Solutions of word equations over partially commutative structures

Solutions of Word Equations over Partially Commutative Structures*

Volker Diekert[1], Artur Jeż[†2], and Manfred Kufleitner[‡1]

1 **Institut für Formale Methoden der Informatik, Universität Stuttgart, Germany**
2 **Institute of Computer Science, University of Wrocław, Poland**

Abstract

We give NSPACE($n \log n$) algorithms solving the following decision problems. Satisfiability: Is the given equation over a free partially commutative monoid with involution (resp. a free partially commutative group) solvable? Finiteness: Are there only finitely many solutions of such an equation? PSPACE algorithms with worse complexities for the first problem are known, but so far, a PSPACE algorithm for the second problem was out of reach. Our results are much stronger: Given such an equation, its solutions form an EDT0L language effectively representable in NSPACE($n \log n$). In particular, we give an effective description of the set of all solutions for equations with constraints in free partially commutative monoids and groups.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems; F.4.2 Grammars and Other Rewriting Systems; F.4.3 Formal Languages

Keywords and phrases Word equations; EDT0L language; trace monoid; right-angled Artin group; partial commutation

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Free partially commutative monoids (a.k.a. *trace monoids*) and groups (a.k.a. *RAAGs*: *right-angled Artin groups*) are well-studied objects, both in computer science (latest since [18]) and in mathematics (with increasing impact since [24]). For years, decidability of the *satisfiability problem* (i.e., the problem whether a given equation is solvable) over these structures was open. A positive solution for trace monoids was obtained by Matiyasevich [17] and for RAAGs by Diekert and Muscholl [8]. The known techniques did not cope with the *finiteness problem* (i.e., the problem whether a given equation has only finitely many solutions). Decidability of finiteness for trace monoids was wide open, whereas for RAAGs a sophisticated generalization of Razborov-Makanin diagrams and geometric methods, available for groups, yielded decidability [3], but without any complexity estimation.

We give a simple and effective description of the set of all solutions for equations with constraints in free partially commutative monoids and groups; the correctness proof is mathematically challenging. Once the correctness is established, the simplicity is also reflected in a surprisingly low complexity. We give an upper bound of NSPACE($n \log n$) for both satisfiability and finiteness—each problem for trace monoids as well as for RAAGs. Even for satisfiability this complexity improves the previously known upper bounds. On the other hand these problems are NP-hard. It remains open whether NSPACE($n \log n$) is optimal.

* Full proofs can be found on arXiv [6].
† Artur Jeż was supported by a return fellowship of the Alexander von Humboldt Foundation.
‡ Manfred Kufleitner was supported by the grants DI 435/5-2 and KU 2716/1-1 of the DFG.


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

To obtain these results we apply a recent *recompression technique* [12], which was used as a simple method to solve word equations. It uses simple compression operations: compress ab into a letter c; and modify the equation so that such operations are sound. An algebraic setting of the current paper enables a shift of perspective: the inverse operation, replacing c by ab, is an endomorphism. Thus, the set of all solutions of an equation (solvable or not) can be represented as a graph, whose nodes are labeled with equations and edges by endomorphisms of free monoids. This graph can also be seen as a nondeterministic finite automaton (NFA) that accepts a rational set of endomorphisms over a free monoid. (Recall that a subset in a monoid M is *rational* if it is accepted by some NFA whose transitions have labels from M.) It is known that applying a rational set of endomorphisms to a letter yields an EDT0L language [1], and our construction guarantees that the obtained EDT0L language describes exactly the set of all solution of the given equation. Moreover, as usual in automata theory, the structure of the NFA reflects whether the solution set is finite. Last not least, our method is conceptually simpler than all previously known approaches to solving equations over free partially commutative structures.

Studying word equations is part of combinatorics on words for more than half a century [2]. From the very beginning, motivation came partly from group theory: the goal was to understand and parametrize solutions for equations in free groups. For example, Lyndon and Schützenberger needed sophisticated combinatorial arguments to give a parametrized solution to the equation $a^m = b^n c^p$ in a free group [14]. On the other hand, it is known that a parametric description of the solution set is not always possible [10]. The satisfiability of word equations in free monoids and free groups became a main open problem due to its connection with Hilbert's tenth problem. The problem was solved affirmative by Makanin in his seminal papers [15, 16]. His algorithms became famous also due to the difficulty of the termination proof and the extremely high complexity. A breakthrough to lower the complexity was initiated by Plandowski and Rytter [21], who were the first to apply compression techniques on word equations. Compression was also essential in showing that the satisfiability of word equations is in PSPACE [19]. This approach was further developed [12] using the "recompression technique", which simplified all existing proofs for solving word equations; in particular, it provided an effective description of all solutions; a similar representation was given earlier by Plandowski [20]. In free groups, an algorithmic description of all solutions was known much earlier due to Razborov [22]. His description became known as a *Makanin-Razborov diagram*, a major tool in the positive solution of Tarski's conjectures about the elementary theory in free groups [13, 23]. None of these results provided a structural result on the set of all solutions; interest in such results was explicitly expressed [11] by asking whether it is an "indexed language". Apparently, this question was posed without too much hope that a positive answer is within reach. However, the answer was positive for quadratic equations [9] (which is a severe restriction); the general case was established in [4]. Actually, a stronger result holds: the set of all solutions for equations in free monoids (as well as in free groups) is an EDT0L language, which is a proper subclass of indexed languages. The closest results on word equations with partial commutation are in [8], but techniques used there do not apply here as they boil down to a purely combinatorial construction of a normal form and ignore the algebraic structure as well as the set of all solutions.

2 Main result

Given a finite *alphabet* Γ, the *free monoid* Γ^* is the set of all finite words over Γ with concatenation. The empty word is denoted by 1. The length of a word w is denoted by $|w|$;

by $|w|_a$ we count how often the letter a appears in w. A *resource function* $\rho : \Gamma \to 2^{\mathfrak{R}}$ maps elements of Γ to subsets of a finite *set of resources* $\mathfrak{R}$. We assume that $\mathfrak{R}$ is of constant size. The pair (Γ, ρ) is called a *resource alphabet*. If $\rho(a) = S$, then a is called an *S-constant*; a nonempty sequence of S-constants is an *S-run*.

