2 y^- t) \vee (w_1 s, u_2) = (w_1 s, u_2 y^- t)$ exists. This contradicts the fact that (w_1, w_2) and (u_1, u_2) belong to P. So case 4.3.2 is ruled out.

Case 4.4: $w_1 \geq_{\text{pref}} v_1 0$ and $w_2 \leq_{\text{pref}} v_2^-$; since $(w_1, w_2) \neq (v_1 0, v_2^-)$, $\leq_{\text{pref}}$ or $\geq_{\text{pref}}$ is strict.
Case 4.4.1: $w_1 >_{\text{pref}} v_1 0$ and $w_2 = v_2^-$.

Then $|w_1| + |w_2| > |v_1 0| + |v_2^-| = |v_1 0| + |v_2|$, hence (w_1, w_2) has greater depth than $(v_1 0, v_2)$, which contradicts the fact that $(v_1 0, v_2)$ has maximum depth in P. So case 4.4.1 is ruled out.
Case 4.4.2: $w_1 \geq_{\text{pref}} v_1 0$ and $w_2 <_{\text{pref}} v_2^-$.

Then $w_1 = v_1 0 s$; also, $w_2 <_{\text{pref}} v_2$ (since v_2 and v_2^- only differ in the right-most bit), so $v_2 = w_2 t$, for some $s, t \in A^*$. Now, $(w_1, w_2) \vee (v_1 0, v_2) = (v_1 0 s, w_2) \vee (v_1 0, w_2 t) = (v_1 0 s, w_2 t)$ exists. This contradicts the fact that (w_1, w_2) and $(v_1 0, v_2)$ belong to P. So case 4.4.2 is ruled out.

Since we now ruled out all sub-cases of case 4, this completes the proof (by contradiction) of the Claim.

Summary of the proof so far: We have $(v_1 0, v_2) \in P$ for some maximum-depth vertex (v_1, v_2) in the interior of the P-DAG. (The cases where, instead, we have $(v_1 1, v_2)$ or $(v_1, v_2 0)$, or $(v_1, v_2 1)$ in P, are similar.)

If we also have $(v_1 1, v_2) \in P$ then the Lemma holds.

If $(v_1 1, v_2) \notin P$ then there exists $(u_1, u_2) \in P$ that has a join with $(v_1 1, v_2)$. Four cases are possible, of which cases 1, 2, and 3 were ruled out. In case 4 we showed that $(v_1 0, v_2^-) \in P$; hence in case 4, $(v_1 0, v_2)$ and $(v_1 0, v_2^-)$ belong to P, i.e., the Lemma holds for the interior vertex $(v_1 0, v_2\{0,1\}^{-1})$.

The following is an example where not every maximum-depth interior leaf has two children in P. Consider the maximal joinless code $P = \{(0,0), (0,1), (1,\varepsilon)\}$. Here the interior leaf $v = (\varepsilon, 0)$ has maximum depth, and has only one child in P (namely $(0,0)$). Nevertheless, there is another maximum-depth interior leaf, namely $(0,\varepsilon)$, that has two children in P (namely $(0,0)$ and $(0,1)$).

(3) Example (from [28, Ex. 12.8]): Let $P = \{(0,0,\varepsilon),\ (1,\varepsilon,0),\ (\varepsilon,1,1),\ (0,1,0), (1,0,1)\} \subset 3\{0,1\}^*$. It is easy to verify that P is a finite maximal joinless code, and that no leaf of the interior DAG has two children in P. □

Remark about Lemma 2.10: Version 1 of this paper [10] stated incorrectly that "for every $n \geq 1$ and every leaf v of the interior DAG of P: $v_+ = v \cdot (\{\varepsilon\}^i \times A \times \{\varepsilon\}^{n-i-1})$ (for some i, $0 \leq i < n$)". This statement had to be modified for $n = 2$ (from "for every leaf" to "there exists a leaf"), and dropped for $n \geq 3$. The above counter-example for $n \geq 3$ was given in [28, Ex. 12.8].

Lemma 2.11. *Let $P \subset nA^*$ be a finite set. For any $p = (p_1, \ldots, p_n) \in P$ and $i \in \{1, \ldots, n\}$, let*

$$P'_{p,i} \ = \ (P \setminus \{p\}) \ \cup \ \{(p_1, \ldots, p_{i-1}, p_i a, p_{i+1}, \ldots, p_n) : a \in A\}.$$

Then we have:

(1) *P is joinless iff $P'_{p,i}$ is joinless.*
(2) *P is a maximal joinless code iff $P'_{p,i}$ is a maximal joinless code.*

The set $P'_{p,i}$ is called a one-step restriction *of P ("restriction" because $P'_{p,i} \cdot (nA^*) \subsetneqq P \cdot (nA^*)$); and P is called a* one-step extension *of $P'_{p,i}$. Clearly, $|P'_{p,i}| = |P| - (|A| - 1)$.*

Proof. **(1)** [⇒] Let us assume that P is joinless. For any $a, a' \in A$ with $a \neq a'$, the join of $(p_1, \ldots, p_{i-1}, p_i a, p_{i+1}, \ldots, p_n)$ and $(p_1, \ldots, p_{i-1}, p_i a', p_{i+1}, \ldots, p_n)$ does not exist, since $p_i a$ and $p_i a'$ are not prefix-comparable.

If $q \in P \setminus \{p\}$ and $(p_1, \ldots, p_{i-1}, p_i a, p_{i+1}, \ldots, p_n)$ were both initial factors of some $z \in nA^*$, then q and p would also both be initial factors of z, contradicting the assumption that P is joinless.

Finally, all pairs $q_1, q_2 \in P \setminus \{p\}$ $(\subset P'_{p,i})$ are joinless since P is joinless. Thus $P'_{p,i}$ is joinless.

[⇐] Let us assume that $P'_{p,i}$ is joinless. Then every pair $q_1, q_2 \in P \setminus \{p\}$ $(\subset P'_{p,i})$ is joinless.

If $q \in P \setminus \{p\}$ and p had a join z, then both p and q would be initial factors of z. By Lemma 2.5, $z_j = \max\{q_j, p_j\}$ for all $j \in \{1, \ldots, n\}$. We have two cases.

Case 1: $z_i = p_i$ (for the i used in $P'_{p,i}$).

This is equivalent to q_i being a prefix of p_i. Then q_i is a prefix of $p_i a$ as well (for every $a \in A$), hence $q \in P \setminus \{p\}$ and $(p_1, \ldots, p_{i-1}, p_i a, p_{i+1}, \ldots, p_n)$ have a join. But this contradicts the assumption that $P'_{p,i}$ is joinless.

Case 2: $z_i \neq p_i$, and $z_i = q_i$ (for the i used in $P'_{p,i}$).

Then p_i is a strict prefix of q_i $(= z_i)$, hence $p_i a$ is a prefix of q_i for some $a \in A$. It follows that z has q and $(p_1, \ldots, p_{i-1}, p_i a, p_{i+1}, \ldots, p_n)$ as initial factors; this contradicts the assumption that $P'_{p,i}$ is joinless.

(2) [$\Rightarrow$] Suppose P is a maximal joinless code. Hence, every $x \in nA^*$ has a join with some $q \in P$ (otherwise x could be added to P, which would contradict that P is maximal joinless). We want to show that x also has a join with some element of $P'_{p,i}$.

If $q \neq p$ then $q \in P'_{p,i}$, hence x also has a join with some $q \in P'_{p,i}$.

If $q = p$, i.e., $z = x \vee p$, then $z_j = \max\{x_j, p_j\}$ for all $j \in \{1, \ldots, n\}$. We have two cases.

Case 1: $z_i = p_i$ (for the i used in $P'_{p,i}$).

This is equivalent to x_i being a prefix of p_i. Then x_i is a prefix of $p_i a$ too (for every $a \in A$), hence x and $(p_1, \ldots, p_{i-1}, p_i a, p_{i+1}, \ldots, p_n)$ have a join. So, x has a join with some element of $P'_{p,i}$.

Case 2: $z_i \neq p_i$, and $z_i = x_i$ (for the i used in $P'_{p,i}$).

Then p_i is a strict prefix of x_i $(= z_i)$, hence $p_i a$ is a prefix of x_i for some $a \in A$. It follows that z has x and $(p_1, \ldots, p_{i-1}, p_i a, p_{i+1}, \ldots, p_n)$ as initial factors; this implies that x has a join with $(p_1, \ldots, p_{i-1}, p_i a, p_{i+1}, \ldots, p_n) \in P'_{p,i}$ (for this particular $a \in A$).

[$\Leftarrow$] Suppose that $P'_{p,i}$ is maximal joinless. Then every $x \in nA^*$ has a join with some $q \in P'_{p,i}$. We want to show that x also has a join with some element of P.

If $q \neq (p_1, \ldots, p_{i-1}, p_i a, p_{i+1}, \ldots, p_n)$ for all $a \in A$, then $q \in P$ so x also has a join with $q \in P$.

If $q = (p_1, \ldots, p_{i-1}, p_i a, p_{i+1}, \ldots, p_n)$ for some $a \in A$, then let z be the join of x and $(p_1, \ldots, p_{i-1}, p_i a, p_{i+1}, \ldots, p_n)$. Then z has x and $(p_1, \ldots, p_{i-1}, p_i a, p_{i+1}, \ldots, p_n)$ as initial factors, hence p is an initial factor of z. Hence $x \vee p$ exists, so x has a join with an element of P. $\square$

The properties of joinless codes given in Lemma 2.11 do not hold for initial factor codes in general. For example, for $A = \{0, 1\}$ consider the initial factor code $P = \{(\varepsilon, 0), (0, \varepsilon)\}$. Then for $p = (0, \varepsilon)$ and $i = 2$ we obtain $P'_{p,i} = \{(\varepsilon, 0), (0, 0), (0, 1)\}$, which is not an initial factor code.

The process of one-step restriction or extension can be iterated, which inspires the following definition and the algorithm.

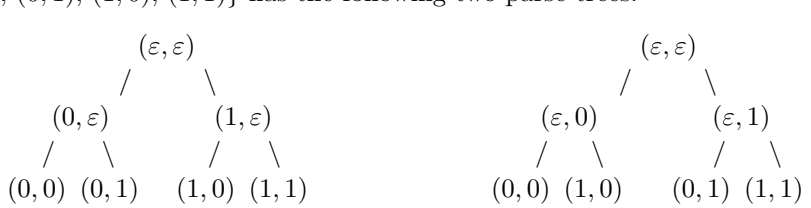
Definition 2.12 *(Parse trees).* Let $P \subset nA^*$ be a finite joinless code. A *parse tree of* P is any subtree T of the DAG of P with the following properties:

(1) The root of T is $(\varepsilon)^n$ (i.e., the root of the DAG of P); and the set of leaves of T is P (i.e., the leaves of the DAG of P).

(2) For every interior vertex v of T the set of children in T is $v \cdot (\{\varepsilon\}^{i-1} \times A \times \{\varepsilon\}^{n-i})$, for a unique $i \in \{1, \ldots, n\}$. So v has exactly $|A|$ children in T.

Given the DAG of P and a subtree T, it is easy to check whether T is a parse tree of P; one just needs to check that $(\varepsilon)^n$ occurs in T, and that every vertex in T is reachable from $(\varepsilon)^n$; moreover, for each vertex v of T one checks whether it is in P, or whether its set of children is of the form $v \cdot (\{\varepsilon\}^{i-1} \times A \times \{\varepsilon\}^{n-i})$. Recall the DAGs and trees are not oriented (children and leaves are not ordered).

A maximal joinless code P can have more than one parse tree. E.g., the joinless set $\{(0,0), (0,1), (1,0), (1,1)\}$ has the following two parse trees:



Burillo and Cleary [18] give a similar tree description of tilings of $[0,1]^2$, and point out that the tree is not unique.

If P is not maximal (as a joinless code) then it has no parse tree (according to our definition of parse tree).

By Lemma 2.10(2), every maximal joinless code in $2\{0,1\}^*$ has at least one parse tree. But in nA^* with $n \geq 3$ there are maximal joinless codes that have no parse tree, by Lemma 2.10(3); geometrically, codes in $3\{0,1\}^*$ without parse tree correspond to tilings of the cube that cannot be obtained by successive bipartitions of cuboids (perpendicularly to an axis). This motivates the following.

Questions. Is there a simple geometric or combinatorial characterization of the finite maximal joinless codes in nA^* (for $n \geq 3$) that have no parse tree? Is the non-existence of a parse tree equivalent to the presence of one of certain joinless subsets ("forbidden patterns")? An example of such a forbidden pattern is the subset $\{(0,0,\varepsilon), (1,\varepsilon,0), (\varepsilon,1,1)\}$ of Lawson and Vdovina [28], used in 2.10(3).

The following algorithm nondeterministically constructs any parse tree of P, if a parse tree exists. If P has no parse tree the algorithm will discover this for some (but not all) nondeterministic choices. For a finite joinless code $P \subset 2\{0,1\}^*$, the deterministic version of the algorithm decides whether P is maximal (as a joinless code).

Outline of the algorithm: Initially, the algorithm puts P into T (as its leaf set), and makes a working copy P_0 of P. The algorithm keeps looking for an initial factor v of an

element of P_0 such that $v \cdot (\{\varepsilon\}^{i-1} \times A \times \{\varepsilon\}^{n-i}) \subseteq P_0$ (for some $i \in \{1, \ldots, n\}$). When such a v is found, it is added to T and to P_0; and $v \cdot (\{\varepsilon\}^{i-1} \times A \times \{\varepsilon\}^{n-i})$ is removed from the working copy P_0. If $(\varepsilon)^n$ is reached, and put into T, the construction of T is complete and the algorithm concludes that P is maximal (as a joinless code), and that it has a parse tree.

The algorithm can be made deterministic by picking a total order for nA^* (e.g., the lexicographic dictionary order), and always picking the first v that works.

Notation. $\mathsf{init}(P_0)$ denotes the set of *strict* initial factors of the elements of P_0; because of strictness (and since P_0 is joinless), $P_0 \cap \mathsf{init}(P_0) = \varnothing$.

Algorithm.

INPUT: A finite set $P \subset nA^*$, given by a list of n-tuples of strings in A^*.
PRECONDITION: $P \neq \{(\varepsilon)^n\}$, and P is joinless. (This can easily be checked, by Lemma 2.5.)
OUTPUT: A set of vertices $V(T)$ and edges $E(T)$ of a parse tree of P, if P has a parse tree;

$P_0 := P$; # P_0 is a working copy of P
$V(T) := P$; $E(T) := \varnothing$;
while $(\exists v \in \mathsf{init}(P_0))\ (\exists i \in \{1, \ldots, n\})\ [\, v \cdot (\{\varepsilon\}^{i-1} \times A \times \{\varepsilon\}^{n-i}) \subseteq P_0\,]$:
    choose any v that satisfies the while-condition;
    # for a deterministic algorithm, pick the first v that works (in a fixed total order)
    $V(T) := V(T) \cup \{v\}$;
    $E(T) :=$ $E(T) \cup$ set of all edges from v to the elements of $v \cdot (\{\varepsilon\}^{i-1} \times A \times \{\varepsilon\}^{n-i})$;
    $P_0 := (P_0 \setminus v \cdot (\{\varepsilon\}^{i-1} \times A \times \{\varepsilon\}^{n-i})) \cup \{v\}$; # Hence P_0 remains joinless.
if $(\varepsilon)^n \in V(T)$:
    then output $(V(T), E(T))$ and conclude that P is maximal;
    else (in case $n = 2$ and $A = \{0, 1\}$) conclude that P is not maximal
        (and hence has no parse tree). □

Proposition 2.13. *Let P be any finite joinless code in $2\{0,1\}^*$.*

Then P has a parse tree iff P is maximal as a joinless code.

The Algorithm (deterministic version) decides maximality of P and finds a parse tree in polynomial time, when P is given as a list of pairs of bitstrings.

Proof. The Algorithm uses one-step extensions of maximal joinless codes; by Lemma 2.11, each one-step extension or restriction preserves joinlessness and maximality. Since $\{\varepsilon\}^n$ is a maximal joinless code, it follows that P is maximal if the root $(\varepsilon)^n$ is reached. It follows also that if the root is reached, a parse tree of P exists (and the Algorithm returns such a tree).

Conversely (for $n = 2$ and $A = \{0,1\}$), if P (or, at any later stage, P_0) is maximal, then by Lemma 2.10(2) there exists v in the interior DAG such that $v \cdot (\{\varepsilon\} \times \{0,1\} \cup \{0,1\} \times \{\varepsilon\}) \subseteq P$ (or $\subseteq P_0$). And this process does not stop until $P_0 = \{\varepsilon\}^n$. □

Corollary 2.14 *(Cardinality of joinless codes). Let n be any positive integer and A any finite alphabet.*

(0.1) *For every $k_1, \ldots, k_n \in \mathbb{N}$: $\mathsf{X}_{i=1}^{n} A^{k_i}$ is a maximal joinless code that has a parse tree.*
(0.2) *For any finite joinless code $P \subset nA^*$: P is maximal iff P can be transformed into $\mathsf{X}_{i=1}^{n} A^{k_i}$ by a finite sequence of restriction steps, where $k_i = \max\{|v_i| : (v_1, \ldots, v_n) \in P\}$ for $1 \le i \le n$.*

(1) *For every finite maximal joinless code $P \subseteq nA^*$ there exists $N \in \mathbb{N}$ such that*

$$|P| = 1 + (|A| - 1) \cdot N.$$

(1.1) *If P has a parse tree then P can be obtained from $\{\varepsilon\}^n$ by a finite sequence of one-step restrictions. The number of one-step restrictions used is equal to the number of interior vertices of every parse tree of P, and is equal to $N = (|P| - 1)/(|A| - 1)$.*
(1.2) *If P has no parse tree, then P can be obtained from $\{\varepsilon\}^n$ by a finite sequence of one-step restrictions, followed by a finite sequence of one-step extensions.*

Even when P has no parse tree, $N = (|P| - 1)/(|A| - 1)$ is still the number of interior vertices in any parse tree of any maximal joinless code that has a parse tree and that has the same cardinality as P (e.g., of the form $P_1 \times \{\varepsilon\}^{n-1}$ where P_1 is a prefix code in A^*).
(2) *Conversely, for all $N \in \mathbb{N}$ there are maximal joinless codes in nA^* of cardinality $1 + (|A| - 1) \cdot N$. In particular, when $|A| = 2$ every positive integer is the cardinality of some maximal joinless code.*

Proof. (0.1) Let $C(k_1, \ldots, k_i, \ldots, k_n) = \mathsf{X}_{i=1}^{n} A^{k_i}$. Let us prove by induction on $\sum_{i=1}^{n} k_i$ that $C(k_1, \ldots, k_i, \ldots, k_n)$ has a parse tree. For $C(0 \ldots, 0, \ldots, 0) = \{\varepsilon\}^n$, the parse tree consists of one vertex. Inductively,

$$\begin{aligned} &C(k_1, \ldots, k_{i-1}, k_i + 1, k_{i+1}, \ldots, k_n) = \mathsf{X}_{j=1}^{i-1} A^{k_j} \times A^{k_i+1} \times \mathsf{X}_{j=i+1}^{n} A^{k_j} \\ &= (\mathsf{X}_{j=1}^{n} A^{k_j}) \cdot (\{\varepsilon\}^{i-1} \times A \times \{\varepsilon\}^{n-i-1}) \\ &= C(k_1, \ldots, k_i, \ldots, k_n) \cdot (\{\varepsilon\}^{i-1} \times A \times \{\varepsilon\}^{n-i-1}) \\ &= \bigcup_{v \in A^{k_i}} \big((\mathsf{X}_{j=1}^{i-1} A^{k_j} \times \{v\} \times \mathsf{X}_{j=i+1}^{n} A^{k_j}) \cdot (\{\varepsilon\}^{i-1} \times A \times \{\varepsilon\}^{n-i-1})\big). \end{aligned}$$

So, $C(k_1, \ldots, k_{i-1}, k_i + 1, k_{i+1}, \ldots, k_n)$ is obtained form $C(k_1, \ldots, k_i, \ldots, k_n)$ by $|A|^{k_i}$ one-step restrictions (one one-step restriction for every $v \in A^{k_i}$). It follows that if $C(k_1, \ldots, k_i, \ldots, k_n)$ has a parse tree then $C(k_1, \ldots, k_{i-1}, k_i + 1, k_{i+1}, \ldots, k_n)$ has a parse tree. Moreover, any joinless code that has a parse tree is maximal.

