Finding All Solutions of Equations in Free Groups and Monoids with Involution

Volker Diekert

Institut für Formale Methoden der Informatik, University of Stuttgart, Germany

Artur Jeż

Institute of Computer Science, University of Wroclaw, Poland

Wojciech Plandowski

Institute of Informatics, University of Warsaw, Poland

Abstract

The aim of this paper is to present a PSPACE algorithm which yields a finite graph of exponential size and which describes the set of all solutions of equations in free groups as well as the set of all solutions of equations in free monoids with involution in the presence of rational constraints. This became possible due to the recently invented *recompression* technique of the second author.

While the recompression technique was successfully applied for pure word equations without involution or rational constraints it could not be used as a black box for free groups (even without rational constraints). Actually, the presence of an involution (inverse elements) and rational constraints complicates the situation and some additional analysis is necessary. Still, the recompression technique is general enough to accommodate both extensions. In the end, it simplifies proofs that satisfiability of word equations is in PSPACE (Plandowski 1999) and the corresponding result for equations in free groups with rational constraints (Diekert, Hagenah and Gutiérrez 2001). As a by-product we obtain a direct proof that it is decidable in PSPACE whether or not the solution set is finite. [1]

Keywords: word equations, equations in free groups, string unification, regular constraints

Introduction

A word equation is a simple object. It consists of a pair (U, V) of words over constants and variables and a solution is a substitution of the variables by words in constants such that U and V become identical words. The study of word equations has a long tradition. Let *WordEquation* be the problem of deciding whether a given word equation has a solution. It is

[1] A preliminary version of this paper was presented as an invited talk at CSR 2014 in Moscow, June 7-11, 2014.

fairly easy to see that WordEquation reduces to Hilbert's 10th Problem (in Hilbert's famous list presented in 1900 for his address at the International Congress of Mathematicians). Hence in the mid 1960s the Russian school of mathematics outlined the roadmap to prove undecidability of Hilbert 10th Problem via undecidability of WordEquation. The program failed in the sense that Matiyasevich proved Hilbert's 10th Problem to be undecidable in 1970, but by a completely different method, which employed number theory. The missing piece in the proof of the undecidability of Hilbert's 10th Problem was based on methods due to Robinson, Davis, and Putnam [22]. On the other hand, in 1977 Makanin showed in a seminal paper [18] that WordEquation is decidable! The program went a different way, but its outcome were two major achievements in mathematics. Makanin's algorithm became famous since it settled a long standing problem and also because his algorithm had an extremely complex termination proof. In fact, his paper showed that the existential theory of equations in free monoids is decidable. This is close to the borderline of decidability as already the $\forall\exists^3$ positive theory of free monoids is undecidable [8]. Furthermore Makanin extended his results to free groups and showed that the existential and positive theories in free groups are decidable [19, 20]. Later Razborov was able in [28] (partly shown also in [29]) to describe the set of all solutions for systems of equations in free groups (see also [15] for a description of Razborov's work). This line of decidability results culminated in the proof of Tarski's conjectures by Kharlampovich and Myasnikov. In a series of papers ending in [16] they showed: 1.) The elementary theory of free groups is decidable. 2.) Free non-abelian groups are elementary equivalent. The second result has also been shown by Sela [32], independently.

Another branch of research was to extend Makanin's result to more general algebraic structures including free partially commutative monoids [21, 6], free partially commutative monoids with involution, graph groups (also known as right-angled Artin groups) [7], graph products [5], and hyperbolic groups [30, 2]. In all these cases the existential theory of equations is decidable. Proofs used the notion of *equation with rational constraints*, which was first developed in the habilitation of Schulz [31]. The concept of equation with rational constraints is used also throughout the present paper.

In parallel to these developments there were drastic improvements in the complexity of deciding WordEquation. It is fairly easy to see that the problem is NP-hard. Thus, NP is a lower bound. First estimations for the time complexity on Makanin's algorithm for free monoids led to a tower of several exponentials, but it was lowered over time to EXPSPACE in [10]. On the the other hand it was shown in [17] that Makanin's algorithm to solve equations in free groups is not primitive recursive. (Already in the mid 1990 this statement was somehow puzzling and counter-intuitive, as it suggested a strange crossing of complexities: The existential theory in free monoids seemed to be easier than the one in free groups, whereas it was already known at that time that the positive theory in free monoids is undecidable, but decidable in free groups.) The next important step was done by Plandowski and Rytter, whose approach [27] was the first essentially different than Makanin's original solution. The main idea was to apply compression to WordEquation and the result was that the length-minimal solution of a word equation compresses well, in the sense that Lempel-Ziv encoding, which is a popular practical standard of compression, of such a solution is expo-

nentially smaller than the solution itself (if the solution is at least exponential in the length of the equation). This yielded an $\mathsf{npoly}(n, \log N)$ algorithm for length n WordEquation with a length-minimal solution of length N, note that at that time the only available bound on N was the triply exponential bound by Makanin. Still, this result prompted Plandowski and Rytter to formulate a (still open) conjecture that WordEquation is NP-complete.

Soon after a doubly exponential bound on N was shown by Plandowski [23], this bound in particular used the idea of representing the solutions in a compressed form (in fact, the equation as well is kept in a compressed form) as well as employing a novel type of factorisations. Exploiting better the interplay between factorisations and compression Plandowski showed that WordEquation is in PSPACE, i.e., it can be solved in polynomial space and exponential time [24]. His method was quite different from Makanin's approach and more symmetric. Furthermore, it could be also used to generate all solutions of a given word equation [25], however, this required nontrivial extensions of the original method.