A *resource monoid* $M(\Gamma, \rho)$ is the quotient of all finite words Γ^* by a partial commutation: $M(\Gamma, \rho) = \Gamma^* / \{ab = ba \mid \rho(a) \cap \rho(b) = \emptyset\}$, i.e., letters $a \neq b$ commute if and only if they do not share a resource. Resource monoids can equivalently be seen as *free partially commutative monoids* or *trace monoids*. We choose the resource-based approach as it best suits our purposes. Elements of a resource monoid are called *traces*. The natural projection π maps elements of the free monoid Γ^* to traces in $M(\Gamma, \rho)$; this is not a bijection and we view $w \in \Gamma^*$ as a word representation of the trace $\pi(w)$. In a monoid, an element v is a *factor* of w if $w = pvq$ for some p, q. We assume that the monoid $M(\Gamma, \rho)$ is equipped with an *involution*, that is, a bijection $x \mapsto \overline{x}$ on $M(\Gamma, \rho)$ such that $\overline{\overline{x}} = x$, $\overline{xy} = \overline{y}\,\overline{x}$ for all $x, y \in M(\Gamma, \rho)$. To make the definition well defined, we require that $\rho(x) = \rho(\overline{x})$ for $x \in \Gamma$. In the following, a *trace monoid* means a *resource monoid with involution*. A *morphism* $\varphi : M \to M'$ between monoids with involution is a homomorphism additionally respecting the involution. If Δ is a subset of M, then we often denote the restriction of φ to Δ by φ. If $\varphi(d) = d$ for all $d \in \Delta$, then φ is a *Δ-morphism*.

If there is no letter $a \in \Gamma$ with $a = \overline{a}$, then, by adding defining relations $a\overline{a} = 1$ for all $a \in \Gamma$, we obtain the *free partially commutative group* $G(\Gamma, \rho)$. Free partially commutative groups are also known as *right-angled Artin groups* or *RAAGs* for short. As a set, we can identify a RAAG $G(\Gamma, \rho)$ with the subset traces of the trace monoid $M(\Gamma, \rho)$ without factors $a\overline{a}$. Such traces are called *reduced*. We take inversion on groups as involution; the canonical projection of the monoid $M(\Gamma, \rho)$ onto the group $G(\Gamma, \rho)$ respects the involution.

Let (Γ, ρ) be a resource alphabet. An *equation* is a pair of words (U, V) over an alphabet $\Gamma = A \cup \mathcal{X}$ has a partition into *constants* A and *variables* $\mathcal{X}$, both sets are closed under involution. A *constraint* is a morphism $\mu : M(\Gamma, \rho) \to N$, where N is a finite monoid with involution. For our purposes, it is enough to consider constraints such that the elements of N can be represented by $\mathcal{O}(\log |\Gamma|)$ bits, and that all necessary computations in N (multiplication, involution, etc.) can be performed in space $\mathcal{O}(\log |\Gamma|)$ and the specification of these operations requires $\mathcal{O}(|\Gamma| \log |\Gamma|)$ space. If (U, V) is an equation over (Γ, ρ), then we define the input size of an equation with constraints as $n = |UV| + |\Gamma|$.

We write (U, V, μ) for an equation (U, V) with constraints μ. A *solution* of (U, V, μ) over $M(A, \rho)$ is an A-morphism $\sigma : M(A \cup \mathcal{X}, \rho) \to M(A, \rho)$ such that $\sigma(U) = \sigma(V)$ and $\mu\sigma(X) = \mu(X)$ for all $X \in \mathcal{X}$. If the equation is over $G(A, \rho)$, then instead of $\sigma(U) = \sigma(V)$ we require $\pi\sigma(U) = \pi\sigma(V)$ for the canonical projection $\pi : M(A, \rho) \to G(A, \rho)$. We also say that σ *solves* (U, V, μ) in $M(A, \rho)$ (resp. in $G(A, \rho)$). For equations over $G(A, \rho)$ we only allow solutions where the trace $\sigma(X)$ is reduced for all $X \in \mathcal{X}$. The main result of this paper is that the set of all solutions of a trace equation (resp. an equation in a RAAG) with rational constraints is an effectively computable EDT0L language, and the underlying automaton reflects whether there are infinitely many solutions.

▶ **Theorem 1. [Monoid version]** *There is an* NSPACE$(n \log n)$ *algorithm for the following task. The input is a resource alphabet $(A \cup \mathcal{X}, \rho)$ with involution and a trace equation (U, V, μ) with constraints μ in constants A and variables $\mathcal{X} = \{X_1, \ldots, X_k\}$. The algorithm computes an alphabet $C \supseteq A$ of size $\mathcal{O}(n)$, constants $c_1, \ldots, c_k \in C$, and an NFA $\mathcal{A}$ accepting a rational set $\mathcal{R}$ of A-endomorphisms on C^* such that: $h(C^*) \subseteq A^*$ for all $h \in \mathcal{R}$ and under the canonical projection $\pi : A^* \to M(A, \rho)$ we have*

$$\{(\pi h(c_1), \ldots, \pi h(c_k)) \mid h \in \mathcal{R}\} = \{(\sigma(X_1), \ldots, \sigma(X_k)) \mid \sigma \text{ solves } (U, V, \mu) \text{ in } M(A, \rho)\}\,.$$

Thus, the set of all solutions is an effectively computable EDT0L language. Furthermore, (U, V, μ) has a solution if and only if $\mathcal{A}$ accepts a nonempty set; (U, V, μ) has infinitely many solutions if and only if $\mathcal{A}$ has a directed cycle. These conditions can be tested in $\mathsf{NSPACE}(n \log n)$.

[Group version] *The same, but solutions σ satisfy $\sigma(U) = \sigma(V)$ in the RAAG $G(A, \rho)$ and for a variable X the solution $\sigma(X)$ is restricted to be a reduced trace.*

Theorem 1 generalizes to systems of equations. Another generalization are finitely generated *graph products* with involution over free monoids, free groups, and finite groups. See [7] for a definition and the known results concerning solvability of equations in graph products. This generalization is rather technical but does not reveal new ideas; it is done elsewhere.

3 Basic concepts

Equations in RAAGs can be reduced to equations in resource monoids [8], such an approach is standard since its introduction for free groups [5], which are reduced to free monoids. In essence, the reduction simulates the inverse operation by involution and it enforces that the solution in the monoids is in the reduced form by (additional) constraints. We employ a similar approach; thus, our presentation focuses on the equations over resource monoids.