(0.2) Let

$$\ell_i(P) = \max\{|v_i| : (v_1, \ldots, v_n) \in P\}, \text{ for } 1 \le i \le n; \text{ and}$$
$$\nu(P) = \textstyle\prod_{i=1}^{n} |A|^{\ell_i(P)} - \sum_{u \in P} \sum_{i=1}^{n} |u_i|.$$

The fact that P can be restricted to $C(\ell_1(P), \ldots, \ell_n(P))$ follows by induction on $\nu(P)$:
If $\nu(P) = 0$ then $P = C(\ell_1(P), \ldots, \ell_n(P))$.

If $\nu(P) > 0$, and $(u_1, \ldots, u_n) \in P$ is such that $|u_i| < \ell_i(P)$ for some i, then a one-step restriction decreases $\nu(P)$, as $(u_1, \ldots, u_n)$ is replaced by $(u_1, \ldots, u_n) \cdot (\{\varepsilon\}^{i-1} \times A \times \{\varepsilon\}^{n-i-1})$.

(1.1) We prove the equivalent statement that from P one can reach $\{\varepsilon\}^n$ by $N = (|P| - 1)/(|A| - 1)$ ones-step extensions. We use induction on $|P|$. When $|P| = 1$ then $P = \{\varepsilon\}^n$, and the formula holds. For $|P| > 1$, an extension step can be applied to some leaf of the interior of a parse tree of P, by Lemmas 2.10 and 2.11. In this extension step, a new maximal joinless code Q is obtained; one leaf of the interior the parse tree of P becomes a leaf of the parse tree of Q, so this parse tree of Q has $N - 1$ interior vertices; and $|Q| = |P| - (|A| - 1)$. By induction, $N - 1 = (|Q| - 1)/(|A| - 1)$; and the latter is equal to $(|P| - (|A| - 1) - 1)/(|A| - 1) = (|P| - 1)/(|A| - 1) - 1$. Hence $N = (|P| - 1)/(|A| - 1)$.

(1.2) By applying one-step restrictions as in part (0.2), from any maximal joinless code P one can reach $\mathsf{X}_{i=1}^{n} A^{k_i}$, where k_i is as in part (0.2). And from $\mathsf{X}_{i=1}^{n} A^{k_i}$ one can reach $\{\varepsilon\}^n$ by one-step extensions by (0.1). In any one-step restriction or extension the cardinality of the maximal joinless code increases or decreases by $|A| - 1$. So, to reach P from $\{\varepsilon\}^n$ we can apply restrictions to reach $\mathsf{X}_{i=1}^{n} A^{k_i}$, then apply extensions to obtain P.

(1) The formula follows from (1.1) and (1.2).
(2) For the existence of codes of the given cardinality, take for example $Q \times \{\varepsilon\}^{n-1}$, where Q is any maximal prefix code in A^*, and apply the corresponding result for maximal prefix codes (which is folklore; see e.g. [6, Lemma 9.9(0)]). □

Proposition 2.15. *There exist* polynomial-time algorithms *that on input $P \subset nA^*$ (a finite set, given by an explicit list of n-tuples of strings) decide whether P has the following properties:*
(1) *P is* joinless*;*
(2) *P is* maximal *as a joinless code.*

Proof. The input to the algorithms is P, given as a list of n-tuples of strings, so the input size is $\sum_{p \in P} \sum_{i=1}^{n} |p_i|$.

(1) Lemma 2.5, applied to every two elements $u, v \in P$ with $u \neq v$, will decide in quadratic time whether P is joinless.
(2) For finite joinless codes in $2\{0,1\}^*$, the Algorithm given after Definition 2.12 has polynomial time complexity, in view of Corollary 2.14 which proves that every parse tree of P has size that is linearly bounded in terms of $|P|$.

For nA^* in general, the Algorithm that follows Proposition 2.17, based on the generalized Kraft equality, decides in polynomial time whether a joinless code is maximal. □

The corresponding questions about initial factor codes are also decidable in polynomial time. Suppose $P \subset nA^*$ is finite and given by an explicit list of n-tuples of strings. It is

easy to decide whether P is an initial factor code; it is sufficient to check for every two elements $u, v \in P$ with $u \neq v$, whether $u \|_{\text{init}} v$. A code P is maximal as an initial factor code iff for every interior vertex v of the P-DAG, the $n\,|A|$ children of v in the nA^*-DAG are also children of v in the P-DAG.

An algorithm for testing maximality of a joinless code can be derived from the following generalization of the Kraft (in)equality to higher dimensions. We mentioned earlier that in the geometric description of the Brin-Thompson groups, a word $x = (x_1, \ldots, x_n) \in n\,\{0,1\}^*$ represents the hyperrectangle $\mathsf{X}_{i=1}^{n}[0.x_i\,,\ 0.x_i + 2^{-|x_i|}[$ (where we close the intervals whose right-bound is 1). The measure of this hyperrectangle is $2^{-(|x_1| + \ldots + |x_n|)}$. More generally, we have the following.

Definition 2.16. Let A be an alphabet of cardinality $|A| = k \geq 2$. For every $x = (x_1, \ldots, x_n) \in nA^*$ we define the measure

$$\mu(x) = k^{-(|x_1| + \ldots + |x_n|)}.$$

For every joinless code $P \subset nA^*$ (not necessarily finite) we define the measure

$$\mu(P) = \sum_{x \in P} \mu(x).$$

Proposition 2.17 *(n-dimensional Kraft (in)equality). Let $P \subset nA^*$ be a finite joinless code, where $|A| \geq 2$ and $n \geq 1$. Then we have:*

(1) $\mu(P) \leq 1$.
(2) *P is maximal (as a joinless code) iff* $\mu(P) = 1$.

Proof. This follows from the geometric picture. For a joinless code P, all the words in P represent non-overlapping hyperrectangles in $[0,1]^n$, so their total measure is at most the measure of $[0,1]^n$, which is 1.

And P is maximal iff the corresponding hyperrectangles tile $[0,1]^n$, which is iff the sum of the measures of the hyperrectangles is 1. □

Proposition 2.17 probably holds for infinite joinless codes too; but since we don't need it in that case, we'll that question open.

Proposition 2.17 leads to the following algorithm.

Algorithm (maximality of a finite joinless code).
INPUT: A finite set $P \subset nA^*$, given as an explicit list of words.
PRECONDITION: P is joinless. (This is easily checked, by Proposition 2.15(1).)
QUESTION: Is P maximal?

Compute $\mu(P) = \sum_{x \in P} k^{-\sum_{i=1}^{n} |x_i|}$ in fractional base-k representation;
if $\mu(P) = 1$, output "yes";
else, output "no". □

This algorithm runs in polynomial time, in terms of the total input length $\sum_{x\in P}\sum_{i=1}^{n}|x_i|$. In fractional base-$k$ representation the sum $\mu(P)$ is easy to compute.

We will need the intersection of joinless generated right ideals, and the elementwise join of joinless codes.

Proposition 2.18. *Let $P, Q \subset nA^*$ be joinless codes.*

(1) *The* elementwise join $P \vee Q$, *defined by*

$$P \vee Q \;=\; \{p \vee q \;:\; p \in P,\; q \in Q\},$$

is a joinless code. (Here, $p \vee q$ ranges over the joins that exist.)

Hence, $|P \vee Q| \;\leq\; |P| \cdot |Q|$.

(2) *P and Q are both maximal (as joinless codes) iff $P \vee Q$ is maximal.*

(3) $(P \vee Q) \cdot (nA^*) \;=\; P \cdot (nA^*) \;\cap\; Q \cdot (nA^*)$.

Hence, if $P\,(nA^*)$ and $Q\,(nA^*)$ are joinless generated then so is $P\,(nA^*) \cap Q\,(nA^*)$.

Proof. (1) Suppose $p, p' \in P$, $q, q' \in Q$, and $p \neq p'$ or $q \neq q'$. Then $(p \vee q) \vee (p' \vee q')$ does not exist, because $(p \vee q) \vee (p' \vee q')$ would have p, p', q, and q' as prefixes. But either p and p' (if different) or q and q' (if different) do not have a join.

(2) [$\Leftarrow$] If $P \vee Q$ is maximal then every $x \in nA^*$ has a join with some $p \vee q \in P \vee Q$, i.e., x and $p \vee q$ are initial factors of some $z \in nA^*$. Then p and q are also initial factors of z, so $x \vee p$ and $x \vee q$ exist. Hence, every $x \in nA^*$ has a join with some $p \in P$ and some $q \in Q$, thus P and Q are maximal.

[$\Rightarrow$] If P is maximal then every $x \in nA^*$ has a join with some $p \in P$; and if Q is maximal, $x \vee p$ has a join with some $q \in Q$. Hence, x, p, and q, are all initial factors of some word z, hence $z \vee p \vee q$ exists. So, every $x \in nA^*$ has a join with some $p \vee q$, so $P \vee Q$ is maximal.

(3) [$\supseteq$] Every $w \in P\,(nA^*) \cap Q\,(nA^*)$ satisfies $w = pu = qv$ for some $p \in P$, $q \in Q$, and $u, v \in nA^*$. This implies that p and q are initial factors of w, so $p \vee q$ exists, and is an initial factor of w. Hence, $w \in (P \vee Q) \cdot (nA^*)$.

[$\subseteq$] If $p \vee q$ exists then it has p and q as initial factors, hence $p \vee q \in P\,(nA^*) \cap Q\,(nA^*)$. □

2.3. Right ideal morphisms of nA^*, and string-based definition of $nG_{k,1}$ and nV

Just as for A^*, one defines the concepts of *right ideal morphism*, domain code, and image code in nA^*. We only consider domain and image codes that are *joinless*. Indeed, if $P \subset nA^*$ is not joinless, some definitions of right ideal morphisms on P will be inconsistent. E.g., let $P = \{(0,\varepsilon), (\varepsilon,0)\}$, so $(0,\varepsilon) \vee (\varepsilon,0) = (0,0)$; and let $f(0,\varepsilon) = (0,0)$ and $f(\varepsilon,0) = (1,1)$; then $f(0,0) = f((0,\varepsilon)\cdot(\varepsilon,0)) = (0,0)\cdot(\varepsilon,0) = (0,00) \neq (10,1) = (1,1)\cdot(0,\varepsilon) = f((\varepsilon,0)\cdot(0,\varepsilon)) = f(0,0)$; so $f(0,0)$ receives two different values.

Before we get to $nG_{k,1}$ we define the following monoid:

Definition 2.19.

$n\mathcal{RI}_A^{\mathsf{fin}} = \{f : f$ is a right ideal morphism of nA^* such that f is injective, and $\mathrm{domC}(f)$ and $\mathrm{imC}(f)$ are *finite, maximal, joinless* codes$\}$.

"Maximal" means maximal as a joinless code. Usually we just write $n\mathcal{RI}^{\mathsf{fin}}$ when a fixed alphabet A is used.

Lemma 2.20. *For every $f \in n\mathcal{RI}^{\mathsf{fin}}$: $f(\mathrm{domC}(f)) = \mathrm{imC}(f)$.*

Hence, if $f \in n\mathcal{RI}^{\mathsf{fin}}$ then $f^{-1} \in n\mathcal{RI}^{\mathsf{fin}}$, and $\mathrm{domC}(f^{-1}) = \mathrm{imC}(f)$, $\mathrm{imC}(f^{-1}) = \mathrm{domC}(f)$.

Proof. For every $p_1 \in \mathrm{domC}(f)$: $f(p_1) = q_1 u \in \mathrm{Im}(f)$, for some $q_1 \in \mathrm{imC}(f)$ and $u \in nA^*$. Since $q_1 \in \mathrm{Im}(f)$, $q_1 = f(p_2 v)$ for some $p_2 \in \mathrm{domC}(f)$ and $v \in nA^*$. Hence, $q_1 u = f(p_2 v)\ u = f(p_2 vu)$. Thus, $f(p_1) = q_1 u = f(p_2 vu)$. Since f is injective, this implies that $p_1 = p_2 vu$. Since $p_1, p_2 \in \mathrm{domC}(f)$, which is an initial factor code, $p_1 = p_2$ and $u = v = (\varepsilon)^n$. Hence, $f(p_1) = q_1 u = q_1 \in \mathrm{imC}(f)$. So $f(\mathrm{domC}(f)) \subseteq \mathrm{imC}(f)$.

Conversely, if $q \in \mathrm{imC}(f)$, then $q = f(p)\, v$ for some $p \in \mathrm{domC}(f)$ and $v \in nA^*$. Since $f(p) \in \mathrm{Im}(f)$ and $q \in \mathrm{imC}(f)$ (which is the initial factor code that generates $\mathrm{Im}(f)$), we conclude that $q = f(p)$ and $v = (\varepsilon)^n$. Hence, $q \in f(\mathrm{domC}(f)$. So, $\mathrm{imC}(f) \subseteq f(\mathrm{domC}(f))$.

Now f^{-1} satisfies the following: For $q \in \mathrm{imC}(f)$, $f^{-1}(q) = p$ iff $p \in \mathrm{domC}(f)$ and $f(p) = q$. Hence $f^{-1} \in n\mathcal{RI}^{\mathsf{fin}}$, and $\mathrm{domC}(f^{-1}) = \mathrm{imC}(f)$, and $\mathrm{imC}(f^{-1}) = \mathrm{domC}(f)$. □

Lemma 2.21. *Let $f \in n\mathcal{RI}^{\mathsf{fin}}$ and let $P \subset nA^*$ be a finite set.*

(1.1) *If $P \subset \mathrm{Dom}(f)$ we have: $f(P)$ is joinless iff P is joinless.*

(1.2) *If $P \subset \mathrm{Dom}(f)$ and P is joinless, we have: P is maximal iff $f(P)$ is maximal.*

(2.1) *In general (not assuming $P \subset \mathrm{Dom}(f)$), we have:*

$f(P \vee \mathrm{domC}(f))$ *is joinless iff P is joinless.*

(2.2) *In general, if P is joinless then the following are equivalent:*

P *is maximal,*
$P \vee \mathrm{domC}(f)$ *is maximal,*
$f(P \vee \mathrm{domC}(f))$ *is maximal.*

Proof. (1.1) [⇐] Let $p, q \in P$, and assume by contradiction that there exists $z \in nA^*$ such that $f(p)$ and $f(q)$ are initial factors of z. Then $z = f(p)\, u = f(q)\, v$ for some $u, v \in nA^*$. Hence, $f^{-1}(z) = f^{-1}(f(p)\, u) = f^{-1}(f(p))\ u$; the latter holds since $f^{-1} \in n\mathcal{RI}^{\mathsf{fin}}$, and $f(p) \in \mathrm{Dom}(f^{-1}) = \mathrm{Im}(f)$ (by Lemma 2.20). Hence, $f^{-1}(z) = pu$. Similarly, $f^{-1}(z) = qv$. So, $pu = qv$, but that contradicts the assumption that P is joinless.
(1.1) [⇒] Conversely, if some $p, q \in P$ have a join z then $z = pu = qv$ for some $u, v \in nA^*$. Then $f(z) = f(p)\, u = f(q)\, v$, so $f(p) \vee f(q)$ exists, hence $f(P)$ is not joinless.
(1.2) [⇒] Suppose P is maximal, and assume by contradiction that $f(P)$ is not maximal. Then there exists $x \in nA^*$ such that $\{x\} \cup f(P)$ is a joinless code. Since $f^{-1} \in n\mathcal{RI}^{\mathsf{fin}}$,

$f^{-1}(\{x\} \cup f(P))$ is joinless (by what was proved in the previous paragraph). So, $f^{-1}(\{x\} \cup f(P)) = P \cup \{f^{-1}(x)\}$ is joinless, which contradicts P the assumption that P is maximal. Thus, if P is maximal then $f(P)$ is maximal.
(1.2) [$\Leftarrow$] Similarly, if $f(P)$ is maximal then $f^{-1}f(P)$ is maximal (since $f^{-1} \in n\mathcal{RI}^{\mathsf{fin}}$). Hence if $f(P)$ is maximal, P is maximal.
(2.1) If P is joinless iff $P \vee \mathrm{domC}(f)$ is joinless, by Lemma 2.18(1), since $\mathrm{domC}(f)$ is joinless for all $f \in n\mathcal{RI}^{\mathsf{fin}}$. And $P \vee \mathrm{domC}(f)$ is joinless iff $f(P \vee \mathrm{domC}(f))$ is joinless, by (1.1).
(2.2) If P is maximal then $P \vee \mathrm{domC}(f)$ is maximal by Lemma 2.18(2), since $\mathrm{domC}(f)$ is maximal for $f \in n\mathcal{RI}^{\mathsf{fin}}$. This implies that $f(P \vee \mathrm{domC}(f))$ is maximal, by (1.2). And if $f(P \vee \mathrm{domC}(f))$ is maximal then $P \vee \mathrm{domC}(f)$ is maximal, again by (1.2). Moreover, maximality of $P \vee \mathrm{domC}(f)$ implies maximality of P (and of $\mathrm{domC}(f)$), by Lemma 2.18(2). □

Every right ideal morphism $f \in n\mathcal{RI}^{\mathsf{fin}}$ is uniquely determined by its restriction to $\mathrm{domC}(f)$; this is an obvious consequence of the fact that f is a right-ideal morphism and $\mathrm{domC}(f)$ is a joinless code. So f is determined by the finite function $f\colon \mathrm{domC}(f) \to \mathrm{imC}(f)$.