Using a standard technique one can ensure that there are no self-involuting constants in the initial equation [8]. This step is not needed for RAAGs as $a = \overline{a} = a^{-1}$ implies $a^2 = 1$ in groups, but RAAGs are torsion-free. For technical reasons we introduce a new special symbol $\#$, which serves as a marker and becomes the only self-involuting constant; set $\rho(\#) = \mathfrak{R}$ and $\emptyset \neq \rho(a) \subsetneq \mathfrak{R}$ for all other constants. We let

$$W_{\text{init}} = \#X_1\# \cdots X_k\#U\#V\#\overline{U}\#\overline{V}\#\overline{X_k}\# \cdots \overline{X_1}\#.$$

During the process, the $\#$'s will not be touched, so we keep control over the prefix corresponding to $\#X_1\# \cdots \#X_k\#$ which encodes the tuples $(\sigma(X_1), \ldots, \sigma(X_k))$. Moreover, we have $\sigma(U) = \sigma(V)$ if and only if $\sigma(W) = \sigma(\overline{W})$. Thus, we can treat a single trace as an equation. Solutions become A-morphisms σ satisfying $\sigma(W) = \sigma(\overline{W})$.

The equations that we consider are over a more general structure than a trace monoid. To simplify the notion, we denote the equation and a monoid over which it is by a tuple $(W, B, \mathcal{X}, \rho, \theta, \mu)$, where $W \in (B \cup \mathcal{X})^*$ is the "equation" with constants B and variables $\mathcal{X}$, the mapping $\rho : B \cup \mathcal{X} \to 2^{\mathfrak{R}}$ is the resource function, and $\mu : M(B \cup \mathcal{X}, \rho) \to N$ represents the constraints (given by the mapping $\mu : B \cup \mathcal{X} \to N$). Since $2^{\mathfrak{R}}$ is a commutative monoid, we shall view ρ as a morphism from $M(B \cup \mathcal{X}, \rho)$ to $2^{\mathfrak{R}}$, too. The symbol θ refers to a "type" which adds partial commutation. A type is given by a list of certain pairs $(x, y) \in (B \cup \mathcal{X})^+ \times B^+$; and each such pair yields a defining equation $xy = yx$. For example, we typically have $(X, y) \in \theta$ when considering a solution σ with $\sigma(X) \in y^+$. Then $\rho(X) = \rho(y)$, but we wish to use the fact that $\sigma(X)y = y\sigma(X)$. This is the purpose of a type. We only use types in the subprocedures of block and quasi-block compression, see Section 4.3.1. Such a monoid is denoted as $M(B \cup \mathcal{X}, \rho, \theta, \mu)$. In most cases, θ is empty. Then we use $(W, B, \mathcal{X}, \rho, \mu)$ as an abbreviation of $(W, B, \mathcal{X}, \rho, \emptyset, \mu)$ and $M(B \cup \mathcal{X}, \rho, \theta, \mu)$ as an abbreviation of $M(B \cup \mathcal{X}, \rho, \mu)$.

A *B-solution* of $(W, B, \mathcal{X}, \rho, \mu)$ is a B-morphism $\sigma : M(B \cup \mathcal{X}, \rho, \mu) \to M(B, \rho, \mu)$ such that $\sigma(W) = \sigma(\overline{W})$ (i.e., it solves the equation) and $\mu(\sigma(X)) = \mu(X)$ (i.e., it satisfies the constraints).

During the algorithm we "increase the resources" of constants. It is useful to assume that for every constant $a \in A$ and every set of resources S with $\rho(a) \subsetneq S$ the alphabet A

has a corresponding constant with set of resources S. We denote such a constant by (a, S), the involution on it is defined by $\overline{(a, S)} = (\overline{a}, S)$. We naturally identify a with $(a, \rho(a))$. We assume that the initial alphabet A is closed under taking such constants, i.e., if $a \in A$ and $\rho(a) \subseteq S$, then $(a, S) \in A$.

In some cases, when we "increase the resources", we prefer to use a fresh constant of appropriate resources: For a constant a with $\rho(a) \subsetneq S$, by $[a, S]$ we denote a "fresh" S-constant outside A such that $\rho([a, S]) = S$, $\mu([a, S]) = \mu(a)$ and $\overline{[a, S]} = [\overline{a}, S]$; replacing a with $[a, S]$ is called *lifting*.

During the algorithm we perform various operations on variables and constants. As a rule, whenever we perform such an operation, we perform a symmetric action on the involuted constants/variables. That is, whenever we replace X by aX, we replace $\overline{X}$ by $\overline{X}\,\overline{a}$; and when we replace ab by c, then we also replace $\overline{b}\,\overline{a}$ by $\overline{c}$. This simplifies the description, as actions performed on "the right side" of X are actions performed on "the left side" of $\overline{X}$.

Whenever we perform operations on variables/constants, we want the constraints and resources to remain unaffected (except for lifting, in which case we explicitly change the set of resources); if we replace a trace W by a trace W', then (if not explicitly stated otherwise) we ensure that $\rho(W) = \rho(W')$ and $\mu(W) = \mu(W')$. For instance, when replacing X by aX', we set $\mu(X')$ so that $\mu(aX') = \mu(X)$. The same applies to ρ. Similarly, when replacing ab by c, we set $\rho(c) = \rho(ab)$ and $\mu(c) = \mu(ab)$. In particular, we do not mention in the description of the procedures that we perform such operations.

A trace $a_1 a_2 \cdots a_n$ has many word representations and we would like to formalize a notion that some constants occur before others (in all word representations). To this end consider a set of positions $\{1, 2, \ldots, n\}$ and the smallest partial order $\preceq$ such that $i \preceq j$ if both $i \leq j$ and $\rho(a_i) \cap \rho(a_j) \neq \emptyset$. A *Hasse diagram* $H(W)$ of a trace $W = a_1 a_2 \cdots a_n$ is a graph with a set of nodes $\{1, 2, \ldots, n\}$, labeled with $a_1, a_2, \ldots, a_n$. It contains (directed) edges between immediate successors, i.e., (i, j) is an edge if $i \prec j$ and $i \preceq k \preceq j$ implies $k \in \{i, j\}$. By a standard result in trace theory [18], we have $W = W'$ in $M(\Gamma, \rho)$ if and only if $H(W)$ and $H(W')$ are isomorphic as abstract node-labeled directed graphs. When considering traces we usually work with their Hasse diagrams. If this causes no confusion, we identify $W = a_1 \cdots a_n$ with $H(W)$ and refer to labels $a_1, a_2, \ldots$ rather than to node names.

A constant $a \in A$ is *minimal* in a trace W if it is minimal in its Hasse diagram, which means that $W = aY$ for some trace Y. We denote the set of minimal elements of W by $\min(W)$. Maximal elements are left-right dual; they are denoted by $\max(W)$.

An arc $a \to b$ is an *S-arc* if $S \in \{\rho(a), \rho(b)\}$; it is *balanced* if $\rho(a) = \rho(b)$, *unbalanced* otherwise.

4 NFA recognising the set of all solutions

In this section we define the NFA $\mathcal{A}$ that recognises the set of all solutions of a trace equation, treated as a set of endomorphisms $\mathrm{End}(C^*)$ over an alphabet $C \supseteq A$.

4.1 The automaton

We first fix an alphabet of constants $C \supseteq A$ of size κn, where κ is a suitable constant (which depends on the number of resources $|\mathfrak{R}|$, viewed as $\mathcal{O}(1)$) and a set of variables Ω with $|\Omega| \leq |C|$. Henceforth, we assume $A \subseteq B \subseteq C$ and $\mathcal{X} \subseteq \Omega$.

The states of the automaton $\mathcal{A}$ are equations of the form $(W, B, \mathcal{X}, \rho, \theta, \mu)$. Each state $V = (W, B, \mathcal{X}, \rho, \theta, \mu)$ has a *weight* $\|V\|$ which is a 5-tuple of natural numbers:

$$\|V\| = (|W|, \omega, \omega', |\theta|, |B|) \in \mathbb{N}^5$$

with $\omega = \sum_{a \in B} (|\mathfrak{R}| - |\rho(a)|) \cdot |W|_a$ and $\omega' = |W| - |\{a \in B \mid |W|_a \geq 1\}|$. We order tuples in $\mathbb{N}^5$ lexicographically. The NFA contains only states V whose *max-norm* $\|V\|_\infty = \max\{|W|, \omega, \omega', |\theta|, |B|\} \in \mathbb{N}$ is at most $\kappa' n$ for a suitable constant κ'.

The initial state is $(W_{\text{init}}, A_{\text{init}}, \mathcal{X}_{\text{init}}, \rho_{\text{init}}, \mu_{\text{init}})$, it corresponds to the input equation. A state $(W, B, \emptyset, \rho, \mu)$ without variables is final if $W = \overline{W}$ has the prefix $\#c_1\# \cdots \#c_k\#$, where $c_1, \ldots, c_k$ are the distinguished constants. We require that the initial state has no incoming and the final states no outgoing transitions.

The transitions, say between $V = (W, B, \mathcal{X}, \rho, \theta, \mu)$ and $V' = (W', B', \mathcal{X}', \rho', \theta', \mu')$, are labeled by A-morphisms and they either affect the variables (*substitution transitions*), or the monoid (*compression transitions*). The former is formalized using a B-morphism $\tau : M(B \cup \mathcal{X}, \rho, \theta, \mu) \to M(B' \cup \mathcal{X}', \rho', \theta', \mu')$. In this case we put several requirements on the equations: the new equation should be obtained by substitution $\tau(X)$ for each X, there are no new constants, resources and constraints of X and $\tau(X)$ should be the same; this is formalized as

$$W' = \tau(W), \quad B' = B, \quad \rho = \rho'\tau, \quad \mu = \mu'\tau. \tag{1}$$

Moreover, $\tau(X)$ is either the empty word (it removes X from W) or $\tau(X) \in \mathcal{X}^* B^+ \mathcal{X}^*$ (at least one constant pops up in the substituted variable). Note that the requirement $W' = \tau(W)$ implicitly upper-bounds the *size* $\|\tau\|$, defined as $\sum_{a \in B \cup \mathcal{X}} |\tau(a)|$, to be linear.

Furthermore, as $B' = B$ we have a natural identity morphism from $M(B', \rho', \theta', \mu')$ to $M(B, \rho, \theta, \mu)$, call it the *associated morphism* and denote it by ε. This morphism labels the transition, its direction is opposite of the transition and τ; we denote the transitions from V to V' with a corresponding morphism h by $V \xrightarrow{h} V'$.

A compression transition leaves the variables invariant and so it is defined by an $(A \cup \mathcal{X})$-morphism $h : M(B' \cup \mathcal{X}, \theta', \rho') \to M(B \cup \mathcal{X}, \theta, \rho)$, note that it could be that $\theta \neq \theta'$ which corresponds to a type introduction or removal; this is the associated morphism in this case. A morphism h defined by, say $h(c) = ab$, represents a compression of a factor ab into a single letter c. For its properties, W is obtained by decompression of new constants, and the resources and constraints are preserved:

$$W = h(W'), \quad \rho' = \rho h, \quad \mu' = \mu h. \tag{2}$$

As in the case of substitutions, the assumption that $W = h(W')$ implicitly upper-bounds $\|h\|$ to linear values.

The transition in the NFA is in the direction of the compression which is opposite to direction of the morphism h. Note that $W = h(W')$ implies $\|V'\| < \|V\|$. For technical reasons we do not allow compression transitions which introduce self-involuting letters (such as $c \mapsto a\overline{a}$); we never compress the marker symbol $\#$. Moreover, following the last compression transition to final states, the restriction $\|V'\| < \|V'\|$ is not applied.

So far the defined NFA can have many useless states, so as a last step we *trim* the automaton, i.e., we remove all vertices not appearing on some accepting path.

The algorithmic part is finished: $\mathcal{A}$ can be constructed using standard arguments in $\mathsf{NSPACE}(n \log n)$.

By the usual definition, the recognized language $\mathcal{R}$ consists of all A-morphisms $h_1 \cdots h_k$, where $h_1, \ldots, h_k$ are consecutive labels on an accepting path. We claim that the set of all

solutions is exactly $\{(\pi h(c_1), \ldots, \pi h(c_k)) \mid h \in \mathcal{R}\}$ where $\pi : A^* \to M(A, \rho)$ is the natural projection.