Conversely, let $P, Q \subset nA^*$ be two finite maximal joinless codes with the same cardinality, and let $F\colon P \to Q$ by any bijection from P onto Q. Then F determines a right ideal morphism f of nA^*, such that F is the restriction of f to its domain code; f is defined in a unique way by $f(pv) = F(p)\, v$ for all $p \in P$, $v \in nA^*$. Since P is joinless, f is well defined.

Definition 2.22 *(Table).* A bijection $F\colon P \to Q$ between finite maximal joinless codes $P, Q \subset nA^*$ is called a *table*.

Tables and right ideal morphisms in $n\mathcal{RI}^{\mathsf{fin}}$ determine each other bijectively, and can the treated as "the same thing".

Every function $f \in n\mathcal{RI}^{\mathsf{fin}}$ determines a permutation $f^{(\omega)}$ of nA^ω, as follows. For any $w \in nA^\omega$ there exists a unique $p \in \mathrm{domC}(f)$ such that $w = pu$ for some $u \in nA^\omega$, by Lemma 2.8. Then we define $f^{(\omega)}$ by

$$f^{(\omega)}(w) = f(p)\, u.$$

The converse does not hold; i.e., $f \in n\mathcal{RI}^{\mathsf{fin}}$ is not determined by $f^{(\omega)}$, as will be seen in Lemma 2.24.

Definition 2.23 *(End-equivalence).* Two right ideal morphisms $f, g \in n\mathcal{RI}^{\mathsf{fin}}$ are *end-equivalent* iff f and g agree on $\mathrm{Dom}(f) \cap \mathrm{Dom}(g)$. This will be denoted by $f \equiv_{\mathrm{end}} g$.

By Proposition 2.18, $\mathrm{Dom}(f) \cap \mathrm{Dom}(g)$ is generated by a joinless code, namely $\mathrm{domC}(f) \vee \mathrm{domC}(g)$.

In [9] the congruence $\equiv_{\mathrm{end}}$ is defined in much greater generality, and other congruences are introduced.

Lemma 2.24. *For all $f, g \in n\mathcal{RI}^{\mathsf{fin}}$: $f \equiv_{\mathrm{end}} g$ iff $f^{(\omega)} = g^{(\omega)}$.*

Proof. For every $f \in n\mathcal{RI}^{\mathsf{fin}}$, $\mathrm{domC}(f)$ and $\mathrm{imC}(f)$ are maximal joinless codes. Therefore (by Lemma 2.9): $\mathrm{domC}(f) \cdot (nA^\omega) = nA^\omega = \mathrm{imC}(f) \cdot (nA^\omega)$. And by Lemma 2.18, $\mathrm{domC}(f) \vee \mathrm{domC}(g)$ is also a maximal joinless code.

Let $R = \mathrm{Dom}(f) \cap \mathrm{Dom}(g)$, and let $f|_R$ and $g|_R$ be the restrictions of f or g to R. Then $f \equiv_{\mathrm{end}} g$ is equivalent to $f|_R = g|_R$.

[$\Rightarrow$] Suppose $f \equiv_{\mathrm{end}} g$, i.e., $f|_R = g|_R$, where $R = \mathrm{Dom}(f) \cap \mathrm{Dom}(g)$. For every $w \in nA^\omega$, let $z \in nA^*$ be an initial factor of w such that in all coordinates, z is longer than the longest coordinate of any element of $P = \mathrm{domC}(f) \vee \mathrm{domC}(g)$. And $w = zu$ for some $u \in nA^\omega$. Since P is a maximal joinless code, z has a join with an element of P; by the chosen length of z, z has an initial factor in P, hence $z \in R$. Now $f^{(\omega)}(zu) = f(z)\, u$, since $z \in R \subseteq \mathrm{Dom}(f)$; and $g^{(\omega)}(zu) = g(z)\, u$, since $z \in R \subseteq \mathrm{Dom}(g)$. Since $f(z) = g(z)$ (because $f|_R = g|_R$), it follows that $f^{(\omega)}(zu) = g^{(\omega)}(zu)$.

[$\Leftarrow$] Suppose $f^{(\omega)} = g^{(\omega)}$. For every $r \in R$ and every $u \in nA^\omega$, $f^{(\omega)}(ru) = g^{(\omega)}(ru)$. And since $r \in R = \mathrm{Dom}(f) \cap \mathrm{Dom}(g)$, $f^{(\omega)}(ru) = f(r)\, u$, and $g^{(\omega)}(ru) = g(r)\, u$. From $f(r)\, u = g(r)\, u$ it follows that $f(r) = g(r)$. Hence, $f|_R = g|_R$, i.e., $f \equiv_{\mathrm{end}} g$. □

Lemma 2.25. *For all $f_1, f_2 \in n\mathcal{RI}^{\mathsf{fin}}$: $(f_2 \circ f_1)^{(\omega)} = f_2^{(\omega)} \circ f_1^{(\omega)}$.*

The relation $\equiv_{\mathrm{end}}$ is a congruence *on $n\mathcal{RI}^{\mathsf{fin}}$.*

Proof. For every $w \in nA^\omega$ there exist $r \in \mathrm{Dom}(f_2 \circ f_1)$ and $u \in nA^\omega$ such that $w = ru$; this follows from Lemma 2.8. Then $r \in \mathrm{Dom}(f_1)$ and $f_1(r) \in \mathrm{Dom}(f_2)$. Now by the definition of $f^{(\omega)}(w)$, $((f_2 \circ f_1)^{(\omega)}(w) = (f_2 \circ f_1)(r)\, u = f_2(f_1(r))\, u$. And $f_2^{(\omega)}(f_1^{(\omega)}(ru)) = f_2^{(\omega)}(f_1(r)\, u) = f_2(f_1(r))\, u$; the latter holds since $f_1(r) \in \mathrm{Dom}(f_2)$. This proves that $(f_2 \circ f_1)^{(\omega)}(w) = f_2^{(\omega)}(f_1^{(\omega)}(w)$.

It follows immediately that $\equiv_{\mathrm{end}}$ is a congruence on $n\mathcal{RI}^{\mathsf{fin}}$ (by Lemma 2.24). □

Next we develop criteria about extensions and restrictions of functions in $n\mathcal{RI}^{\mathsf{fin}}$ that enable us to decide efficiently whether two tables determine end-equivalent functions. The Remark below applies to finite maximal joinless codes in $2\{0,1\}^*$ and is similar to a criterion for end-equivalence of finite maximal prefix codes in A^*. But because of Lemma 2.10(3) it does not apply for $n \geq 3$. For nA^* in general, Proposition 2.26 gives an efficient algorithm for deciding whether two tables determine end-equivalent functions.

Remark *(Extension-restriction criterion).* Let P, Q be finite maximal joinless codes in $2\{0,1\}^*$, let $F\colon P \to Q$ be a table, and let $f \in 2\mathcal{RI}^{\mathsf{fin}}$ be the corresponding right ideal morphism of $2\{0,1\}^*$. Then: f is extendable in $2\mathcal{RI}^{\mathsf{fin}}$ iff there exist $p = (p_1, p_2)$, $q = (q_1, q_2) \in 2\{0,1\}^*$ such that for every $a \in \{0,1\}$:

(1) $\{(p_1, p_2a) : a \in \{0,1\}\} \subseteq P$ (or $\{(p_1a, p_2) : a \in \{0,1\}\} \subseteq P$), and
(2) $\{(q_1, q_2a) : a \in \{0,1\}\} \subseteq Q$ (or $\{(q_1a, q_2) : a \in \{0,1\}\} \subseteq Q$), and
(3) $F(p_1, p_2a) = (q_1, q_2a)$ (or $F(p_1a, p_2) = (q_1a, q_2)$).

In that case, let

$P' = (P \setminus \{(p_1, p_2a) : a \in \{0,1\}\}) \cup \{p\}$, (or $(P \setminus \{(p_1a, p_2) : a \in \{0,1\}\}) \cup \{p\}$), and

$Q' = (Q \setminus \{(q_1, q_2a) : a \in \{0,1\}\}) \cup \{q\}$, (or $(Q \setminus \{(q_1a, q_2) : a \in \{0,1\}\}) \cup \{q\}$).

Then P' and Q' are finite maximal joinless codes in $2\{0,1\}^*$, and f can be extended to a function $f' \in 2\mathcal{RI}^{\mathsf{fin}}$ with table F': $P' \to Q'$ defined by

$F'(p) = q$, and

$F'(p') = F(p')$ for all $p' \in P \setminus \{(p_1, p_2a) : a \in \{0,1\}\}$ (or $P \setminus \{(p_1a, p_2) : a \in \{0,1\}\}$).

The passage from f to f' is called a *one-step extension*, and f is called a *one-step restriction* of f'.

The Remark follows from Lemma 2.10(2), in the same way as for prefix codes in A^* (see [5, Lemma 2.2] and [25]). The Remark is not always applicable when $n \geq 3$, by Lemma 2.10(3).

Proposition 2.26 *(Restrictions, and deciding $\equiv_{\mathrm{end}}$).*
(1) *Let $F : P \to Q$ be a table, and let $f \in n\mathcal{RI}_A^{\mathsf{fin}}$ be the right-ideal morphism given by this table. Suppose $P' \cdot (nA^*) \subseteq P \cdot (nA^*)$, where $P' \subset nA^*$ is a finite maximal joinless code. Then the* restriction $f' = f|_{P' \cdot (nA^*)}$ *of f to $P' \cdot (nA^*)$ is an element of $n\mathcal{RI}_A^{\mathsf{fin}}$ with table $F' : P' \to f(P')$.*

Moreover, $f \equiv_{\mathrm{end}} f|_{P' \cdot (nA^*)}$.

(2) *Let $F^{(j)} : P^{(j)} \to Q^{(j)}$ be tables (for $j = 1, 2$), and let $f^{(j)} \in n\mathcal{RI}_A^{\mathsf{fin}}$ be the right-ideal morphisms given by these tables. Then:*

$$f^{(1)} \equiv_{\mathrm{end}} f^{(2)} \text{ iff } F^{(1)}|_{P^{(1)} \vee P^{(2)}} = F^{(2)}|_{P^{(1)} \vee P^{(2)}} ,$$

where $F^{(j)}|_{P^{(1)} \vee P^{(2)}}$ is the table of the restriction of $f^{(j)}$ to the finite maximal joinless code $P^{(1)} \vee P^{(2)}$.

Hence there is a polynomial-time algorithm that decides whether the tables $F^{(1)}$ and $F^{(2)}$ represent $\equiv_{\mathrm{end}}$-equivalent elements of $n\mathcal{RI}_A^{\mathsf{fin}}$.

Proof. (1) The restricted table $F' : P' \to f(P')$ is defined as follows. For every $p' \in P'$ there exists $p = (p_1, \ldots, p_n) \in P$ and $w = (w_1, \ldots, w_n) \in nA^*$ such that $p' = pw$. We define $F'(p') = F(p)\ w$.

In order to verify that F' is a well defined function, suppose that $p' = p^{(1)}\, u = p^{(2)}\, v$ for some $p^{(1)}, p^{(2)} \in P$ and $u, v \in nA^*$. Since P is joinless, it follows that $p^{(1)} = p^{(2)}$; let $p^{(1)} = p^{(2)} = p$. Since multiplication in nA^* is cancelative, $pu = pv$ implies $u = v$. So, $p' \in P'$ determines a unique $p \in P$ and $w = u = v \in nA^*$ such that $p' = pw$. Hence $F'(p') = F(p)\ w$ defines $F'(p')$ in a unique way.

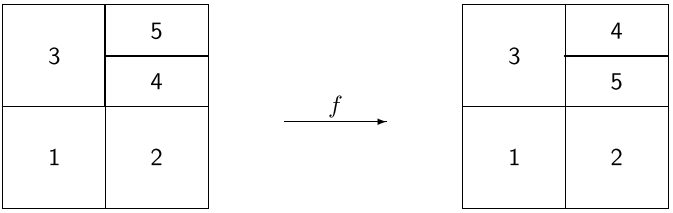


Fig. 2. Geometric representation of f (Lemma 2.27).

(2) The largest common restriction of $f^{(1)}$ and $f^{(2)}$ is $f^{(1)} \cap f^{(2)}$, which has domain $\mathrm{Dom}(f^{(1)}) \cap \mathrm{Dom}(f^{(2)}$, and domain code $\mathrm{domC}(f^{(1)} \cap f^{(2)}) = P^{(1)} \vee P^{(2)}$ (by Lemma 2.18). So, $f^{(1)} \equiv_{\mathrm{end}} f^{(2)}$ iff the tables of $f^{(1)}$ and $f^{(2)}$, restricted to $P^{(1)} \vee P^{(2)}$, are the same.

We have $|P^{(1)} \vee P^{(2)}| \leq |P^{(1)}| \cdot |P^{(2)}|$. And for all $p^{(1)} \in P^{(1)}$ and $p^{(2)} \in P^{(2)}$, $|p^{(1)} \vee p^{(2)}|_{\max} \leq \max\{|p^{(1)}|_{\max}, |p^{(1)}|_{\max}\}$. (Recall the notation $|x|_{\max} = \max\{|x_i| : 1 \leq i \leq n\}$.) Hence one can check in polynomial time whether $f^{(1)} \equiv_{\mathrm{end}} f^{(2)}$. □

Lemma 2.27 *(Non-uniqueness of maximal extensions). There exists a right ideal morphism $f \in 2\,\mathcal{RI}_2^{\mathsf{fin}}$ such that f has* **two** maximal extensions *in $2\,\mathcal{RI}_2^{\mathsf{fin}}$.*

As a consequence, nV with $n \geq 2$ cannot be defined by maximum extended morphisms (unlike V).

Proof. Let f be defined by $\mathrm{domC}(f) = \mathrm{imC}(f) = \{(0,0), (0,1), (1,0), (1,10), (1,11)\}$, and the table

x	(0,0)	(0,1)	(1,0)	(1,10)	(1,11)
$f(x)$	(0,0)	(0,1)	(1,0)	(1,11)	(1,10)

The geometric representation of f is given in Fig. 2 (with mapping-by-number as in [14]): In Fig. 2, the squares labeled "1" and "2" could be merged into one binary rectangle; alternatively, the squares labeled "1" and "3" could be merged into one binary rectangle. After either step, no further extension is possible. Thus, f has the following two maximal extensions F_1 and F_2:

(1) $\mathrm{domC}(F_1) = \mathrm{imC}(F_1) = \{(\varepsilon, 0), (0,1), (1,10), (1,11)\}$, and
$F_1 = \{((\varepsilon,0),(\varepsilon,0)),\ ((0,1),(0,1)),\ ((1,10),(1,11)),\ ((1,11),(1,10))\}$.
(2) $\mathrm{domC}(F_2) = \mathrm{imC}(F_2) = \{(0,\varepsilon), (1,0), (1,10), (1,11)\}$, and
$F_2 = \{((0,\varepsilon),(0,\varepsilon)),\ ((1,0),(1,0)),\ ((1,10),(1,11)),\ ((1,11),(1,10))\}$. □

We now give the definition of $nG_{k,1}$ and nV based on strings.

Definition 2.28 *(Brin-Thompson groups nV and $nG_{k,1}$).* Let $A = \{0, \ldots, k-1\}$ and $n \geq 2$. The Brin-Thompson group $nG_{k,1}$ is $n\,\mathcal{RI}_A^{\mathsf{fin}}/\equiv_{\mathrm{end}}$. Equivalently, $nG_{k,1}$ is the group determined by the action of $n\,\mathcal{RI}_A^{\mathsf{fin}}$ on nA^ω. When $k = 2$ we obtain nV.

Every element of $nG_{k,1}$ can be represented (in infinitely many ways) by a table of the form $F\colon P \to Q$, which is a bijection between two finite maximal joinless codes.

Lemma 2.29 *(Composition in $nG_{k,1}$ based on tables). Let $F_j : P_j \to Q_j$ be a table representing $f_j \in n\mathcal{RI}^{\mathsf{fin}}$, which in turn determines $f_j^{(\omega)} \in nG_{k,1}$ (for $j = 1, 2$). Then the* composite $f_2 \circ f_1$*, and hence also $f_2^{(\omega)} \circ f_1^{(\omega)}$, is represented by the table*

$$(f_2 \circ f_1)|_P : P \to Q, \text{ where}$$
$$P = f_1^{-1}(P_2 \vee Q_1),$$
$$Q = f_2(P_2 \vee Q_1).$$

Proof. It is a general fact about partial functions f_2, f_1, that $\mathrm{Dom}(f_2 \circ f_1) = f_1^{-1}(\mathrm{Dom}(f_2) \cap \mathrm{Im}(f_1))$, and $\mathrm{Im}(f_2 \circ f_1) = f_2(\mathrm{Dom}(f_2) \cap \mathrm{Im}(f_1))$. Obviously, $f_2 \circ f_1 = (f_2 \circ f_1)|_{\mathrm{Dom}(f_2 \circ f_1)}$.

For $f_2, f_1 \in n\mathcal{RI}^{\mathsf{fin}}$, given by tables, $\mathrm{Dom}(f_2) \cap \mathrm{Im}(f_1) = (P_2 \vee Q_1) \cdot (nA^*)$ (by Lemma 2.18). And $f_1^{-1}(P_2 \vee Q_1)$ and $f_2(P_2 \vee Q_1)$ are maximal joinless codes (by Lemmas 2.20 and 2.21). Moreover, $f_1^{-1}(\mathrm{Dom}(f_2) \cap \mathrm{Im}(f_1)) = f_1^{-1}(P_2 \vee Q_1) \cdot (nA^*)$, and $f_2(\mathrm{Dom}(f_2) \cap \mathrm{Im}(f_1)) = f_2(P_2 \vee Q_1) \cdot (nA^*)$. Hence, $(f_2 \circ f_1)|_{\mathrm{domC}(f_2 \circ f_1)}$ is given by the table described in the Lemma. □

3. The word problem of nV is in coNP

For a fixed group G with a fixed finite generating set Γ the *word problem* is the following decision problem.

INPUT: A string $w \in (\Gamma^{\pm 1})^*$.

QUESTION: Does w represent the identity element of G?

We mentioned in the Introduction that nV is finitely generated for all $n \geq 1$. The groups $nG_{k,1}$, for $k > 2$, are presumably finitely generated too, but this has not been proved, so we will only address the word problem of nV here.

Notation. We mostly use the alphabet $A = \{0, 1, \ldots, k-1\}$, usually with $k = 2$.

For any integer $j \geq 0$, let $nA^{\leq j} = \{(x_1, \ldots, x_n) \in nA^* \colon |x_i| \leq j \text{ for } i = 1, \ldots, n\}$; for a string $w \in A^*$, $|w|$ denotes the length of w.

Definition of coNP and NP: We use the following logic-based definitions of coNP and NP (see e.g., [23]). Let Γ be a finite alphabet. A set $S \subseteq \Gamma^*$ is in coNP iff there exists $m \geq 1$, a two-variable predicate $R(.,.) \subseteq mA^* \times \Gamma^*$, and a polynomial $p(.)$, such that

(1) $R \in \mathsf{P}$ (i.e., the membership problem of R is in P);

(2) $S = \{w \in \Gamma^* : (\forall x \in mA^{\leq p(|w|)})\, R(x, w)\}$.

Similarly, S is in NP iff for some $m \geq 1$, some $R(.,.) \subseteq mA^* \times \Gamma^*$ in P, and some polynomial $p(.)$,

$$S = \{w \in \Gamma^* : (\exists x \in mA^{\leq p(|w|)})\, R(x,w)\}.$$

Definition 3.1 *(max length).*
For every $z = (z_1, \ldots, z_n) \in nA^*$: $\ell(z) = \max\{|z_1|, \ldots, |z_n|\}$.
For every finite set $P \subset nA^*$: $\ell(P) = \max\{\ell(z) : z \in P\}$.
For every $f \in n\mathcal{RI}^{\mathsf{fin}}$: $\ell(f) = \max\{\ell(z) : z \in \mathrm{domC}(f) \cup \mathrm{imC}(f)\}$.

Proposition 3.2 *(Length formula). For all $f_2, f_1 \in n\mathcal{RI}^{\mathsf{fin}}$: $\ell(f_2 \circ f_1) \leq \ell(f_2) + \ell(f_1)$.*

Proof. Let $F_i : P_i \to Q_i$ be a table for f_i $(i = 1, 2)$. Recall the table for $f_2 \circ f_1$, given in Lemma 2.29. We have:

$$\ell(P_2 \vee Q_1) = \max\{\ell(P_2),\ \ell(Q_1)\}. \tag{L1}$$

Indeed, for every $p = (p_1, \ldots, p_n) \in P_2$, $q = (q_1, \ldots, q_n) \in Q_1$, and $i \in \{1, \ldots, n\}$ we have: $|(p \vee q)_i| = \max\{|p_i|, |q_i|\}$ (by Lemma 2.5).
We have:

$$\ell(f_2(P_2 \vee Q_1)) \leq \ell(Q_2) + \ell(Q_1) \leq \ell(f_2) + \ell(f_1). \tag{L2}$$

Indeed, $(p \vee q)_i = \max_{\leq_{\mathrm{pref}}}\{p_i, q_i\}$, for every $p \in P_2$, $q \in Q_1$, and $i \in \{1, \ldots, n\}$. By Proposition 2.18(3), $p \vee q \in \mathrm{Dom}(f_2)$. Since p is an initial factor of $p \vee q$ there exists $u \in nA^*$ such that $pu = p \vee q$. Since $(p \vee q)_i = \max_{\leq_{\mathrm{pref}}}\{p_i, q_i\}$, the following holds: $u_i = \varepsilon$ when $(p \vee q)_i = p_i$, and u_i is a suffix of q_i when $(p \vee q)_i = q_i$. Hence, $\ell(u) \leq \ell(q)$. Now, $f_2(p \vee q) = f_2(p)\, u$, where $f_2(p) \in Q_2$ (since $p \in P_2$). And $q \in Q_1$. Hence $\ell(f_2(p \vee q)) \leq \ell(f_2(p)) + \ell(u) \leq \ell(Q_2) + \ell(Q_1)$.
We also have:

$$\ell(f_1^{-1}(P_2 \vee Q_1)) \leq \ell(P_1) + \ell(P_2) \leq \ell(f_2) + \ell(f_1). \tag{L3}$$

Indeed, f_1^{-1} is given by the table $f_1^{-1}|_{Q_1}$: $Q_1 \to P_1$. Consider any $p \vee q$ for $p \in P_2$, $q \in Q_1$. Since q is an initial factor of $p \vee q$ there exists $v \in nA^*$ such that $qv = p \vee q$. Since $(p \vee q)_i = \max_{\leq_{\mathrm{pref}}}\{p_i, q_i\}$, the following holds: $v_i = \varepsilon$ when $(p \vee q)_i = q_i$, and v_i is a suffix of p_i when $(p \vee q)_i = p_i$. Hence, $\ell(v) \leq \ell(p)$. Now, $f_1^{-1}(p \vee q) = f_1^{-1}(q)\ v$, where $f_1^{-1}(q) \in P_1$ (since $q \in Q_1$). And $p \in P_2$. Hence $\ell(f_1^{-1}(p \vee q)) \leq \ell(f_1^{-1}(q)) + \ell(v) \leq \ell(P_1) + \ell(P_2)$.

Finally, since $\ell(f_2 \circ f_1) = \max\{\ell(f_1^{-1}(P_2 \vee Q_1)),\ \ell(f_2(P_2 \vee Q_1))\}$, we obtain:

$$\ell(f_2 \circ f_1) \leq \max\{\ell(Q_2) + \ell(Q_1),\ \ell(P_1) + \ell(P_2)\} \leq \ell(f_2) + \ell(f_1). \tag{L4}$$ □

Corollary 3.3. *Let $f_t, \ldots, f_1 \in n\mathcal{RI}^{\mathsf{fin}}$, and let $\lambda \in \mathbb{N}$ be such that $\ell(f_j) \leq \lambda$ for $j = 1, \ldots, t$. Then $\ell(f_t \circ \ldots \circ f_1) \leq \lambda\, t$.* □

Lemma 3.4. *For any $f \in n\mathcal{RI}^{\mathsf{fin}}$ and $\lambda = \ell(\mathrm{domC}(f))$ we have: $nA^\lambda \subset \mathrm{Dom}(f)$.*
Hence, $f|_{nA^\lambda}$ determines $f^{(\omega)}$. In particular, $f|_{nA^\lambda} = \mathsf{id}|_{nA^\lambda}$ iff $f^{(\omega)} = \mathbf{1}$ in $nG_{k,1}$.

Proof. By Corollary 2.14(0), nA^λ is a maximal joinless code, hence every element $p \in \mathrm{domC}(f)$ has a join with some element $u \in nA^\lambda$. Since $\lambda = \ell(\mathrm{domC}(f))$, p is actually an initial factor of u. Hence, $u \in P \cdot (nA^*)$ $(= \mathrm{Dom}(f))$. This proves that $nA^\lambda \subset \mathrm{Dom}(f)$.

For any finite maximal joinless code $P \subset \mathrm{Dom}(f)$, the restriction $f|_P\colon P \to f(P)$ is a table for $f^{(\omega)}$, hence it determines $f^{(\omega)}$. Since nA^λ is a finite maximal joinless code contained in $\mathrm{Dom}(f)$, the result follows. □

Lemma 3.5. *The word problem of nV over any finite generating set belongs to* coNP.

Proof. Let Γ be any finite generating set of nV. To simplify the notation we assume that Γ is closed under inverse, i.e., $\Gamma = \Gamma^{\pm 1}$. Every $\gamma \in \Gamma$ is represented by a table F_γ: $P_\gamma \to Q_\gamma$. For any $w \in \Gamma^*$, let $f_w \in n\mathcal{RI}^{\mathsf{fin}}$ be the function obtained by composing the generators in w (given by tables). Let $F_w\colon P \to Q$ be the table of f_w. By Proposition 3.2 and Corollary 3.3: $\ell(f_w) \le c_\Gamma\,|w|$, where $c_\Gamma = \max\{\ell(\gamma) : \gamma \in \Gamma\}$, and $|w|$ denotes the length of w as a word over Γ. So c_Γ is a known constant, determined by the finite generating set Γ.

For the word problem we have: $w = \mathbf{1}$ in nV iff $f_w^{(\omega)} = \mathsf{id}$ (the identity function on A^ω) iff $f_w = \mathsf{id}|_{\mathrm{Dom}(f_w)}$ in $n\mathcal{RI}^{\mathsf{fin}}$. Since $\mathrm{domC}(f_w) = P$ (in the table $F_w\colon P \to Q$), we have: $f_w = \mathsf{id}|_{\mathrm{Dom}(f_w)}$ iff $P = Q$ and $F_w = \mathsf{id}|_P$. By Proposition 3.2, $P \cup Q \subset nA^{\le c_\Gamma |w|}$. We can further restrict f_w to $nA^{c_\Gamma |w|} \cdot (nA^*)$; then by Lemma 3.4, we obtain the following coNP-formula for the word problem:

$$w = \mathbf{1} \text{ in } nV \text{ iff } (\forall x \in nA^{c_\Gamma |w|})\ [\, f_w(x) = x\,].$$

We still need to show that the predicate $R(x, w)$, defined by

$$R(x,w) \;\Leftrightarrow\; [\,(\forall i \in \{1,\dots,n\})[|x_i| = c_\Gamma |w|] \Rightarrow f_w(x) = x\,],$$

belongs to P. I.e., we want a deterministic polynomial-time algorithm that on input $w \in \Gamma^*$ and $x \in nA^{c_\Gamma |w|}$, checks whether $f_w(x) = x$. To do this we apply, to $x \in nA^{c_\Gamma |w|}$, the tables of the generators $\gamma_j \in \Gamma$ that appear in $w = \gamma_t \dots \gamma_1$. We compute $x \longmapsto \gamma_1(x) = y^{(1)} \longmapsto \gamma_2(y^{(1)}) = y^{(2)} \longmapsto \dots \longmapsto \gamma_t(y^{(t-1)}) = y^{(t)} = f_w(x)$. Since $x \in nA^{c_\Gamma |w|} \subseteq \mathrm{Dom}(f_w)$, every $y^{(j)}$ is defined. Moreover, $x = pu$ for some $p \in P$ and $u \in nA^{\le c_\Gamma |w|}$. By Proposition 3.2, $|y^{(j)}| \le n\,c_\Gamma\, j + |u| \le 2n\,c_\Gamma\,|w|$. After computing $y^{(t)}$ we check whether $y^{(t)} = x$.

The application of the table of γ_j to $y^{(j-1)}$ takes time proportional to $|y^{(j-1)}|$ (for $j = 1, \dots, t$). So, the time complexity of verifying whether x and w satisfy the predicate is (up to a constant multiple) $\le |x| + \sum_{j=1}^{t} |y^{(j)}| \le n\,c_\Gamma\,|w| + |w| \cdot 2\,n\,c_\Gamma\,|w|$. Hence the time-complexity of the predicate is quadratic in $|w|$. □

4. coNP-completeness of the word problem of nV

In this section we prove that the word problem of nV with $n \ge 2$, over any finite generating set, is coNP-hard with respect to polynomial-time many-one reduction. The

result for all $nV, n \geq 2$, follows quickly from the result for $2V$. We proved already in Lemma 3.5 that the word problem of nV belongs to coNP.

For $2V$, coNP-hardness of the word problem follows fairly directly from the coNP-hardness of the word problem of V over the infinite generating set $\Gamma_V \cup \tau$, by making use of the shift σ (subsection 4.6). Here, Γ_V is any finite generating set of V and τ is the set of position transpositions $\{\tau_{i,i+1} : i \geq 1\}$.

At the end of subsection 4.1 we show that the word problem of V over $\Gamma_V \cup \tau$ belongs to coNP. The main difficulty is to prove that the word problem of V over $\Gamma_V \cup \tau$ is coNP-hard; this is proved in subsections 4.2 - 4.4, by constructing a binary conjunctive polynomial-time reduction of the circuit equivalence problem to this word problem. An alternative proof, that gives a polynomial-time many-one reduction, appears in subsection 4.5.

4.1. Preliminaries on the word problem and complexity

We give some definitions and facts about complexity and the word problem of a group, especially when an infinite generating set is used. Here we use finite and infinite alphabets (but we always point out when an alphabet is infinite).

Definition 4.1. Let Σ_1, Σ_2 be two finite alphabets, and let m be a positive integer. A polynomial-time *conjunctive reduction* of arity m from $L_1 \subseteq \Sigma_1^*$ to $L_2 \subseteq \Sigma_2^*$ is a polynomial-time computable total function $\rho : \Sigma_1^* \to m\Sigma_2^*$ such that for all $x \in \Sigma_1^*$:

$$x \in L_1 \text{ iff } \rho(x) \in \mathsf{X}_{j=1}^{m} L_2 \ (= mL_2).$$

Equivalently, $L_1 = \rho^{-1}(\mathsf{X}_{j=1}^{m} L_2)$. In other words, ρ reduces the problem L_1 to m instances of the problem L_2, and the m answers are combined by "and".

A polynomial-time conjunctive reduction of arity 1 is called a *many-one* reduction.

The reductions in Definition 4.1 are a very special case of polynomial-time truth-table reductions; see e.g. [20, Def. 7.18]. It is straightforward to show that each of P, NP, and coNP, is closed under downward polynomial-time conjunctive reduction of bounded arity.

In this paper we use the following definition of coNP-hardness and coNP-completeness.

Definition 4.2. Let Σ_0 be a finite alphabet. A problem $L_0 \subseteq \Sigma_0^*$ is coNP-*hard* iff for every finite alphabet Σ and every problem $L \subseteq \Sigma^*$ there exists a polynomial-time conjunctive reduction ρ of bounded arity that reduces L to L_0.

Moreover, $L_0 \subseteq \Sigma_0^*$ is coNP-*complete* iff L_0 is coNP-hard and L_0 belongs to coNP.

There are many well-known coNP-complete problems, e.g., the tautology problem for boolean formulas, the integer linear programming equivalence problem, the 4-coloring problem, the connectivity lower-bound problem (see e.g. [22], [6, Introduction]). We will use the equivalence problem for acyclic boolean circuits (defined in Section 4.2).

Definition 4.3. Let G be a group, and let Γ ($\subseteq G$) be a (possibly infinite) generating set for G. For words $w_1, w_2 \in (\Gamma^{\pm 1})^*$ we say that *"$w_1 = w_2$ in G"* iff the generator sequences w_1 and w_2 have the same value when their elements are multiplied in G. In a similar way, for $w \in (\Gamma^{\pm 1})^*$ and $g \in G$, we say *"$w = g$ in G"* iff g is the value obtained when the elements of w are multiplied in G. We also use the notation $w_1 =_G w_2$ or $w =_G g$ for this.

To simplify the notation, we will from now on take group generating sets Γ that are closed under inverse, i.e., $\Gamma = \Gamma^{\pm 1}$.

Lemma 4.4 *(Folklore). Let G_2 be a finitely generated subgroup of a finitely generated group G_1, and let Γ_i be a finite generating set of G_i for $i = 1, 2$.*

(1) *If the word problem of G_1 over Γ_1 is decidable in deterministic (or nondeterministic, or co-nondeterministic) time $\leq t_1(.)$, then the word problem of G_2 over Γ_2 is decidable in deterministic (respectively in nondeterministic, or co-nondeterministic) time $\leq t_1(O(.))$.*
(2) *If a problem $L \subseteq \Sigma^*$ (where Σ is finite) is reducible to the word problem of G_2 over Γ_2 by a polynomial-time conjunctive reduction of arity m, then L is also reducible to the word problem of G_1 over Γ_1 by a polynomial-time conjunctive reduction of arity m.*

Proof. To simplify the notation, let us assume that Γ_1 and Γ_2 are closed under inverse.
(1) Since $G_2 \subseteq G_1$, for every generator $\gamma \in \Gamma_2$ there exists a word $w_\gamma \in {\Gamma_1}^*$ such that $\gamma =_{G_1} w_\gamma$. Then the total function

$$\rho_{2,1} : \ x_1 \dots x_n \in {\Gamma_2}^* \longmapsto w_{x_1} \cdot \dots \cdot w_{x_n} \in {\Gamma_1}^*$$

is a one-to-one linear-time reduction of the word problem of G_2 over Γ_2 to the word problem of G_1 over Γ_1; here, "$\cdot$" denotes concatenation. The length of $w_{x_1} \cdot \dots \cdot w_{x_n}$ is $\leq c\,|x_1 \dots x_n|$, where $c = \max\{|w_\gamma| : \gamma \in \Gamma_2\}$. Hence, if the word problem of G_1 over Γ_1 has time-complexity $\leq t_1(.)$, then the word problem of G_2 over Γ_2 has time-complexity $\leq t_1(cn)$ for inputs of length n.
(2) For every $\gamma \in \Gamma_2$ let $w_\gamma \in \Gamma_1^*$ be such that $\gamma = w_\gamma$ in G_1, and let $W(.)$ be the free monoid homomorphism from Γ_2^* into Γ_1^* determined by $W(\gamma) = w_\gamma$ for all $\gamma \in \Gamma_2$. Let $\rho : \Sigma^* \to m\,\Gamma_2^*$ be a polynomial-time conjunctive reduction of arity m, such that for all $v \in \Sigma^*$: $v \in L$ iff $\rho(v) = (\varepsilon)^m$. Then

$$x \in \Sigma^* \ \longmapsto \ \rho(x) = (y_1, \dots, y_m) \in m\,\Gamma_2^* \longmapsto (W(y_1), \dots, W(y_m)) \in m\,\Gamma_1^*$$

is a polynomial-time conjunctive reduction of arity m, from L to the word problem of G_1 over Γ_1. □

For the word problem of groups, *infinite generating sets* cannot always be avoided, because some groups are not finitely generated, and because some finitely generated groups have interesting infinite generating sets. In order to apply the concepts of decidability or computational complexity to groups with infinite generating sets, we encode countable generating sets over a finite alphabet. We will use the following.