The correctness proof boils down to show that we can calculate the exact constants κ, κ' (depending on $\mathfrak{R}$ but not on n) and to prove soundness and completeness, i.e., that $h \in \mathcal{R}$ yields a solution and that every solution can be obtained in this way. Out of those, soundness is relatively easy to show, see Section 4.2, the completeness argument spans over Sections 4.3–4.4. Those arguments also show the other claims on the automaton (conditions for emptiness and acyclicity).

4.2 Soundness

As the final states have only one solution (identity), using an induction on the following Lemma, any accepting path labeled with $h_1, \ldots, h_k$ yields a solution $\pi h_1 \cdots h_k$, which shows soundness.

▶ **Lemma 2.** *Given two states $V = (W, B, \mathcal{X}, \rho, \theta, \mu)$ and $V' = (W', B', \mathcal{X}', \rho', \theta', \mu')$, if $V \xrightarrow{h} V'$ and V' has a B'-solution σ' then V has a B-solution $\sigma = h\sigma'$.*

The proof follows by a mechanical application of (1) or (2).

4.3 On-the-fly construction of the NFA

While we described the NFA recognizing all solutions, we did not discuss how to find the appropriate constants κ, κ' nor how to show completeness. For this it is easier to first describe the construction as an "on-the-fly" algorithm, that is, given an equation $(W, B, \mathcal{X}, \rho, \theta, \mu)$ (= current state V of the NFA) and its B-solution σ we will transform it into a different equation $(W', B', \mathcal{X}', \rho', \theta', \mu')$ (= next state V' of the NFA) and a corresponding B'-solution σ', where $V \xrightarrow{h} V'$ and $\sigma = h\sigma'$. Thus we moved from one state of the NFA to the other, without the knowledge of the full NFA. Note that the solutions are not given explicitly, but they are "used" in the nondeterministic choices of the algorithm.

For a fixed set of resources S traces consisting only of S-constants and variables behave as words and we apply to them the known recompression approach: we iteratively apply compression operations to S-runs (so we replace S-runs by new single S-constants). Those operations are applied on constants in the equation, but conceptually we apply them to a solution of the trace equation. To make this approach sound, we also modify the variables, by popping S-constants from them. We apply these operations until each S-run (in the solution) is reduced to a single S-constant; when the compressions and popping operations are applied in appropriate order, the size of the trace equation remains linear.

Compression of S-runs alone is not enough, as there are constants of different resources in the solution of the trace equation. To remedy this, we gradually linearize the solution. This is done by increasing the set of resources of particular constants: when we compressed each S-run to a single constant, we lift all S-constants, so that all S-constants and S-variables are eliminated from the equation. To make the whole approach work, we define an order $\leq$ on sets of resources: it is any linear order that extends the partial ordering by the size, i.e., $|S| \leq |T|$ implies $S \leq T$. A set of resources S is called *minimal* (for a solution σ), if it is minimal according to $\leq$ in the set $\{T \mid \text{there is a } T\text{-constant in } \sigma(W)\}$. We process the sets of resources according to $\leq$, each time treating a minimal sets of resources.

4.3.1 Fixed resources

We define the actions of the algorithm eliminating the S-constants for a fixed minimal set of resources S. To thi end, we need some notions of "easy" and "difficult" factors of $\sigma(W)$.

▶ **Definition 3.** Let $(W, B, \mathcal{X}, \rho, \mu)$ be a state and σ its B-solution. A factor v of $\sigma(W)$ is *visible* if for some occurrence at least one of its positions is obtained form a position labeled by a constant in W; a factor is *invisible* if it is not visible. A trace v is *crossing* if for some occurrence of v in $\sigma(W)$ some but not all positions belong to the substitution of a variable X by $\sigma(X)$; and this occurrence is *visibly crossing*. A trace is *noncrossing* if it is not crossing.

The factors that we typically consider are *pairs*, i.e., ab where $a \neq b \neq \overline{a}$, *$a$-blocks*, i.e., a maximal factor of the form a^ℓ (this occurrence of a^ℓ if not part of a factor $a^{\ell+1}$), and *a-quasi-blocks*, i.e., $(a\overline{a})^\ell$ that is not part of a factor $(a\overline{a})^{\ell+1}$. In the latter case, $a\overline{a}$ is called a *quasi-letter*. The intuitive meaning of a quasi-letter is that we cannot compress $a\overline{a}$ into a single constant as it is would be self-involuting, hence we treat those two letters as if they were a single constant.

Given a subalphabet $S_\pm$, we consider an *involuting partition* (S_+, S_-) that satisfies the conditions $\overline{S_+} = S_-$, $S_+ \cap S_- = \emptyset$ and $S_+ \cup S_- = S_\pm$. Such a partition is *crossing* if at least one pair $ab \in S_+S_-$ is; it has crossing quasi-blocks if there is $a \in S_+$ that has crossing quasi-blocks. Lastly, $S_\pm$ has crossing blocks if there is $a \in S_\pm$ that has crossing blocks.

Pair compression is implemented essentially in the same way as in the case of word equations. Given a pair ab with $a \neq b \neq \overline{a}$ we want to replace each factor ab in $\sigma(W)$ with a fresh constant c. This is easy, when ab is noncrossing: it is enough to perform this operation on W and each $\sigma(X)$, the latter is done implicitly and we obtain a different solution σ' in this way. We also set ρ and μ for c appropriately: $\rho(c) = \rho(ab)$ and $\mu(c) = \mu(ab)$. Performing several such compressions is possible for $ab \in S_+S_-$, where (S_+, S_-) is a noncrossing involuting partition, as for each constant in $\sigma(W)$ we can uniquely determine to which replaced pair it belongs (if any). We do not compress pairs $a\overline{a}$, though, as this would create a self-involuting letter.

We need to ensure that indeed (S_+, S_-) is noncrossing. A pair $ab \in S_+S_-$ is crossing if aX is a factor of W and $b \in \min(\sigma(X))$. The other option is that Xb is a factor and $a \in \max(\sigma(X))$; it is taken care by considering $\overline{b}\,\overline{X}$ and the pair $\overline{b}\overline{a}$. Then we replace X with bX. After doing this, for all variables, the partition (S_+, S_-) is noncrossing and so we can compress pairs in this partition.

Pair compression cannot be applied to aa, as it makes the compression of longer blocks ambiguous. However, when a has no crossing block, (in several steps) we replace each a-block a^λ by c_λ. Similarly as in the case of pair compression, we can compress blocks of several letters in parallel, as blocks of different letters do not overlap.

Again, to apply this subprocedure we need to ensure that each $a \in S_\pm$ has no crossing blocks. Given a visibly crossing block a^ℓ, popping one node may be not enough as this block may still be crossing. Thus for each variable X we pop its whole a-prefix whenever $a \in \min(X) \cap S$, where a^ℓ is the a-prefix of a trace V when ℓ is maximal with $V = a^\ell V'$.

We do not apply the pair compression to $a\overline{a}$ as this introduces self-involuting letters. Instead, we perform a variant of block compression on them: the quasi-block compression. We replace each a-quasi-block $(a\overline{a})^\lambda$ with $c_\lambda\overline{c_\lambda}$; note that we treat a and $\overline{a}$ asymmetrically. We again perform this operation in parallel (in several steps), for all $a \in S_+$, where (S_+, S_-) is an involuting partition.

For uncrossing of quasi-blocks we act the same as for uncrossing of blocks, but we pop the whole $(a\overline{a})$-prefix when $a \in S_+$; the $a\overline{a}$ prefix of V is the longest factor $V' \in \overline{a}(a\overline{a})^* \cup (a\overline{a})^*$

such that $V = V'V''$.

Using those operations we can process a minimal set of resources S: We iterate the following operations as long as something changes in the equation. For each variable we guess whether it has a minimal S-letter and if so we pop this letter. Then we compute the set $S_\pm$ of visible S-constants. We uncross blocks from $S_\pm$ and then compress blocks of $S_\pm$. We then arbitrarily partition $S_\pm$ into an involuting partition (S_+, S_-). Then we uncross quasi-blocks for S_+ and then compress quasi-blocks from S_+. We again partition $S_\pm$ into an involuting partition (S_+, S_-); the partition is chosen so that there are many occurrences of pairs in S_+S_- in the equation, see the appendix. Finally, we uncross (S_+, S_-) for pair compression and perform the pair compression for (S_+, S_-).

Using similar arguments as in the case of word equations, one can show that the procedure FixedResources(S) for a fixed set of resources uses linear space. Concerning the S-runs after FixedResources(S), ideally all S-runs are of length 1 and are either visible or invisible. This is not entirely true, as $a\overline{a}$ cannot be compressed, but those are the longest visible S-runs that can prevail.

▶ **Lemma 4.** *Let S be minimal. The length of the equation during* FixedResources(S) *is linear. After* FixedResources(S) *there are no crossing S-runs, no S-variables. Furthermore, visible S-runs have length at most 2.*

4.3.2 Lifting arcs

Compression of S-runs alone is not enough, as there are runs for different sets of resources. To remedy this we linearize the trace, for technical reasons it is easier to lift whole Hasse arcs rather than individual nodes.

To lift a Hasse arc $e = (a \to b)$ we want to relabel its ends by $[a, \rho(a) \cup \rho(b)]$ and $[b, \rho(a) \cup \rho(b)]$, i.e., by fresh $(\rho(a) \cup \rho(b))$-constants. For correctness reasons we need to also lift the edges that "correspond" to e; moreover, as in the case of compression, lifting may be difficult when an arc connects constants in the equation with constants in the substitution for a variable. Those notions are formalized below.

▶ **Definition 5.** Let $(W, B, \mathcal{X}, \rho, \mu)$ be a state and σ its B-solution. A Hasse-arc $a \to b$ in $\sigma(W)$ is *visible* (*invisible*, *visibly crossing*) if the corresponding factor ab in $\sigma(W)$ has this property. Let $\sim$ be the smallest equivalence relation which satisfies the following conditions:

- If $e = (a \to b)$ in $\sigma(W)$ and $f = (\overline{b} \to \overline{a})$ is the corresponding arc in $\sigma(\overline{W})$, then $e \sim f$.
- If e is invisible and inside some $\sigma(X)$ where $X \in \mathcal{X}$ and f is a corresponding arc in some different $\sigma(X)$, then $e \sim f$.

We say that e is *crossing* if there is exists a visibly crossing f with $f \sim e$; e is *free* otherwise.

Note that for arcs the notion of crossing/free is finer than for traces: since it is possible that $e \not\sim e'$ while both are of the form $(a \to b)$, in particular e could be free and e' crossing.

When $e = (a \to b)$ is a free unbalanced arc, the promised linearization of traces is done through *lifting*: let $S = \rho(a) \cup \rho(b)$, then for $f \sim e$ we change the label on each of its ends from $c \in \{a, b, \overline{a}, \overline{b}\}$ to $[c, S]$. Note that this balances f. To make this operation well defined, we partially linearizes a trace: each position that was before (after) any of relabeled a, b is now before (after) both of $[a, S], [b, S]$ (the same is done for arc $\overline{b} \to \overline{a}$).

We can lift free arcs "for free", but some S-arcs may be crossing. Freeing them is similar to uncrossing factors, but we need to take into the account that $\rho(a) \neq \rho(b)$. Thus ab could be a crossing arc in aX and b is not a minimal element of $\sigma(X)$, so it cannot be popped. Freeing is done in two stages: first we deal with the case when b is an S-letter.

Then for $\sigma(X) = PbQ$, such that $S \neq \rho(P) \subsetneq \rho(X)$ we pop the whole P, which is done by introducing a fresh variable, i.e., we substitute $X \mapsto X'bX$. The new solution is $\sigma'(X') = P$ and $\sigma'(X) = Q$. Then we deal with the case when a is an S-letter (and b not). Thus for $\sigma(X) = PbQ$, where $\rho(a) \cap \rho(P) = \emptyset$, we substitute $X \mapsto X'bX$. The new solution is $\sigma'(X') = P$ and $\sigma'(X) = Q$. Those operations are called splitting of variables. Observe that the first splitting can be done for any set of S-constants and all variables in parallel, while the second can be performed in parallel for all variables and any set of constants that is a subset of $\{b \mid \rho(b) \cap S \neq \emptyset\}$.