Definition 4.5 *(Encoding).* An *encoding* of a countable set Γ is an injective total function $\mathsf{code} : \Gamma \to \{0,1\}^*$ such that $\mathsf{code}(\Gamma)$ is a prefix code that is accepted by a finite-state automaton.

For a word of generators $w = w_1 \dots w_m \in \Gamma^*$ we define $\mathsf{code}(w)$ by the concatenation $\mathsf{code}(w) = \mathsf{code}(w_1) \cdot \dots \cdot \mathsf{code}(w_m)$. Hence, $\mathrm{Im}(\mathsf{code}(.)) = \mathsf{code}(\Gamma^*) = (\mathsf{code}(\Gamma))^*$, which is a finite-state language.

Since the function code is injective it has an inverse function, code^{-1}, whose domain is $\mathsf{code}(\Gamma^*)$. Every countable set admits an encoding of the above type (e.g., with image set $0^*1 = \{0^n 1 : n \in \omega\}$).

The *word problem for a group G with an infinite generating set Γ and encoding* $\mathsf{code} : \Gamma \to \{0,1\}^*$ is specified as follows.

INPUT: $x \in \{0,1\}^*$.

PRECONDITION: $x \in \mathsf{code}(\Gamma)^*$. (Since $\mathsf{code}(\Gamma)^*$ is finite-state, the precondition is easy to check.)

QUESTION: $\mathsf{code}^{-1}(x) = \mathbf{1}$ in G? (Here, $\mathbf{1}$ denotes the identity element of G.)

Equivalently, the word problem is the membership problem of the language

$$\mathrm{WP}_{G,\Gamma,\mathsf{code}} = \{x \in \{0,1\}^* : \mathsf{code}^{-1}(x) = \mathbf{1} \text{ in } G\}.$$

From now on, by *complexity of the word problem* of G over Γ we mean the complexity of $\mathrm{WP}_{G,\Gamma,\mathsf{code}}$; note that the problem depends on G, Γ, and code.

Lemma 4.6. *Let G_2 be a subgroup of a countable group G_1, let Γ_i ($\subseteq G_i$) be a countable generating set of G_i, and let $\mathsf{code}_i(.)$ be an encoding of Γ_i, for $i = 1, 2$. We also assume that there is a total function $h : \Gamma_2 \to \Gamma_1^*$ with the following properties (that connect the encodings code_1 and code_2):*

- *For all $\gamma \in \Gamma_2$: $\gamma = h(\gamma)$ in G_1;*
- *the function $h_0 : \mathsf{code}_2(\gamma) \in \mathsf{code}(\Gamma_2) \longmapsto \mathsf{code}_1(h(\gamma)) \in \mathsf{code}(\Gamma_1)^*$ is computable in linear time.*

The function h is extended to a free-monoid homomorphism $\Gamma_2^* \to \Gamma_1^*$ that will also be called h; so for every $w \in \Gamma_2^*$ we have: $w = h(w)$ in G_1.

Then the following hold:

(1) *If $\mathrm{WP}_{G_1,\Gamma_1,\mathsf{code}_1}$ is in $\mathsf{DTime}(t)$ (or in $\mathsf{NTime}(t)$, or in $\mathsf{coNTime}(t)$), then $\mathrm{WP}_{G_2,\Gamma_2,\mathsf{code}_2}$ is in $\mathsf{DTime}(t(O(.)))$ (respectively in $\mathsf{NTime}(t(O(.)))$, or $\mathsf{coNTime}(t(O(.)))$).*

(2) *If $L \subseteq \Sigma^*$ (where Σ is finite) is reducible to $\mathrm{WP}_{G_2,\Gamma_2,\mathsf{code}_2}$ by a polynomial-time conjunctive reduction of arity m, then L is also reducible to $\mathrm{WP}_{G_1,\Gamma_1,\mathsf{code}_1}$ by a polynomial-time conjunctive reduction of arity m. Hence, if $\mathrm{WP}_{G_2,\Gamma_2,\mathsf{code}_2}$ is hard for a complexity class (e.g., coNP), then $\mathrm{WP}_{G_1,\Gamma_1,\mathsf{code}_1}$ is also hard for that complexity class.*

The functions h and h_0 in the Lemma have the commuting diagram

$$h_0 \circ \mathsf{code}_2(.) = \mathsf{code}_1 \circ h(.);$$

equivalently, $h_0 = \mathsf{code}_1 \circ h(.) \circ \mathsf{code}_2^{-1}(.)$, and $h(.) = \mathsf{code}_1^{-1} \circ h_0 \circ \mathsf{code}_2(.)$. Note that in general h cannot be viewed as a computable function (as opposed to h_0), since its domain and image are arbitrary countable sets.

Proof. (1) For any $w \in \Gamma_2^*$ we have: $w = \mathbf{1}$ in G_2 over Γ_2 iff $h(w) = \mathbf{1}$ in G_1 over Γ_1. Therefore, $x = \mathsf{code}_2(w) \in \mathrm{WP}_{G_2,\Gamma_2,\mathsf{code}_2}$ iff $h_0(x) = \mathsf{code}_1(h(w)) \in \mathrm{WP}_{G_1,\Gamma_1,\mathsf{code}_1}$.

Thus we have the following algorithm for the membership problem of $\mathrm{WP}_{G_2,\Gamma_2,\mathsf{code}_2}$ on input $x \in \{0,1\}^*$: First, check whether $x \in \mathsf{code}_2(\Gamma_2)^*$; this can be checked in linear time, since $\mathsf{code}_2(\Gamma_2)^*$ is finite-state. Second, compute $h_0(x)$ (in linear time). Finally, check whether $h_0(x)$ is in $\mathrm{WP}_{G_1,\Gamma_1,\mathsf{code}_1}$, in time $\leq t(|h_0(x)|) \leq t(O(|x|))$.

(2) Let $\rho : x \in \{0,1\}^* \longmapsto (y_1, \ldots, y_m) \in m\,\{0,1\}^*$ be a polynomial-time conjunctive reduction of arity m from L to $\mathrm{WP}_{G_2,\Gamma_2,\mathsf{code}_2}$. Hence, $x \in L$ iff $\{y_1, \ldots, y_m\} \subset \mathrm{WP}_{G_2,\Gamma_2,\mathsf{code}_2}$. By the definition of h and h_0, the latter holds iff $\{h_0(y_1), \ldots, h_0(y_m)\} \subset \mathrm{WP}_{G_1,\Gamma_1,\mathsf{code}_1}$. Thus the function $x \longmapsto (h_0(y_1), \ldots, h_0(y_m))$, where $(y_1, \ldots, y_m) = \rho(x)$, is a polynomial-time conjunctive reduction of arity m from L to $\mathrm{WP}_{G_1,\Gamma_1,\mathsf{code}_1}$. □

Some conventions and a fact about the Thompson group V over $\Gamma_V \cup \tau$:

We pick a finite generating set Γ_V for V, and for notational convenience we will assume that $\Gamma_V = \Gamma_V^{\pm 1}$. We also use the set of bit position transpositions $\tau = \{\tau_{j,j+1} : j \geq 2\}$. We assume $\Gamma_V \cap \tau = \varnothing$.

Definition 4.7 *(Size of a generator).* For any generator $\delta \in \Gamma_V \cup \tau$ we define the *size* $\|\delta\|$ as follows: For $\delta = \gamma \in \Gamma_V$ we let $\|\gamma\| = 1$, and for $\delta = \tau_{j,j+1} \in \tau$ we let $\|\tau_{j,j+1}\| = j + 1$. For a string of generators $w = w_m \ldots w_1$ with $w_i \in \Gamma_V \cup \tau$ for $i = 1, \ldots, m$, the *size* of w is defined by $\|w\| = \sum_{i=1}^{m} \|w_i\|$.

For the word w as above, the *length* of w is $|w| = m$.

Lemma 4.8. *The word problem of V over $\Gamma_V \cup \tau$ belongs to* coNP.

Proof. We have $\ell(\tau_{j,j+1}) = j + 1 = \|\tau_{j,j+1}\|$ (where $\ell(.)$ was defined in Definition 3.1 based on tables of elements of $n\,\mathcal{RI}^{\mathsf{fin}}$). And there is a positive integer constant c such that for all $\gamma \in \Gamma_V$: $\ell(\gamma) \leq c$. Hence, for any $w \in (\Gamma_V \cup \tau)^*$ we have (by Proposition 3.2): $\ell(w) \leq c\,\|w\|$.

Now the proof of Lemma 3.5 can be applied. For any $v \in (\Gamma_V \cup \tau)^*$, let $f_v \in \mathcal{RI}^{\mathsf{fin}}$ be the right ideal morphism of $\{0,1\}^*$ generated by v. Then for every $w \in (\Gamma_V \cup \tau)^*$ we have:

$$w = \mathbf{1} \text{ in } V \text{ iff } (\forall x \in \{0,1\}^{c\,\|w\|})\,[\,f_w(x) = x\,].$$

The predicate $R(x, w)$, defined by $[\,x \in \{0,1\}^{c\,\|w\|} \Rightarrow f_w(x) = x\,]$, is in P. This uses the same proof as Lemma 3.5, and the fact that w is encoded over $\{0,1\}^*$ in such a way that $\ell(w)$, $\|w\|$, and the length of the encoding, are linearly related. Hence the above $\forall$-formula is a coNP-formula for the word problem of V over $\Gamma_V \cup \tau$. □

Outline of the proof of coNP-hardness of the word problem of V over $\Gamma_V \cup \tau$:
In subsections 4.2 - 4.4 we follow (a part of) the strategy of [6], where another finitely presented group with coNP-complete word problem was constructed.
1. Every acyclic boolean circuit C is "simulated" by a element of V, represented by a word w_C over $\Gamma_V \cup \tau$, such that the size of w_C is polynomially bounded by the size of C (subsection 4.2, Definition 4.10 and Theorem 4.12).
2. The equivalence problem for acyclic boolean circuits is reduced (by a polynomial-time one-one reduction) to the generalized word problem of the subgroup $\mathrm{pFix}_V(0)$ in V (subsection 4.3, Corollary 4.16).
3. Thanks to the "commutation test", the generalized word problem of $\mathrm{pFix}_V(0)$ in V is reduced to two instances of the word problem of V over $\Gamma_V \cup \tau$ (subsection 4.4, Lemma 4.19). This reduction is a 2-ary conjunctive linear-time reduction ("2" comes from the fact that V is 2-generated).

4.2. Circuits and the Thompson group V

Our first step in the proof of coNP-hardness is to represent acyclic boolean circuits by words over the generating set $\Gamma_V \cup \tau$ of V.

An *acyclic boolean circuit* is specified by a directed acyclic graph (DAG) without isolated vertices, together with a vertex labeling. This labeling associates (1) an input variable with each source vertex, (2) an output variable with each sink vertex, and (3) a gate (of type NOT, FORK, AND, or OR) with each interior vertex. By definition, a source vertex is a vertex of in-degree 0; a sink vertex of out-degree 0; an interior vertex is a vertex whose in-degree and out-degree are both non-zero. A source vertex is also called *input port*, and a sink vertex is also called *output port*.

A *gate* is, by definition, a total function $\{0,1\}^m \to \{0,1\}^n$ (for some $m, n \geq 1$). We consider the following four types of gates, where $u \in \{0,1\}^{j-1}$, $x_j, x_{j+1} \in \{0,1\}$, and $v \in \{0,1\}^{n-j} \cup \{0,1\}^{n-j-1}$.

$\mathrm{NOT}_j : u\, x_j\, v \mapsto u\, \overline{x}_j v$; here, $m \geq 1$ and $j \leq n = m$;

$\mathrm{AND}_{j,j+1} : u\, x_j x_{j+1}\, v \mapsto u\, (x_j \,\&\, x_{j+1})\, v$; here, $m \geq 2$ and $j \leq m-1 = n$;

$\mathrm{OR}_{j,j+1} : u\, x_j x_{j+1}\, v \mapsto u\, (x_j \,\mathsf{or}\, x_{j+1})\, v$; here, $m \geq 2$ and $j \leq m-1 = n$;

$\mathrm{FORK}_j : u\, x_j\, v \mapsto u\, x_j\, x_j\, v$; here, $1 \leq j \leq m$, and $n = m+1$.

The operation FORK_j makes an extra copy of x_j. In traditional circuit theory, FORKs are not used separately; instead, NOT, AND, and OR are allowed to produce several copies of the output bit. However, using FORK as a separate gate simplifies the conversion of a circuit into a sequence of functions. We also use the *wire-crossing* operation, which swaps the "wires" i and j (where $1 \leq i < j \leq m$); this is the function

$\tau_{i,j} : u\, x_i\, v\, x_j\, w \longmapsto u\, x_j\, v\, x_i\, w$,

where, $u \in \{0,1\}^{i-1}$, $v \in \{0,1\}^{j-i}$, $w \in \{0,1\}^{m-j-1}$, $m \geq 2$, and $n = m$. This operation is not a gate; it is not associated with a vertex, but follows from the incidence relation of the graph.

Note that all the gates NOT_j are different functions for different values of j; the same applies to all $\text{AND}_{j,j+1}$, and all $\text{OR}_{j,j+1}$. However in the presence of the operations $\tau_{i,j}$ it is sufficient to use just one set of gates {NOT, AND, OR, FORK}, applied to bit positions 1, or 1 and 2. E.g., $\text{NOT}_i = \tau_{i,1} \circ \text{NOT}_1 \circ \tau_{i,1}$. Thus, here we view acyclic circuits as expressions over the generating set $\{\text{NOT, AND, OR, FORK}\} \cup \{\tau_{i,j} : j > i \geq 1\}$. Note that $\tau_{i,j} \in V$, with $\text{domC}(\tau_{i,j}) = \text{imC}(\tau_{i,j}) = \{0,1\}^j$.

An acyclic circuit C with sequence of input variables $(x_1, \ldots, x_m)$ (with values ranging over $\{0,1\}^m$), and sequence of output variables $(y_1, \ldots, y_n)$ (with values in $\{0,1\}^n$), determines an *input-output function* $f_C : \{0,1\}^m \to \{0,1\}^n$; this is a total function. Any total function of the form $F : \{0,1\}^m \to \{0,1\}^n$ is called a *boolean function*. In circuit theory it is proved that for every boolean function F there exists an acyclic circuit whose input-output function is F; see e.g. [23,44,39,21].

Two circuits C_1 and C_2 are called *equivalent* iff $f_{C_1} = f_{C_2}$.

The equivalence problem for acyclic boolean circuits (in short, the *circuit equivalence problem*) is specified as follows:

INPUT: C_1, C_2 (two circuits, described by DAGs with gate labels on the vertices);

QUESTION: $f_{C_1} = f_{C_2}$?

In order to consider the complexity of problems about circuits we need to define the *size* of an acyclic boolean circuit C, denoted by $|C|$, and simply called **circuit size**; it is defined as follows: If C has k_1 gates of type NOT or FORK, k_2 gates of type AND or OR, and n output variables, then the size of C is defined to be $|C| = k_1 + 2 \cdot k_2 + n$. Equivalently, $|C|$ *is the number of edges* (or *wires*) between gates, or from an input to a gate, or from a gate to an output (for that reason, gates with two input variables are counted twice).

Remarks concerning circuit definitions: Acyclic circuits and their sizes are defined in a variety of ways in the literature [23,36,39,44,22,26,20,27]; however, all these definitions lead to sizes that are *polynomially equivalent* (i.e., each one is polynomially bounded in terms of every other one). In the theory of NP- or coNP-completeness, polynomial differences are not significant.

(1) In the literature, the circuit size is usually defined as the number of vertices. Since we do not use isolated vertices in a circuit, we have $n_V \leq n_E \leq n_V^2$ (where n_V and n_E denote the number of vertices and edges). So n_V and n_E are polynomially equivalent.

(2) In the literature the input and output variables are usually not called vertices, but in that case they are nevertheless counted among the vertices in the definition of circuit size.

(3) When a circuit is described by a bitstring s_C, the length satisfies $n_E \leq |s_C| \leq c\, n_E \log_2 n_V$, for some constant $c \geq 1$. Typically, such a description of C lists all the edges, where each edge is given as a pair of strings (the names of two vertices, each vertex name having length $\leq 1 + \log_2 n_V$). An additional list is given that associates a gate or an input variable or an output variable with each vertex. An input variable x_i is described by a code word (for x) and the binary representation of i; the output variables

y_j are described similarly. In any case, $|s_C|$ and n_E are polynomially equivalent.
(4) In the literature, the FORK-gate is usually not used explicitly; instead, the AND-, OR-, and NOT- gates, as well as the input variables, are allowed to have a *fan-out*. However, even in that case, every wire goes to a gate or an output variable, so the total of all be fan-outs is $\leq n_V^2$. A gate with fan-out k can be replaced by a gate with fan-out 1 and $k-1$ FORK-gates. This leads to a circuit with gates that have fan-out 1, except for FORK-gates with fan-out 2; the size increase is polynomially bounded.
(5) In the literature, AND and OR-gates are allowed to have a *fan-in* ≥ 2. But every fan-in wire comes from a gate of an input variable, so the total of all be fan-ins is $\leq n_V^2$. An OR-gate with fan-in k can be replaced by $k-1$ OR-gates with fan-in 2 (and similarly for AND). This leads to a circuit with gates that have fan-in ≤ 2; the size increase is polynomially bounded.

Remarks on complexity: The circuit equivalence problem is a well-known problem that is coNP-complete. It is fairly straightforward to prove that the problem is in coNP. Moreover, the tautology problem for boolean formulas (which is a classical coNP-complete problem) is a special case of the circuit equivalence problem (and is reduced to the circuit equivalence problem by converting a boolean formula into a circuit and asking whether a given circuit is equivalent to a circuit for the constant-1 function). See [27, Introduction] for comments on the circuit equivalence problem, see [36] for a circuit-based proof of NP-completeness of the satisfiability problem for boolean formulas, and see [23,22] for general information.

The following well-known fact implies that every $\tau_{i,j}$ can be expressed as a composition of elements of $\tau = \{\tau_{k,k+1} : k \geq 1\}$; the expression has linear length in terms of j.

Lemma 4.9. *As elements of V the transpositions satisfy*

$$\tau_{i,j} \;=\; \tau_{i,i+1}\,\tau_{i+1,i+2}\,\cdots\,\tau_{j-2,j-1}\,\tau_{j-1,j}\,\tau_{j-2,j-1}\,\cdots\,\tau_{i+1,i+2}\,\tau_{i,i+1},\ \text{if } 1 \leq i < j.$$

The word length of $\tau_{i,j}$ over τ is therefore $\leq 2(j-i)-1$. □

We want to represent the circuit gates NOT, OR, AND, and FORK, by elements of V. For this, the main problem is that the input-output function of a circuit is not necessarily a permutation. Therefore we introduce the following notion of "simulation" of a circuit C by a Thompson group element Φ_C and by a word w_C over $\Gamma_V \cup \tau$ (Definition 4.10 and Theorem 4.12 below). See the discussion in [6] for additional motivation of our definition of simulation.

Definition 4.10 *(Simulation).* Let $f : \{0,1\}^m \to \{0,1\}^n$ be a total function. An element $\Phi_f \in V$ *simulates* f iff for all $x \in \{0,1\}^m$: $\Phi_f(0\,x) \;=\; 0\;f(x)\;x$.
When Φ_f is represented by a *word* $w_f \in (\Gamma_V \cup \tau)^*$ we say that w_f simulates f.

According to this definition, f is faithfully described by the action of Φ_f on $0\,\{0,1\}^*$; but there are no constraints on the values of Φ_f for input strings in $1\,\{0,1\}^*$. Since

Φ_f is an element of V it is a bijection between finite maximal prefix codes, whereas f need not be injective nor surjective. So there has to be a big difference between Φ_f and f somewhere. In subsections 4.3 and 4.4 we show that, nevertheless, the equivalence problem of circuits can be reduced to the word problem of V over $\Gamma_V \cup \tau$. In the rest of this subsection we construct Φ_f.

The next Lemma follows immediately from the definition of simulation.

Lemma 4.11. *Let f and g be any boolean functions with the same number of input variables and the same number of output variables. If f and g are simulated by Φ_f, respectively Φ_g, then we have*

$$f = g \text{ iff } (\Phi_f)|_{0\{0,1\}^*} = (\Phi_g)|_{0\{0,1\}^*} . \quad \square$$

We choose the following elements of V to describe the gates NOT, OR, AND, and FORK:

$$\varphi_\neg = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

$$\varphi_\vee = \begin{bmatrix} 0x_1x_2 & 1x_1x_2 \\ (x_1 \vee x_2)\, x_1x_2 & (\overline{x_1 \vee x_2})\; x_1x_2 \end{bmatrix},$$

$$\varphi_\wedge = \begin{bmatrix} 0x_1x_2 & 1x_1x_2 \\ (x_1 \wedge x_2)\, x_1x_2 & (\overline{x_1 \wedge x_2})\; x_1x_2 \end{bmatrix},$$

where x_1 and x_2 range over $\{0,1\}$. Hence, $\mathrm{domC}(\varphi_\neg) = \mathrm{imC}(\varphi_\neg) = \{0,1\}$, and $\mathrm{domC}(\varphi_\vee) = \mathrm{imC}(\varphi_\vee) = \mathrm{domC}(\varphi_\wedge) = \mathrm{imC}(\varphi_\wedge) = \{0,1\}^3$. In order to represent the FORK function we first define

$$\varphi_{0\mathrm{f}} = \begin{bmatrix} 0 & 10 & 11 \\ 00 & 01 & 1 \end{bmatrix};$$

so, $\mathrm{domC}(\varphi_{0\mathrm{f}}) = \{0, 10, 11\}$, and $\mathrm{imC}(\varphi_{0\mathrm{f}}) = \{00, 01, 1\}$. Then FORK is simulated by

$\varphi_{\mathrm{f}} = \tau_{1,2} \circ \varphi_\vee \circ \varphi_{0\mathrm{f}}$.

Indeed, for all $x_1 \in \{0,1\}$: $\tau_{1,2} \circ \varphi_\vee \circ \varphi_{0\mathrm{f}}(0x_1) = 0x_1x_1$.

For every acyclic boolean circuit C we want to find a word $w_C \in (\Gamma_V \cup \tau)^*$ that simulates C; and we want the map $C \mapsto w_C$ to be polynomial-time computable (in terms of $|C|$).

A standard property of DAGs is that every vertex has a *level* (or "layer") corresponding to its "depth" in the DAG. The source vertices have level 0. A gate or an output variable has level 1 iff only input variables of the circuit feed into it. A gate or an output variable has level ℓ iff it receives input from levels $< \ell$ only, and at least one of its inputs comes from level $\ell - 1$. Equivalently, the level of a vertex v is the length of a longest path from a source to v. The maximum level of any sink vertex is called the *depth* of the DAG.

The following theorem is a simplification of [6, Thm. 3.5]. For a word $w \in (\Gamma_V \cup \tau)^*$ we use the *size*, denoted by $\|w\|$, as defined in Definition 4.7.

Theorem 4.12 *(Existence of simulation). There is an injective function $C \mapsto w_C$ from the set of acyclic boolean circuits to the set of words over $\Gamma_V \cup \tau$ with the following properties:*

(1) *w_C simulates the input-output function f_C of C;*
(2) *the size of w_C satisfies $\|w_C\| < c\,|C|^6$ (for some constant $c > 0$);*
(3) *w_C is computable from C in polynomial time, in terms of $|C|$.*

Proof. Item (1) refers to simulation as in Definition 4.10. In the proof we assume that $\varphi_\neg$, $\varphi_\vee$, $\varphi_\wedge$, φ_{f}, $\varphi_{0\mathrm{f}}$, and $\tau_{1,2}$, belong to Γ_V. (If this were not the case, we could express them by fixed words over Γ_V.)

We can assume that our acyclic circuits are *strictly layered*, i.e., a gate or an output variable at level ℓ only receives inputs from level $\ell - 1$. Hence, all the output variables of the circuit are at the same level L, where L is the depth of the circuit. If the layering of a circuit C is not strict, we can insert *identity gates* to obtain strictness. An identity gate has one input variable and one output variable, connected by a wire; the two variables carry the same boolean value. We will count these identity gates as gates in the evaluation of circuit size. In order to make a circuit C strictly layered, fewer than $|C|^2$ identity gates need to be introduced. (Indeed, for each gate we add fewer than $|C|$ identity gates above it; so, in total we add fewer than $|C|^2$ identity gates.)

An acyclic circuit C has input variables $x_1, \ldots, x_m$, output variables $y_1, \ldots, y_n$, and *internal variables* which correspond to the boolean values carried by internal wires (between gates or between a gate and an input or an output port). The internal variables at level ℓ (for $0 \le \ell \le L$) are denoted by y^ℓ_1, y^ℓ_2, …, $y^\ell_{n_\ell}$. When $\ell = L$ (output level) we have $n_L = n$ and $y^L_i = y_i$; when $\ell = 0$ (input level) we have $n_0 = m$ and $y^0_i = x_i$.

For every level ℓ (with $1 \le \ell \le L$) there is a circuit C_ℓ, called the *slice of* C at level ℓ: The input variables of the slice C_ℓ are $y^{\ell-1}_1, \ldots, y^{\ell-1}_{n_{\ell-1}}$; the output variables are $y^\ell_1, \ldots, y^\ell_{n_\ell}$; the gates of C_ℓ are the gates of C at level ℓ; we use the fact that C is strictly layered. In addition to gates, a slice C_ℓ also contains wire-swappings of its inputs, i.e., a bit-position permutation is applied to the $n_{\ell-1}$ input variables. Every permutation of $n_{\ell-1}$ wires can be written as the composite of $\le n_{\ell-1}$ ($< |C_\ell|$) transpositions. And each $\tau_{i,j}$ has word length $\le 2(j-i)-1$ over τ (by Lemma 4.9), hence it has size $\|\tau_{i,j}\| < |C_\ell|^2$. Thus the input-wire permutation of a slice C_ℓ has size $< |C_\ell|^3$. Moreover, every $\tau_{i,j}$ belongs to V, so it does not need any simulation.

We use the notation $Y^\ell = y^\ell_1 y^\ell_2 \ldots y^\ell_{n_\ell}$ (i.e., the concatenation of the variables y^ℓ_i, for $i = 1, \ldots, n_\ell$, and $\ell = 0, \ldots, L$).

Simulation of one slice

In order to construct w_C we first consider the special case where the circuit C consists of just one slice, hence C has depth 2 (the gates of the slice have depth 1, the output variables have depth 2). Identity gates are allowed. We number the gates of C from left to right.

For $k \ge 0$, let K consist of the first k gates of a slice; so, K is a one-slice circuit that has k gates. When $k = 0$, K is empty, w_K is the empty string, and its input-output

function is the identity function ($\in V$). Inductively, let C be a slice obtained from K by adding one gate (AND, OR, NOT, identity, or FORK) on the right of K (with number $k+1$). Inductively we assume that K satisfies the Theorem and that w_K has been constructed. Let $x_1, \ldots, x_m$ be the input variables and let $y_1, \ldots, y_n$ be the output variables of K. We now construct w_C from w_K and the gate being added.

CASE 1: Suppose the slice C is obtained from K by adding, on the right of K, an identity gate or a NOT gate, with new input variable x_{m+1} and new output variable y_{n+1}. If a NOT gate is added, the input-output function of C is $f_C(x_1, \ldots, x_m, x_{m+1}) = (y_1, \ldots, y_n, \overline{x_{m+1}})$, where $f_K(x_1, \ldots, x_m) = (y_1, \ldots, y_n)$. The boolean function f_C is to be simulated by a Thompson group element Φ_{f_C} such that

$$\Phi_{f_C}(0\, x_1 \ldots x_m, x_{m+1}) \;=\; 0\, y_1 \ldots y_n\ \overline{x_{m+1}}\ x_1 \ldots x_m x_{m+1}.$$

We have $w_K \in (\Gamma_V \cup \tau)^*$, where $\Phi_{f_K} \in V$ is the simulation of f_K, which exists by induction. We find w_C as follows:

$$\begin{aligned}
&0\, x_1 \ldots x_m\ x_{m+1} \overset{\Phi_{f_K}}{\longmapsto} 0 y_1 y_2 \ldots y_n\ x_1 \ldots x_m\ x_{m+1} \\
&\overset{\tau_{2,n+m+2}}{\longmapsto} 0\ x_{m+1}\ y_2 \ldots y_n\ x_1 \ldots x_m\ y_1 \overset{\varphi_{\mathrm{f}}}{\longmapsto} 0\ x_{m+1}\ x_{m+1}\ y_2 \ldots y_n\ x_1 \ldots x_m\ y_1 \\
&\overset{\tau_{1,2}}{\longmapsto} x_{m+1}\ 0\ x_{m+1}\ y_2 \ldots y_n\ x_1 \ldots x_m\ y_1 \\
&\overset{\varphi_\neg}{\longmapsto} \overline{x_{m+1}}\ 0\ x_{m+1}\ y_2 \ldots y_n\ x_1 \ldots x_m\ y_1 \overset{\tau_{3,n+m+3}}{\longmapsto} \overset{\tau_{1,2}}{\longmapsto} 0\ \overline{x_{m+1}}\ y_1 y_2 \ldots y_n\ x_1 \ldots x_m x_{m+1};
\end{aligned}$$

applying $\tau_{n+1,n+2}\ \tau_{n,n+1}\ \ldots\ \tau_{3,4}\ \tau_{2,3}(.)$ then yields

$$0\ y_1 \ldots y_n\ \overline{x_{m+1}}\ x_1 \ldots x_m x_{m+1}.$$

So, $w_C = \tau_{n+1,n+2}\ \tau_{n,n+1}\ \ldots\ \tau_{3,4}\ \tau_{2,3}\ \tau_{1,2}\ \tau_{3,n+m+3}\ \varphi_\neg\ \tau_{1,2}\ \varphi_{\mathrm{f}}\ \tau_{2,n+m+2}\ w_K$.
The case where, instead of a NOT gate, an identity gate is added is similar (except that we simply omit $\varphi_\neg$). By Lemma 4.9, we can express $\tau_{2,n+m+2}$ and $\tau_{3,n+m+3}$ over $\tau = \{\tau_{k,k+1} : k \geq 1\}$. Then the size of w_C is

$$\begin{aligned}
\|w_C\| &\leq \|w_K\| + \|\tau_{2,n+m+2}\| + \|\tau_{3,n+m+3}\| + 4 + \textstyle\sum_{k=2}^{n+1} \|\tau_{k,k+1}\| \\
&\leq \|w_K\| + c\,(n+m)^2 + c, \text{ for some constant } c > 1.
\end{aligned}$$

CASE 2: Suppose our slice C is obtained by adding an AND gate or an OR gate to K on the right, with new output variable y_{n+1} and new input variables x_{m+1}, x_{m+2}. We only analyze the OR case, the AND case being almost the same. The input-output function of C is

$$f_C(x_1, \ldots, x_m, x_{m+1}, x_{m+2}) \;=\; (y_1, \ldots, y_n,\ x_{m+1} \vee x_{m+2}),$$

where $f_K(x_1, \ldots, x_m) = (y_1, \ldots, y_n)$. The function f_C is to be simulated by a Thompson group element Φ_{f_C} such that

$$\Phi_{f_C}(0\, x_1 \ldots x_m\ x_{m+1} x_{m+2}) \;=\; 0\ y_1 \ldots y_n\ (x_{m+1} \vee x_{m+2})\ x_1 \ldots x_m x_{m+1} x_{m+2}$$

Let $w_K \in (\Gamma_2 \cup \tau)^*$ be such that $\Phi_{f_K} \in V$ simulates f_K. Then we construct w_C as follows:

$$\begin{aligned}
&0\, x_1 \ldots x_m\ x_{m+1} x_{m+2} \overset{\Phi_{f_K}}{\longmapsto} 0\, y_1 \ldots y_n\ x_1 \ldots x_m\, x_{m+1}\, x_{m+2} \\
&\overset{\varphi_{0\mathrm{f}}}{\longmapsto} 00\ y_1 \ldots y_n\ x_1 \ldots x_m\, x_{m+1}\, x_{m+2}
\end{aligned}$$

$$\overset{\tau_{2,n+m+3}}{\longmapsto}\ \overset{\tau_{3,n+m+4}}{\longmapsto}\ 0\,x_{m+1}x_{m+2}\ y_2\dots y_n\ x_1\dots x_m\ 0y_1$$
$$\overset{\varphi_\vee}{\longmapsto}\ (x_{m+1}\vee x_{m+2})\ x_{m+1}x_{m+2}\ y_2\dots y_n\ x_1\dots x_m\ 0y_1$$
$$\overset{\tau_{2,n+m+3}}{\longmapsto}\ \overset{\tau_{3,n+m+4}}{\longmapsto}\ (x_{m+1}\vee x_{m+2})\ 0\ y_1y_2\dots y_n\ x_1\dots x_m\,x_{m+1}\,x_{m+2};$$

applying $\tau_{n+1,n+2}\ \dots\ \tau_{2,3}\,\tau_{1,2}$ then yields

$$0\ y_1y_2\dots y_n\ (x_{m+1}\vee x_{m+2})\ x_1\dots x_m\ x_{m+1}x_{m+2}.$$

Thus C is simulated by the word

$$w_C = \tau_{n+1,n+2}\ \dots\ \tau_{2,3}\,\tau_{1,2}\ \tau_{3,n+m+4}\,\tau_{2,n+m+3}\ \varphi_\vee\ \tau_{3,n+m+4}\ \tau_{2,n+m+3}\ \varphi_{0\mathrm{f}}\ w_K$$

of size

$$\begin{aligned}\|w_C\| &\le \|w_K\| + 2\,\|\tau_{2,n+m+3}\| + 2\,\|\tau_{2,n+m+4}\| + 2 + \textstyle\sum_{k=1}^{n+1}\|\tau_{k,k+1}\|\\ &\le \|w_K\| + c\,(n+m)^2 + c, \text{ for some constant } c>1.\end{aligned}$$

Case 3: Suppose our slice C is obtained by adding a FORK gate on the right of K, with a new input variable x_{m+1} and two new output variables y_{n+1} and y_{n+2}. The input-output function of C is

$$f_C(x_1,\dots,x_m,x_{m+1}) \ =\ (y_1,\dots,y_n,x_{m+1},x_{m+1}),$$

where $f_K(x_1,\dots,x_m)=(y_1,\dots,y_n)$. The boolean function f_C is to be simulated by a Thompson group element Φ_f such that

$$\Phi_f(0\,x_1\dots x_mx_{m+1}) \ =\ 0\ y_1\dots y_n\ x_{m+1}x_{m+1}\ x_1\dots x_mx_{m+1}.$$

Let $w_K\in(\Gamma_V\cup\tau)^*$ and $\Phi_{f_K}\in V$ be the simulation of f_K, which exists by induction. Then

$$0\,x_1\dots x_m\ x_{m+1}\overset{\Phi_{f_K}}{\longmapsto}0\ y_1y_2\dots y_n\ x_1\dots x_m\ x_{m+1}$$
$$\overset{\tau_{2,n+m+2}}{\longmapsto}0\ x_{m+1}\ y_2\dots y_n\ x_1\dots x_m\ y_1\overset{\varphi_{\mathrm{f}}}{\longmapsto}\overset{\varphi_{\mathrm{f}}}{\longmapsto}0\,x_{m+1}x_{m+1}x_{m+1}\ y_2\dots y_n\ x_1\dots x_m\ y_1$$
$$\overset{\tau_{4,n+m+4}}{\longmapsto}0\,x_{m+1}\,x_{m+1}\ y_1\dots y_n\ x_1\dots x_m\ x_{m+1};$$

applying $\tau_{n+2,n+3}\ \dots\ \tau_{3,4}$ and then $\tau_{n+1,n+2}\ \dots\ \tau_{2,3}$ yields

$$0\ y_1y_2\dots y_n\ x_{m+1}x_{m+1}\ x_1\dots x_mx_{m+1}.$$

This simulates f_C by a word

$$w_C = \tau_{n+1,n+2}\ \dots\ \tau_{2,3}\ \tau_{n+2,n+3}\ \dots\ \tau_{3,4}\ \tau_{4,n+m+4}\ \varphi_{\mathrm{f}}\ \varphi_{\mathrm{f}}\ \tau_{2,n+m+2}\ w_K$$

of size

$$\begin{aligned}\|w_C\| &\le \|w_K\| + 2 + \|\tau_{2,n+m+2}\| + \|\tau_{4,n+m+4}\| + \textstyle\sum_{k=3}^{n+2}\|\tau_{k,k+1}\| + \sum_{k=2}^{n+1}\|\tau_{k,k+1}\|\\ &\le \|w_K\| + c\,(m+n)^2 + c, \text{ for some constant } c>1.\end{aligned}$$

In all three cases the slice C is simulated by a word $w_C\in(\Gamma_V\cup\tau)^*$ of size

$$\|w_C\| \le \|w_K\| + c\,(m+n)^2 + c.$$

Let S now be any slice, and let n_i be the number of interior vertices of S (i.e., the vertices labeled by gates). Then if w_S is constructed by adding n_i ($<|S|$) gates to slices (starting with K being the empty slice, and ending with K being the desired slice S), the size of w_S is

$$\|w_S\| \ \le\ n_i\,(c\,(m+n)^2+c) \ \le\ c_0\,|S|^3,$$

for some constant $c_0 > 1$ (that does not depend on S).