We want to lift all unbalanced S arcs, but this is not possible for *all* such arcs in parallel due to involution: for an S-letter a and a trace bac we have to choose which arc, $b \to a$ or $a \to c$, we lift. But it can be done in stages: let (S_+, S_-) and (T_+, T_-) be involuting partitions of all S-constants and all constants having a common resource with S, i.e., $\{a \mid \rho(a) \cap S \neq \emptyset\}$. Then we process all S arcs in four groups S_+T_+, S_-T_+, S_+T_- and S_-T_-; processing of each one is similar, we describe processing of one — S_+T_+. We first split the variables for S_+ and then for T_+, as described above. Then each arc $(a \to b)$ with $ab \in S_+T_+$ is free, thus we lift those arcs. We continue with groups S_-T_+, S_+T_- and S_-T_-. Note that the processing may introduce new crossing arcs, but it can be shown that they are always in next groups. Afterwards, there are no S arcs.

Let $\mathsf{Remove}(S)$ be the above procedure for lifting the S-letters. It is easy to show that after $\mathsf{Remove}(S)$ all S-constants and S-variables are eliminated.

▶ **Lemma 6.** *After* $\mathsf{Remove}(S)$ *there are neither* S*-constants nor* S*-variables in* $\sigma(X)$*.*

4.3.3 The algorithm

TrEqSat considers possible sets of resources S in order $\leq$ on them. For a fixed S it first runs $\mathsf{FixedResources}(S)$ and then $\mathsf{Remove}(S)$.

4.4 Analysis

We begin with estimating the space usage. Firstly we upper-bound the number of introduced variables: they are introduced only during splitting of variables, which happens $\mathcal{O}(1)$ times per resource set, and each variable introduces $\mathcal{O}(1)$ variables, which have less resources; this yields that the number of occurrences of variables is linear in the size of the input equation.

We then estimate the length of the equation, which is also linear in the size of the input equation: Here the estimations are similar as in the case of word equations. For a fixed resource set S we claim that the number of S-constants in the equation stays linear and that processing S introduces in total $\mathcal{O}(1)$ constants per variable. Together with the estimation on the number of variables this yields a bound on the size of the equation.

This guarantees that our algorithm does not exceed a space limit, but may loop forever. Thus we want to show that solutions in consecutive steps get "smaller". Unfortunately, the length of $\sigma(W)$ is not good enough for our purposes, but we can define the weight of the solutions (for an equation) and indeed show that our subprocedures decrease it. This guarantees termination.

We then move to the correctness of the algorithm, i.e., we show how the algorithm transforms the solutions between different equations obtained on the way. In a first step we equip each solution with a function that tells us, what solution of the *input* equation it represents. Then we show that if subprocedure transforms one equation into the other, then the morphism associated with this transition transforms the solution of the latter equation to a solution of the former, so that they they represent the same solution of the input equation.

4.4.1 Space usage

The below estimations of space usage do not depend on the nondeterministic choices, they apply to *all* executions of the algorithm.

Comparing to the algorithms in the free monoid case, the main difference is that our algorithm introduces new variables to the equation. This is potentially a problem, as the whole recompression is based on the assumption that the number of constants is not altered. However, we can still bound the number of introduced variables.

▶ **Lemma 7.** *During* TrEqSat *there are $\mathcal{O}(n)$ occurrences of variables in the trace equation.*

Fix a variable X for which initially $T = \rho(X)$. Observe that $\rho(X)$ cannot increase, though it can decrease: resources increase by lifting arcs and we only lift free arcs, thus, each resource of the new constant was present on one of the ends of the arc. On the other hand, popping constants as well as splitting may decrease the resources of a variable.

We say that X *directly created* an occurrence of X' when X' was created in Split when it considered X; X created X' when there is a sequence $X = X_1, X_2, \ldots, X_k = X'$ such that X_i directly created X_{i+1}. Consider a variable X, it can be split at most eight times during lifting of crossing arcs when we consider $T' \subseteq \mathfrak{R}$. This gives all variables that are directly created by X. Note, that each of the directly generated variable has less resources than X: when we replace X with $X'bX$, then we require that $\rho(X') \subsetneq \rho(X'bX)$.

Let $f(k)$ be the maximal number of occurrences of variables that can be created by a variable with at most k resources. Using the above analysis we can write a recursive formula for f; as the number of resources is a constant, this yields the bound.

We show that during TrEqSat the length of the trace equation is linear in the size of the variables, this is similar as in the case of word equations and in fact the proof proceeds using similar steps. First, we focus on FixedResources and its processing of a fixed set of resources S. In each application of the while loop we introduce $\mathcal{O}(1)$ S-constants per variable (in case of block and quasi-block compression we may introduce long blocks but they are replaced with $\mathcal{O}(1)$ constants afterwards). On the other hand, using standard expected value argument we can show that compression of a randomly chosen partition results in removal of a constant fraction of S-constants from the equation.

Comparing the number of constants in the equation before and after processing S, it increases only by the S-factors that were popped from variables. There are $\mathcal{O}(1)$ such factors for a variable and each is of length at most 2. Thus for a fixed set of resources the size of the equation increases by $\mathcal{O}(n)$. Summing over possible sets of resources (which is of constant size) yields the claim.

▶ **Lemma 8.** *During* TrEqSat *the length of the trace equation is $\mathcal{O}(n)$.*

4.4.2 Weight of solutions

To guarantee the termination, we show that all subprocedures decrease the (appropriately defined) weight of a solution. This weight is in fact defined with respect to the original solution: The B-solution σ corresponds to some solution of the input equation, as letters of B correspond to some traces in the original equation. To keep track of those traces we use an A-morphism $\alpha : M(B, \rho, \theta, \mu) \to M(A, \rho_0, \mu_0)$; the idea is that $c \in B$ represents a trace $\alpha(c)$ in $M(A, \rho_0, \mu_0)$. Conceptually, $\alpha(\sigma(W))$ is the corresponding solution of the input equation. We call a pair (σ, α) a *solution* at $(W, B, \mathcal{X}, \rho, \mu)$, where σ is a B-solution. Note that this morphism is a tool of analysis and proof, it is neither computed nor stored anywhere by the algorithm.

Using the morphism we define the *weight* of a solution (α, σ) as $\|\alpha, \sigma\| = \sum_{X \in \mathcal{X}} |\alpha\sigma(X)|$. All subprocedures performed by our algorithm do not increase the weight. In order to ensure that they all decrease some "weight", we take into the account also the weight of the equations and define a weight of a solution (α, σ) at a state V as $(\|\alpha, \sigma\|, \|V\|)$ which is evaluated in lexicographic order. All subprocedures decrease such defined weight. Thus, the path in NFA for a fixed solution is finite and terminates in a final state.