Moreover, as we saw when we introduced the notion of slice, in all three cases a bit-position permutation of the input wires of the slice S is attached at the beginning of w_S. This permutation belongs to V and has size $< |S|^3$.

The above construction of each word w_S from S is a polynomial-time algorithm (in terms of $|S|$).

Simulation of a multi-slice circuit

Assume that C is a circuit of depth $L > 2$; the depth is the number of slices. In order to define w_C we use the fact that we have already defined the word w_{C_ℓ} that simulates the slice C_ℓ of C (for every ℓ, $1 \le \ell \le L$). Each word w_{C_ℓ} has all the properties claimed in Theorem 4.12; in particular, w_{C_ℓ} represents the function

$$\Phi_{C_\ell} : \ 0\ Y^{\ell-1} \longmapsto 0\ Y^\ell\ Y^{\ell-1}.$$

Hence, since Φ_{C_ℓ} is a right ideal isomorphism, we also have

$$0\ Y^{\ell-1}\ Y^{\ell-2}\ \dots\ Y^1\ x_1 \dots x_m \overset{\Phi_{C_\ell}}{\longmapsto} 0\ Y^\ell\ Y^{\ell-1}\ Y^{\ell-2}\ \dots\ Y^1\ x_1 \dots x_m.$$

Therefore, $w_{C_L}\, w_{C_{L-1}}\ \dots\ w_{C_\ell}\ \dots\ w_{C_1}$ represents the function

$$\Phi_{C_L C_{L-1} \dots C_1} \colon 0\, x_1 \dots x_m \longmapsto 0\ y_1 \dots y_n\, Y^{L-1} \dots Y^\ell \dots Y^2\, Y^1\, x_1 \dots x_m \ (=_{\text{def}}\ Z),$$

where $y_1 \dots y_n = Y^L$ is the output of C, and $x_1 \dots x_m$ is the input of C.

The length of the word Z ($\in \{0,1\}^*$) is $|Z| \le 1 + |C|$. Indeed, the total number of variables in the circuit (i.e., $n_L + \dots + n_1 + m$) is equal to the total number of wires (i.e., $|C|$); the "+1" comes from the leading bit 0.

Let $\sigma_{i,j} = \tau_{j-1,j}\, \tau_{j-2,j-1}\ \dots\ \tau_{i+1,i+2}\, \tau_{i,i+1}(.)$ (for $1 \le i < j$). Then $\pi_1 = (\sigma_{1,|Z|})^n$ transforms the word Z into

$$0\ Y^{L-1}\ \dots\ Y^\ell\ \dots\ Y^2\ Y^1\ x_1 \dots x_m\ y_1 \dots y_n.$$

Next (and this is a fundamental and crucial idea from *reversible computing*, see e.g., [3,2,21]), to the latter string we apply

$$(w_{C_{L-1}}\ \dots\ w_{C_\ell}\ \dots\ w_{C_2}\, w_{C_1})^{-1}$$

in order to clear away intermediate outputs of all the internal slices. This yields

$$0\ x_1 \dots x_m\ y_1 \dots y_n.$$

Finally, applying the permutation $\pi_2 = (\sigma_{1,n+m})^m$ produces the desired final output

$$0\ y_1 \dots y_n\ x_1 \dots x_m.$$

Therefore we define $w_C \in (\Gamma_V \cup \tau)^*$) by

$$w_C = \pi_2\ (w_{C_{L-1}}\ \dots\ w_{C_1})^{-1}\ \pi_1\ \ w_{C_L}\ w_{C_{L-1}}\ \dots\ w_{C_1}.$$

The word length of π_1 over τ is less than $n\ |Z|$. Since all subscripts in $\sigma_{1,|Z|}$ are $\le |Z|$, the size of π_1 is $\|\pi_1\| < |Z|\ n\ |Z| \le (|C|+1)^3$. Since $m+n \le |C|$, the size of π_2 is also less than $(|C|+1)^3$.

For the size of w_C we have

$$\|w_C\| \leq \|\pi_2\| + \|\pi_1\| + \|w_{C_L}\| + 2\sum_{\ell=1}^{L-1} \|w_{C_\ell}\|.$$

We saw that $\|w_{C_\ell}\| \leq c_0\,|C_\ell|^3$ (for $1 \leq \ell \leq L$); and $\sum_{\ell=1}^{L} |C_\ell| = |C|$ implies $\sum_{\ell=1}^{L} |C_\ell|^3 \leq |C|^3$. Thus $\|w_C\| \leq c \cdot |C|^3$, for some positive constant c.

Since $|C|$ was at most squared in order to obtain strict layering, the above bound becomes

$$\|w_C\| \;\leq\; c\,|C|^6,$$

in terms of the original (not necessarily strictly layered) circuit C.

The word w_C can be written down in linear time, based on the words w_{C_ℓ} $(1 \leq \ell \leq L)$, and we saw that each w_{C_ℓ} can be computed in polynomial time from C_ℓ. □

4.3. Reduction to a generalized word problem of V (over an infinite generating set)

We first extend the classical concepts of stabilizer and fixator to the case of partial injections.

Definition 4.13. A function g *partially stabilizes* a set $S \subseteq \{0,1\}^*$ iff $g(S) \cup g^{-1}(S) \subseteq S$. For a subgroup $G \subseteq V$, the *partial stabilizer* of S (in G) is

$$\mathrm{pStab}_G(S) \;=\; \{g \in G :\; g(S) \,\cup\, g^{-1}(S) \,\subseteq\, S\}.$$

A function g *partially fixes* a set S iff $g(x) = x$ for every $x \in S \cap \mathrm{Dom}(g) \cap \mathrm{Im}(g)$. This is also called partial pointwise stabilization. For a subgroup $G \subseteq V$, the *partial fixator* of S (in G) is

$$\mathrm{pFix}_G(S) \;=\; \{g \in G\colon (\forall x \in S \cap \mathrm{Dom}(g) \cap \mathrm{Im}(g))\; [\, g(x) = x \,]\,\}.$$

We will only use partial stabilizers and fixators for sets S that are right ideals; then $\mathrm{pStab}_G(S)$ and $\mathrm{pFix}_G(S)$ are groups [6, Lemma 4.1]. When $S = P\{0,1\}^*$ is a right ideal, where P is a prefix code, we will abbreviate $\mathrm{pFix}_G(P\,\{0,1\}^*)$ and $\mathrm{pStab}_G(P\,\{0,1\}^*)$ by $\mathrm{pFix}_G(P)$, respectively $\mathrm{pStab}_G(P)$. In particular, we abbreviate $\mathrm{pFix}_V(0\,\{0,1\}^*)$ to $\mathrm{pFix}_V(0)$.

Lemma 4.14. *We have:* $\mathrm{pFix}_V(0) \;\subset\; \mathrm{pStab}_V(0\,\{0,1\}^*) \;\cap\; \mathrm{pStab}_V(1\,\{0,1\}^*)$.

Proof. Obviously, $\mathrm{pFix}_V(0) \subset \mathrm{pStab}_V(0\,\{0,1\}^*)$. Moreover, if we had $g(1x) = 0y$ for any $g \in \mathrm{pFix}_V(0)$ and $x, y \in \{0,1\}^*$, then $0y = g^{-1}(0y) = g^{-1}g(1x) = 1x$; the first equality holds since $g^{-1} \in \mathrm{pFix}_V(0)$. But $0y = 1x$ is false since a string does not start with both 0 and 1. □

The following is little more than a reformulation of the definition of simulation and Lemma 4.11.

Lemma 4.15. *Let f and g be any boolean functions such that f and g have the same number of input variables, and f and g have the same number of output variables. Suppose f and g are simulated by Φ_f, respectively Φ_g ($\Phi_f, \Phi_g \in V$). Then,*

$$f = g \text{ iff } \Phi_f^{-1}\,\Phi_g \in \mathrm{pFix}_V(0).$$

Proof. Let $\{0,1\}^m$ be the common domain of f and g. Then by Lemma 4.11, $f = g$ iff for all $x \in \{0,1\}^m$: $\Phi_f(0x) = \Phi_g(0x)$. Then for all $x \in \{0,1\}^m$: $0x = \Phi_f^{-1}\,\Phi_g(0x) = \Phi_g^{-1}\,\Phi_f(0x)$ (and $\Phi_g^{-1}\,\Phi_f = (\Phi_f^{-1}\,\Phi_g)^{-1}$). Hence, $f = g$ iff $\Phi_f^{-1}\,\Phi_g \in \mathrm{pFix}_V(0)$. □

Theorem 4.12 and Lemma 4.15 give a polynomial-time one-one reduction from the circuit equivalence problem to the generalized word problem of $\mathrm{pFix}_V(0)$ in V, where the elements of V written over $\Gamma_V \cup \tau$. Since the circuit equivalence problem is coNP-complete, it follows that this generalized word problem is coNP-hard. Hence we have:

Corollary 4.16 *(coNP-hard generalized word problem). The generalized word problem of $\mathrm{pFix}_V(0)$ in V over $\Gamma_V \cup \tau$ is coNP-hard.* □

4.4. Reduction to the word problem of V

We will give a linear-time 2-ary conjunctive reduction from the generalized word problem of $\mathrm{pFix}_V(0)$ to the word problem of V over the infinite generating set $\Gamma_V \cup \tau$. This reduction is based on a "commutation test", that was studied in greater generality in [6, Section 5]; here we just use V, based on an alphabet of size 2, which makes everything simpler.

We first need a few lemmas. Recall the notation $u \parallel_{\mathrm{pref}} v$ (u and v are prefix-comparable) and its negation $\nparallel_{\mathrm{pref}}$. For $x \in \{0,1\}^*$ and $L \subseteq \{0,1\}^*$, we define $x^{-1}L = \{v \in \{0,1\}^* : xv \in L\}$.

Lemma 4.17. *If $g \notin \mathrm{pFix}_V(0)$ but $g \in \mathrm{pStab}_V(0)$, then there exists $0x \in \mathrm{domC}(g)$ such that*

$$0x \nparallel_{\mathrm{pref}} g(0x).$$

Hence, $0xu \nparallel_{\mathrm{pref}} g(0xu)$ $\big(= g(0x)\,u\big)$, for all $u \in \{0,1\}^*$.

Proof. Lemma 4.17 is a special case of [6, Lemma 9.6], with a simpler proof. (Note that in [6] the notation $\leq_{\mathrm{pref}}$ for the prefix order was reversed; here, "$p \leq_{\mathrm{pref}} w$" always means p is a prefix of w.)

We prove the contrapositive, i.e., if for all $0x \in \mathrm{domC}(g)$ we have $0x \parallel_{\mathrm{pref}} g(0x)$, then $g \in \mathrm{pFix}_V(0)$.

Case 1: $0x <_{\mathrm{pref}} g(0x)$.

Then $g(0x) = 0x\,v$, for some $v \in \{0,1\}^+$, so $v \in (0x)^{-1}\mathrm{imC}(g)$. Now, $(0x)^{-1}(\mathrm{imC}(g))$ is a maximal finite prefix code (by [6, Lemma 9.4]), which contains the non-empty string v. Hence $(0x)^{-1}\mathrm{imC}(g)$ contains at least one other non-empty string (by [6, Lemma 9.5]), i.e., $\mathrm{imC}(g)$ contains $0xw$ ($\neq 0xv$), for some $w \in \{0,1\}^+$. Hence (since g^{-1} stabilizes $0\{0,1\}^*$), there exists $0x' \in \mathrm{domC}(g)$ such that $0x' \neq 0x$, and $g(0x') \in \mathrm{imC}(g)$ and $g(0x') = 0x\,w >_{\mathrm{pref}} 0x$. Since $\mathrm{imC}(g)$ is a prefix code, $g(0x) \nparallel_{\mathrm{pref}} g(0x')$.

By the (contrapositive) assumption, $0x' \parallel_{\mathrm{pref}} g(0x')$. Hence there are two possibilities:

(1) $0x' \geq_{\mathrm{pref}} g(0x')$: Then $0x' \geq_{\mathrm{pref}} g(0x') >_{\mathrm{pref}} 0x$. So $0x' >_{\mathrm{pref}} 0x$, which contradicts the fact that $\mathrm{domC}(g)$ is a prefix code.

(2) $0x' <_{\mathrm{pref}} g(0x')$: Then $0x' <_{\mathrm{pref}} g(0x') = 0x'\,z$, for some $z \in \{0,1\}^+$; and we saw that also $g(0x') = 0x\,w$. This implies that $0x \parallel_{\mathrm{pref}} 0x'$. Again, this contradicts that $\mathrm{domC}(g)$ is a prefix code.

Thus, case 1 is impossible.

Case 2: $0x >_{\mathrm{pref}} g(0x)$.

Then $0x = g(0x)\,u$, for some $u \in \{0,1\}^+$, hence $u \in (g(0x))^{-1}\mathrm{domC}(g)$. Now $(g(0x))^{-1}\mathrm{domC}(g)$ is a finite maximal prefix code, containing the non-empty string u, hence it contains some other non-empty string. So there exists $0x'$ ($\neq 0x$) with $0x' \in \mathrm{domC}(g) \cap g(0x)\,\{0,1\}^+$.

By the (contrapositive) assumption, $0x' \parallel_{\mathrm{pref}} g(0x')$. Again, we have two possibilities:

(1) $0x' \leq_{\mathrm{pref}} g(0x')$: Then $g(0x') \geq_{\mathrm{pref}} 0x'$, and $0x' >_{\mathrm{pref}} g(0x)$ (since $0x' \in g(0x)\,\{0,1\}^+$). Thus, $g(0x') >_{\mathrm{pref}} g(0x)$, which contradicts the fact that $\mathrm{imC}(g)$ is a prefix code.

(2) $0x' >_{\mathrm{pref}} g(0x')$: Then $0x' = g(0x')\,z$, for some $z \in \{0,1\}^+$; and $0x' = g(0x)\,w$, for some $w \in \{0,1\}^+$ (since $0x' \in g(0x)\,\{0,1\}^+$). Thus, $0x' = g(0x')\,z = g(0x)\,w$, which implies $g(0x') \parallel_{\mathrm{pref}} g(0x)$. Again, this contradicts the fact that $\mathrm{imC}(g)$ is a prefix code.

We conclude that case 2 is impossible.

Now, having ruled out cases 1 and 2, the only remaining possibility is that $0x = g(0x)$, for all $0x \in \mathrm{domC}(g)$. This means that $g \in \mathrm{pFix}_V(0)$. □

Lemma 4.18. *For every $0x, 0y \in 0\,\{0,1\}^*$ such that $0x \nparallel_{\mathrm{pref}} 0y$, there exists $f_0 \in \mathrm{pFix}_V(1)$ and $u \in \{0,1\}^*$ such that*

$$f_0(0xu) = 0xu \text{ and } f_0(0yu) \neq 0yu.$$

Proof. This Lemma is a simplification of [6, Prop. 9.14(1)], and we adapt that proof.

Let $0x, 0y \in 0\,\{0,1\}^*$ be two prefix-incomparable strings, and let $a, b \in \{0,1\}$ with $a \neq b$. Then $0x, 0ya, 0yb$ are prefix-incomparable two-by-two (as is easy to check). We now use [6, Lemma 9.7] to construct a finite maximal prefix code $Q \cup \{0x, 0ya, 0yb, 1\}$, with $Q \subset 0\,\{0,1\}^*$. We define $f_0 \in V$ by

$f_0(0ya) = 0yb$, $f_0(0yb) = 0ya$, $f_0(0x) = 0x$, and

f_0 is the identity on $Q \cup \{1\}$.

So, $Q \cup \{0x, 0ya, 0yb, 1\}$ is the domain code and image code of f_0. Then $f_0 \in \mathrm{pFix}_V(1)$, $f_0(0ya) \neq 0ya$, and $f_0(0xa) = 0xa$ (since $f_0(0x) = 0x$). So here, a plays the role of u. □

Lemma 4.19 *(Commutation test). For all $g \in V$ we have:*

$$g \in \mathrm{pFix}_V(0) \text{ iff } \big(\forall f \in \mathrm{pFix}_V(1)\big)\,[\,fg = gf\,].$$

In words: An element $g \in V$ belongs to the subgroup $\mathrm{pFix}_V(0)$ iff g commutes with all the elements of the subgroup $\mathrm{pFix}_V(1)$.

Proof. [⇐] Suppose $fg = gf$, for all $f \in \mathrm{pFix}_V(1)$, and hence also $g^{-1}f = fg^{-1}$.

(1) We first prove that $g \in \mathrm{pStab}_V(0)$.

If $g(0x) = 1y$ for some $x, y \in \{0,1\}^*$, then $fg(0x) = f(1y) = 1y$ for all $f \in \mathrm{pFix}_V(1)$. And $1y = fg(0x) = gf(0x)$. Hence, $g(0x) = 1y = g(f(0x))$, hence by injectiveness, $0x = f(0x)$ for all $f \in \mathrm{pFix}_V(1)$. So, $f(0x0) = 0x0$ and $f(0x1) = 0x1$, and $0x0 \not\parallel_{\mathrm{pref}} 0x1$. Hence by Lemma 4.18, there exists $f_o \in \mathrm{pFix}_V(1)$ such that $f_o(0x0u) = 0x0u$, and $f_o(0x1u) \neq 0x1u$ (for some $u \in \{0,1\}^*$). The latter inequality contradicts the fact that $f(0x) = 0x$ for all $f \in \mathrm{pFix}_V(1)$.

In a similar way one obtains a contradiction if $g^{-1}(0x) = 1y$ for some $x, y \in \{0,1\}^*$.

(2) We prove next that $g \in \mathrm{pFix}_V(0)$.

Suppose $fg = gf$ for all $f \in \mathrm{pFix}_V(1)$; we saw that then $g \in \mathrm{pStab}_V(0)$. If, by contradiction, $g \notin \mathrm{pFix}_V(0)$, then by Lemma 4.17, there exists $0x \in \mathrm{domC}(g)$ such that $0x \not\parallel_{\mathrm{pref}} g(0x) = 0y$.