4.4.3 Internal operations

So far all the described operations were performed on the equation and had some influence also on the solutions. However, there are also operations that are needed for the proof but are performed either on the monoid or on the solutions alone, hence they do not affect the equation at all. For this reason we call them *internal*. In essence, we apply them to the equation whenever this is possible.

A constant $a \in B \setminus A$ is *useless* if it does not occur in $\sigma(W)$; it is *useful* otherwise; useless constants are invisible. A variable is *useless* if it does not occur in W. We remove from the monoid all useless constants and variables.

Due to compression we can be left with invisible but useful constants, i.e., such that they occur in $\sigma(W)$ but not in W.

We cannot remove such constants from B, as we deal with all solutions. However, we can replace them with corresponding traces over $M(A, \rho_0, \mu_0)$. The idea is that we replaced $\alpha(c)$ with c too eagerly. We revert this compression. We do not revert the linearization of the trace, though. Thus we lift each letter in $\alpha(c)$ so that it has the same resources as c: we replace every invisible letter c with $(a_1, \rho(c))(a_2, \rho(c)) \cdots (a_\ell, \rho(c))$, where $\alpha(c) = a_1 a_2 \cdots a_\ell$, i.e., with a chain of letters corresponding to the trace compressed into c but lifted into current resources of c. Note that we use letters from $A \subseteq B$, so the procedure is not applicable to letters that it just introduced.

4.4.4 Completeness

The last step to show completeness is an observation that each given subprocedure corresponds to a composition of finitely many substitution and compression transitions: indeed, this is done by mechanical verification.

The completeness, formulated below, easily follows: given an equation with a solution (α, σ) we apply the subprocedures that lead to a final state. By observation above each subprocedure corresponds to a short path in the NFA. The guarantee on the size of the states follows from Lemma 8. Finally, we cannot iterate forever, as each subprocedure decreases the weight of the solution at a state.

▶ **Lemma 9.** *There is constant $\kappa'' \geq 1$ (depending on $\mathfrak{R}$ but independent of n) such that for all states V, if $\|V\| \leq \kappa'' \cdot n$ and V has a solution (α, σ), then there exists a path to final state labeled with $h_1, h_2, \ldots, h_k$ such that*

$$\sigma = h_1 h_2 \cdots h_k.$$

References

1 Peter R.J. Asveld. Controlled iteration grammars and full hyper-AFL's. *Information and Control*, 34(3):248–269, 1977.

2 Jean Berstel and Dominique Perrin. The origins of combinatorics on words. *Eur. J. Comb.*, 28:996–1022, 2007.

3 Montserrat Casals and Ilya Kazachkov. On systems of equations over partially commutative groups. *Memoirs Amer. Math. Soc.*, 212:1–153, 2011.

4 Laura Ciobanu, Volker Diekert, and Murray Elder. Solution sets for equations over free groups are EDT0L languages. In *ICALP 2015, Proceedings*, volume 9135 of *LNCS*, pages 134–145. Springer, 2015.

5 Volker Diekert, Claudio Gutiérrez, and Christian Hagenah. The existential theory of equations with rational constraints in free groups is PSPACE-complete. *Information and Computation*, 202:105–140, 2005.

6 Volker Diekert, Artur Jeż, and Manfred Kufleitner. Solutions of Word Equations over Partially Commutative Structures. *ArXiv e-prints*, 2016. `arXiv:1603.02966`.

7 Volker Diekert and Markus Lohrey. Word equations over graph products. *Int. J. Algebra Comput.*, 18:493–533, 2008.

8 Volker Diekert and Anca Muscholl. Solvability of equations in free partially commutative groups is decidable. *Int. J. Algebra Comput.*, 16:1047–1070, 2006.

9 Julien Ferté, Nathalie Marin, and Géraud Sénizergues. Word-mappings of level 2. *Theory Comput. Syst.*, 54:111–148, 2014.

10 Ju. I. Hmelevskiĭ. *Equations in Free Semigroups*. Number 107 in Proc. Steklov Institute of Mathematics. American Mathematical Society, 1976. Translated from the Russian original: Trudy Mat. Inst. Steklov. 107, 1971.

11 Sanjay Jain, Alexei Miasnikov, and Frank Stephan. The complexity of verbal languages over groups. In *LICS 2012, Proceedings*, pages 405–414. IEEE Computer Society, 2012.

12 Artur Jeż. Recompression: a simple and powerful technique for word equations. *J. ACM*, 63:1–51, 2016. Conference version at STACS 2013.

13 O. Kharlampovich and A. Myasnikov. Elementary theory of free non-abelian groups. *J. of Algebra*, 302:451–552, 2006.

14 Roger C. Lyndon and Marcel-Paul Schützenberger. The equation $a^M = b^N c^P$ in a free group. *Michigan Math. J.*, 9:289–298, 1962.

15 Gennadiĭ S. Makanin. The problem of solvability of equations in a free semigroup. *Math. Sbornik*, 103:147–236, 1977. English transl. in Math. USSR Sbornik 32 (1977).

16 Gennadiĭ S. Makanin. Equations in a free group. *Izv. Akad. Nauk SSR*, Ser. Math. 46:1199–1273, 1983. English transl. in Math. USSR Izv. 21 (1983).

17 Yuri Matiyasevich. Some decision problems for traces. In *LFCS 1997, Proceedings*, volume 1234 of *LNCS*, pages 248–257. Springer, 1997.

18 Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.

19 Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. *J. ACM*, 51:483–496, 2004.

20 Wojciech Plandowski. An efficient algorithm for solving word equations. In *STOC*, pages 467–476. ACM, 2006.

21 Wojciech Plandowski and Wojciech Rytter. Application of Lempel-Ziv encodings to the solution of word equations. In *ICALP 1998, Proceedings*, volume 1443 of *LNCS*, pages 731–742. Springer, 1998.

22 Alexander A. Razborov. *On Systems of Equations in Free Groups*. PhD thesis, Steklov Institute of Mathematics, 1987. In Russian.

23 Zlil Sela. Diophantine geometry over groups VIII: Stability. *Ann. of Math.*, 177:787–868, 2013.

24 Daniel Wise. *From Riches to Raags: 3-Manifolds, Right-Angled Artin Groups, and Cubical Geometry*. American Mathematical Society, 2012.