Then, $fg(0x) = f(0y) = gf(0x)$. By Lemma 4.18 there exists $f_o \in \mathrm{pFix}_V(1)$ such that $f_o(0xu) = 0xu$, and $f_o(0yu) \neq 0yu$ (for some $u \in \{0,1\}^*$). Then $f_o(0yu) = f_o g(0xu) = gf_o(0xu) = g(0xu) = 0yu$. So, $f_o(0yu) = 0yu$, which contradicts $f_o(0yu) \neq 0yu$.

[⇒] Let $g \in \mathrm{pFix}_V(0)$ and $f \in \mathrm{pFix}_V(1))$. Then $\mathrm{domC}(f) = \{1\} \cup 0P$, and $\mathrm{domC}(g) = \{0\} \cup 1Q$, where $P, Q \subset \{0,1\}^*$ are finite maximal prefix codes. So, $0P \cup 1Q$ is a finite maximal prefix code.

Then for every $0x \in 0P$: $fg(0x) = f(0x)$, since $g \in \mathrm{pFix}_V(0)$; and $gf(0x) = f(0x)$, since $f(0x) \in 0\{0,1\}^*$ and $g \in \mathrm{pFix}_V(0)$. So, $fg(0x) = gf(0x)$.

Similarly, for all $1x \in 1Q$: $gf(1x) = g(1x)$, since $f \in \mathrm{pFix}_V(1))$; and $fg(1x) = g(1x)$, since $g(1x) \in 1\{0,1\}^*$ and $f \in \mathrm{pFix}_V(1))$. So, $fg(1x) = gf(1x)$.

Hence, $fg = gf$ on the finite maximal prefix code $0P \cup 1Q$. Hence $fg = gf$ in V. □

Lemma 4.20. *The subgroups $\mathrm{pFix}_V(1)$ and $\mathrm{pFix}_V(0)$ are isomorphic to V.*

Proof. Every element of V has a table of the form $\{(x_i, y_i) : 1 \le i \le n\}$, where $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$ are finite maximal prefix codes over $\{0,1\}$. An isomorphism $V \to \mathrm{pFix}_V(1)$ is given by

$$g = \begin{bmatrix} x_1 & \ldots & x_n \\ y_1 & \ldots & y_n \end{bmatrix} \longmapsto \theta(g) = \begin{bmatrix} 1\,0x_1 & \ldots & 0x_n \\ 1\,0y_1 & \ldots & 0y_n \end{bmatrix}.$$

The map θ is obviously a bijection from V onto $\mathrm{pFix}_V(1)$, and it is easy to check that it is a homomorphism. □

coNP-hardness of the word problem of V over $\Gamma_V \cup \tau$:

The commutation test of Lemma 4.19 reduces the generalized word problem of $\mathrm{pFix}_V(0)$ in V (over $\Gamma_V \cup \tau$) to an infinite set of word problems of V, namely $\{fg = gf : f \in \mathrm{pFix}_V(1)\}$.

However, $\mathrm{pFix}_V(1)$ is 2-generated; this follows from Lemma 4.20 and the fact that V is 2-generated [43,31,12]. Obviously, g commutes with all of $\mathrm{pFix}_V(1)$ iff g commutes with the two generators of $\mathrm{pFix}_V(1)$. This reduces the generalized word problem of $\mathrm{pFix}_V(0)$ in V (over $\Gamma_V \cup \tau$) to the conjunction of two instances of the word problem of V over $\Gamma_V \cup \tau$. Hence, the word problem of V over $\Gamma_V \cup \tau$ is coNP-hard with respect to 2-ary conjunctive polynomial-time reduction.

Theorem 4.21 *(coNP-complete word problem). The word problem of V over the generating set $\Gamma_V \cup \tau$ is* coNP*-complete.*

Proof. By Lemma 4.8, this word problem belongs to coNP. By the reasoning in the above few lines, the word problem is coNP-hard. □

4.5. Alternative proof of coNP-completeness of the word problem of V over $\Gamma_V \cup \tau$

The above proof of coNP-completeness of the word problem of V over $\Gamma_V \cup \tau$ was derived from a similar proof for $G_{3,1}$ [6] (in 2003). Since then, Stephen Jordan [27] (in 2013) proved that the equivalence problem for bijective circuits built from copies of the Fredkin gate is coNP-complete. A *bijective circuit* is an acyclic circuit in which every gate has a permutation of $\{0,1\}^j$ as its input-output function (for some $j > 0$, depending on the gate). The input-output function of such a circuit is a permutation of $\{0,1\}^n$ for some $n > 0$ (see e.g. [41]). The *Fredkin gate*, on an input $x_1x_2x_3 \in \{0,1\}^3$, is defined by

$$\mathsf{F}(0\,x_2x_3) = 0\,x_2x_3,$$
$$\mathsf{F}(1\,x_2x_3) = 1\,x_3x_2;$$

see e.g. [21]. This gate is also called the "controlled transposition" (of x_2 and x_3). Clearly, F is the table of an element of V; moreover, with $\{\mathsf{F}\} \cup \tau$ we can compute $\mathsf{F}(x_ix_jx_k)$ for any three different variables in an input $x_1 \dots x_n$ with $i, j, k \in \{1, \dots, n\}$. Hence Jordan's result can be recast as follows:

Theorem 4.22 *(Thompson group form of Jordan's theorem). The subgroup of V generated by $\{\mathsf{F}\} \cup \tau$ has a* coNP*-complete word problem, with respect to many-one polynomial-time reduction.* □

See [7] (and [41]) for further connections between bijective ("reversible") circuits and the Thompson groups.

Theorem 4.22 immediately implies Theorem 4.21, as the word problem of the subgroup $\langle\{\mathsf{F}\}\cup\tau\rangle_V$ reduces to the word problem of V over $\Gamma_V\cup\tau$ by the inclusion map. (Here we assume that $\mathsf{F}\in\Gamma_V$; if that is not the case we can represent F by a fixed word over Γ_V for the reduction; see Lemma 4.6(2).)

An advantage of our method of subsections 4.2 - 4.4 is that it is direct, whereas Jordan's theorem is based on Barrington's theorem [1], which is itself a deep result. However, using Jordan's theorem has the advantage that it yields the following: The word problem of V over $\Gamma_V\cup\tau$ is coNP-complete with respect to polynomial-time *many-one reduction*. The earlier proof only yields polynomial-time binary conjunctive reduction.

4.6. The shift, and the word problem of nV

For all $\tau_{j,j+1}\in\tau\subset V$ with $j\geq 1$, we define $\tau_{j,j+1}\times\mathbf{1}$: $\{0,1\}^*\times\{0,1\}^*\longrightarrow\{0,1\}^*\times\{0,1\}^*$ by

$$\tau_{j,j+1}\times\mathbf{1}:\ (x,y)\ \longmapsto\ (\tau_{j,j+1}(x),\ y).$$

So, $\mathrm{domC}(\tau_{j,j+1}\times\mathbf{1})=\{0,1\}^{j+1}\times\{\varepsilon\}$.

The *shift* $\sigma\in 2V$ is defined by $\mathrm{domC}(\sigma)=\{\varepsilon\}\times\{0,1\}$, $\mathrm{imC}(\sigma)=\{0,1\}\times\{\varepsilon\}$, and

$$\sigma(\varepsilon,a)=(a,\varepsilon),$$

for all $a\in\{0,1\}$. Hence, $\sigma(x,\,ay)=(ax,\,y)$, for all $a\in\{0,1\}$, and $(x,y)\in\{0,1\}^*\times\{0,1\}^*$.

Lemma 4.23. *For all $j\geq 1$: $\tau_{j,j+1}\times\mathbf{1}(.)\ =\ \sigma^{j-1}\circ(\tau_{1,2}\times\mathbf{1})\circ\sigma^{-j+1}(.)$.*

Proof. For any $(x,y)\in\{0,1\}^*\times\{0,1\}^*$, where $x=u\,x_jx_{j+1}\,v$ with $|u|=j-1\geq 0$, and $v\in\{0,1\}^*$, we have:

$$(u\,x_jx_{j+1}v,\ y)\ \overset{\sigma^{-j+1}}{\longmapsto}\ (x_jx_{j+1}\,v,\ u^{\mathrm{rev}}\,y)\ \overset{\tau_{1,2}\times\mathbf{1}}{\longmapsto}\ (x_{j+1}x_j\,v,\ u^{\mathrm{rev}}\,y)\ \overset{\sigma^{j-1}}{\longmapsto}\ (u\,x_{j+1}x_j\,v,\ y).$$

Here, u^{rev} denotes the reverse of u. □

Proof of Theorem 1.1. By Lemma 3.5 the word problem of nV belongs to coNP.

By Theorem 4.21 the word problem of V over $\Gamma_V\cup\tau$ is coNP-hard. By Lemma 4.23, the word problem of V over $\Gamma_V\cup\tau$, reduces to the word problem of $2V$ over a finite generating set; this reduction is the one-one reduction that replaces every generator $\gamma\in\Gamma_V$ by $\gamma\times\mathbf{1}$, and replaces $\tau_{j,j+1}$ by $\sigma^{j-1}\circ(\tau_{1,2}\times\mathbf{1})\circ\sigma^{-j+1}$, as in Lemma 4.23. We can include the set $\{\gamma\times\mathbf{1}:\gamma\in\Gamma_V\}\cup\{\sigma\}$ into the finite generating set of $2V$, or we can express all the elements of this set by a finite set of strings over some other finite generating set of $2V$. Thus the word problem of $2V$ over a finite generating set is coNP-hard.

To show that word problem of nV (for $n>2$) over a finite generating set is coNP-hard, we use the fact that $2V$ is a finitely generated subgroup of nV, and apply Lemma 4.4(2). □

Remark on the distortion of V in $2V$: Burillo and Cleary [18] show that V is exponentially distorted in $2V$ (when both V and $2V$ are over finite generating sets). In Lemma 4.23 we proved that $\tau_{j-1,j}$ has linear word length in $2V$ (as a function of j); but $\tau_{j-1,j}$ has exponential word length in V over any finite generating set. This, again, shows that the distortion of V in $2V$ is at least exponential.

Indeed, for all $j \geq 2$, $\tau_{j-1,j}$ has a table $ux_{j-1}x_j \in \{0,1\}^j \longmapsto ux_jx_{j-1} \in \{0,1\}^j$ (see the beginning of subsection 4.4). It follows from Lemma 2.2 that the table of $\tau_{j-1,j}$ is maximally extended. So, $\tau_{j-1,j}$ has table-size 2^j. Therefore, by [5, Thm. 3.8], the word length $|\tau_{j-1,j}|_V$ of $\tau_{j-1,j}$ in V (over any finite generating set) satisfies $\alpha\, 2^j \leq |\tau_{j-1,j}|_V \leq \beta\, j\, 2^j$ (for some constants $\alpha, \beta > 0$). On the other hand, the embedding of V into $2V$, used in Lemma 4.23, represents $\tau_{j-1,j}$ by a word of length $2j-3$.

"Why" is the word problem of $2V$ coNP-complete? The table-size of an element $f \in 2V$ can be exponentially larger than the word length of f (over a finite generating set); hence, the polynomial-time algorithm for the word problem of V (consisting of simply composing the tables of the generators) turns into an exponential-time algorithm in $2V$. In V we have the table-size formula $|\mathrm{domC}(f_2 \circ f_1)| \leq |\mathrm{domC}(f_2)| + |\mathrm{domC}(f_1)|$; in $2V$ there is no such formula. However, the length-formula of Lemma 3.2 implies rather directly that the word problem of $2V$ belongs to coNP.

The coNP-hardness is less intuitive. The proof that V (over $\Gamma_V \cup \tau$) can simulate circuits is intuitive (if tedious). The commutation test, reducing a generalized word problem to a word problem, is less intuitive, and it is a priori not related to computing. The alternative proof of coNP-hardness of the word problem of V over $\Gamma \cup \tau$ is derived from Jordan's theorem, which is itself based on Barrington's theorem; the latter has always been considered a surprising result.

Using the shift to represent the infinite set τ by a finite set is easy. But the shift is not a circuit element (although it has a computational meaning, namely, as an operation in multi-stack machines).

Acknowledgment

I would like to thank the referee for a thorough reading of the paper.

References

[1] D.A. Barrington, Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1, J. Comput. Syst. Sci. 38 (1) (1989) 150–164.
[2] C. Bennett, Logical reversibility of computation, IBM J. Res. Dev. 17 (1973) 525–532.
[3] C. Bennett, Time/space tradeoffs for reversible computation, SIAM J. Comput. 18 (1989) 766–776.
[4] J.C. Birget, A. Ol'shanskii, E. Rips, M.V. Sapir, Isoperimetric functions of groups and computational complexity of the word problem, Ann. Math. 156 (2) (Sept. 2002) 467–518.
[5] J.C. Birget, The groups of Richard Thompson and complexity, Int. J. Algebra Comput. 14 (5,6) (Dec. 2004) 569–626. Preprint, https://arxiv.org/abs/math/0204292.
[6] J.C. Birget, Circuits, coNP-completeness, and the groups of Richard Thompson, Int. J. Algebra Comput. 16 (1) (Feb. 2006) 35–90. Preprint, https://arxiv.org/abs/math/0310335.

[7] J.C. Birget, Factorizations of the Thompson-Higman groups, and circuit complexity, Int. J. Algebra Comput. 18 (2) (2008) 285–320.
[8] J.C. Birget, Monoid generalizations of the Richard Thompson groups, J. Pure Appl. Algebra 213 (2) (2009) 264–278.
[9] J.C. Birget, Polynomial-time right-ideal morphisms and congruences, Int. J. Algebra Comput. 28 (05) (Aug. 2018) 791–835.
[10] J.C. Birget, The word problem of the Brin-Thompson groups is coNP-complete, Version 1 of https://arxiv.org/abs/1902.03852, 11 Feb. 2019.
[11] C. Bleak, D. Lanoue, A family of non-isomorphism results, Geom. Dedic. 146 (1) (2010) 21–26.
[12] C. Bleak, M. Quick, The infinite simple group V of Richard J. Thompson: presentations by permutations, Groups Geom. Dyn. 11 (2017) 1401–1436.
[13] W.W. Boone, G. Higman, An algebraic characterization of the solvability of the word problem, J. Aust. Math. Soc. 18 (1974) 41–53.
[14] Matthew G. Brin, Higher dimensional Thompson groups, Geom. Dedic. 108 (2004) 163–192.
[15] Matthew G. Brin, Presentations of higher dimensional Thompson groups, J. Algebra 284 (2) (2005) 520–558.
[16] Matthew G. Brin, On the baker's map and the simplicity of the higher dimensional Thompson groups nV, Publ. Math. 54 (2) (2010) 433–439.
[17] M. Burger, Sh. Mozes, Finitely presented simple groups and products of trees, C.R. Acad. Sci., Paris, Sér. 9 324 (1997) 747–752.
[18] J. Burillo, S. Cleary, Metric properties of higher-dimensional Thompson groups, Pac. J. Math. 248 (1) (2010) 49–62.
[19] J.W. Cannon, W.J. Floyd, W.R. Parry, Introductory notes on Richard Thompson's groups, Enseign. Math. 42 (1996) 215–256.
[20] Ding-Zhu Du, Ker-I. Ko, Theory of Computational Complexity, Wiley-Interscience, 2000.
[21] E. Fredkin, T. Toffoli, Conservative logic, Int. J. Theor. Phys. 21 (1982) 219–253.
[22] M.R. Garey, D.S. Johnson, Computers and Intractability: a Guide to the Theory of NP-Completeness, W.H. Freeman and Co., 1979.
[23] J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science (vol. A), MIT Press, 1990.
[24] J. Hennig, F. Mattucci, Presentations for the higher-dimensional Thompson groups, Pac. J. Math. 257 (1) (2012) 53–74.
[25] G. Higman, Finitely Presented Infinite Simple Groups, Notes on Pure Mathematics, vol. 8, The Australian National University, Canberra, 1974.
[26] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.
[27] S.P. Jordan, Strong equivalence of reversible circuits is coNP-complete, Quantum Inf. Comput. 14 (2014) 1302–1307. Preprint, https://arxiv.org/abs/1307.0836, 2 July 2013.
[28] M.V. Lawson, A. Vdovina, Higher dimensional generalizations of the Thompson groups, arXiv: 1909.13254v1, 29 Sept. 2019.
[29] J. Lehnert, P. Schweitzer, The co-word problem for the Higman-Thompson group is context-free, Bull. Lond. Math. Soc. 39 (2) (2007) 235–241.
[30] Y. Lodha, A finitely presented infinite simple group of homeomorphisms of the circle, https://arxiv.org/abs/1710.06220, Oct. 2017.
[31] D.R. Mason, On the 2-generation of certain finitely presented infinite simple groups, J. Lond. Math. Soc. s2-16 (2) (Oct. 1977) 229–231.
[32] N. Matte Bon, Rigidity properties of full groups of pseudogroups over the Cantor set, https://arxiv.org/pdf/1801.10133.pdf, 10 Dec. 2018.
[33] R. McKenzie, R.J. Thompson, An elementary construction of unsolvable word problems in group theory, in: W.W. Boone, F.B. Cannonito, R.C. Lyndon (Eds.), Word Problems, North-Holland, 1973, pp. 457–478.
[34] David Meier, A note on simple free products, Houst. J. Math. 9 (1983) 501–504.
[35] David Meier, Embeddings into simple free products, Proc. Am. Math. Soc. 93 (1985) 387–392.
[36] Ch. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.
[37] C. Röver, Constructing finitely presented simple groups that contain Grigorchuk groups, J. Algebra 220 (1999) 284–313.
[38] M.V. Sapir, J.C. Birget, E. Rips, Isoperimetric and isodiametric functions of groups, Ann. Math. 156 (2) (Sept. 2002) 345–466.
[39] J.E. Savage, Models of Computation, Addison-Wesley, 1998.

[40] E.A. Scott, A construction which can be used to produce finitely presented infinite simple groups, J. Algebra 90 (1984) 294–322.
[41] V. Shende, A. Prasad, I. Markov, J. Hayes, Synthesis of reversible logic circuits, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 22 (6) (2003) 710–722. Preprint, https://arxiv.org/abs/quant-ph/0207001.
[42] Richard J. Thompson, Manuscript (1960s).
[43] R.J. Thompson, Embeddings into finitely generated simple groups which preserve the word problem, in: S. Adian, W. Boone, G. Higman (Eds.), Word Problems II, North-Holland, 1980, pp. 401–441.
[44] I. Wegener, The Complexity of Boolean Functions, Wiley/Teubner, 1987.