Using Plandowski's method Gutiérrez showed that satisfiability of equations in free groups is in PSPACE [11], which led Diekert, Hagenah and Gutiérrez to the result that the existential theory of equations with rational constraints in free groups is PSPACE-complete [4]. Without constraints PSPACE is still the best upper bound, although the existential theories for equations in free monoids (with involution) and free groups are believed to be NP-complete. Since this proof generalized Plandowski's satisfiability result [24], it is tempting to also extend the generator of all solutions [25]. Indeed, Plandowski claimed that his method applies also to free groups with rational constraints, but he found a gap in his generalization [26].

However in 2013 another substantial progress in solving word equations was done due to a powerful recompression technique by Jeż [14]. His new proof that WordEquation is in PSPACE simplified the existing proofs drastically. In particular, this approach could be used to describe the set of all solutions rather easily, so the previous construction of Plandowski [25] was simplified as well.

What was missing however was the extension to include free monoids with involution and therefore free groups and another missing block was the the presence of rational constraints. Both extensions are the subject of the present paper.

Outline

We first follow the approach of [4] how to (bijectively) transform (in polynomial time) the set of all solutions of an equation with rational constraints over a free group into a set of all solutions of an equation with regular constraints over a free monoid with involution, see Section 1.2. Starting at that point in Section 2 we formulate the main technical claim of the paper: (effective) existence of a procedure that transforms equations over the free monoid and (roughly speaking) keeps the set of solutions as well as does not increase the size of the word equation; in particular in this section we make all the intuitive statements precise. Moreover, we show how this procedure can be used to create a PSPACE-transducer which produces a finite graph (of exponential size) describing all solutions and which is nonempty if and only if the equation has at least one solution. Moreover, the graph also encodes whether or not there are finitely many solutions, only. The technique of recompression simplifies

thereby [4] and it yields the important new feature that we can describe all solutions. The next Section 3 is devoted to a proof of the statements from Section 2: we formalize, what type of factors we compress, see Section 3.2, and how to compress them. Lastly, in Section 4, we show that a slight modification of our method also provides an $\mathcal{O}(n^3 \log N)$ upper bound on the (non-deterministic) running time in case of a free group, where n is a size of the equation and N the length of the length-minimal solution.

1. Preliminaries

As already mentioned, the general plan is to reduce the problem of word equation with regular constraints over free group to the problem of word equation with regular constraints over a free monoid with an involution and give an algorithm for the latter problem. In this section we first introduce all notions regarding the word equation over the free monoid, see Section 1.1, and only afterwards the similar notions for a free group together with the reduction of the latter scenario to the former one, see Section 1.2.

1.1. Word equations over a free monoid with involution

Let A and Ω be two finite disjoint sets, called *the alphabet of constants* and *the alphabet of variables* (or *unknowns*), respectively. For the purpose of this paper A and Ω are endowed with an *involution*, which is a mapping $^-$ such that $\overline{\overline{x}} = x$ for all elements. In particular, an involution is a bijection. If involution is defined for a monoid, then we additionally require that $\overline{xy} = \overline{y}\,\overline{x}$ for all its elements x, y. This applies in particular to a free monoid A^* over a set with involution: For a word $w = a_1 \cdots a_m$ we thus have $\overline{w} = \overline{a_m} \cdots \overline{a_1}$. If $\overline{a} = a$ for all $a \in A$ then $\overline{w}$ simply means to read the word from right-to-left. It is sometimes useful to consider *involution closed sets*, i.e., such that $\overline{S} = S$.

A *word equation* is a pair (U, V) of words over $A \cup \Omega$, often denoted by $U = V$. A *solution* σ of a word equation $U = V$ is a substitution σ of unknowns in Ω by words over constants, such that the replacement of unknowns by the substituted words in U and in V give the same word. Moreover, as we work with involutions we additionally demand that the solution satisfies $\sigma(\overline{X}) = \overline{\sigma(X)}$ for all $X \in \Omega$. If an equation does not have simultaneous occurrences of X and $\overline{X}$ where $X \neq \overline{X}$ then this additional requirement is vacuous. A solution is *non-empty*, if $\sigma(X) \neq \epsilon$ for every variable X such that X or $\overline{X}$ occurs in the equation. During the proof we will consider only non-empty solutions. This is non-restrictive, as we can always non-deterministically guess the variables that are assigned ϵ by a solution and remove such variables from the equation. On the other hand, it is useful to assume that a solution assigns ϵ to each variable X such that neither X, nor $\overline{X}$ occur in the equation: during the algorithm we remove the variables that are assigned ϵ in the solution. Nevertheless, we need to know the substitution for such X, as we create the set of all solutions by backtracking.

Example 1. Let $\Omega = \left\{X, Y, \overline{X}, \overline{Y}\right\}$ and $A = \{a, b\}$ with $b = \overline{a}$. Then $XabY = YbaX$ behaves as a word equation without involution One of its solutions is the substitution $\sigma(X) = bab$, $\sigma(Y) = babab$. Under this substitution we have $\sigma(X)ab\sigma(Y) = bababbabab = \sigma(Y)ba\sigma(X)$. It can be proved that the solution set of the equation $XabY = YbaX$ is closely related to Sturmian words [13].

The notion of word equation immediately generalizes to a system of word equations $(U_1, V_1), \ldots, (U_s, V_s)$. In this case a solution σ must satisfy all (U_i, V_i) simultaneously. However, such a system can be reduced to a single equation $(U_1 a \cdots U_s a U_1 b \cdots U_s b, V_1 a \cdots V_s a V_1 b \cdots V_s b)$ where a, b are fresh constants with $a \neq b$. To see the correctness of the reduction consider that $U_1 a \cdots U_s$ and $U_1 b \cdots U_s$ always have the same length, the same applies to $V_1 a \cdots V_s$ and $V_1 b \cdots V_s$, thus this equation is equivalent to two equations $(U_1 a \cdots U_s, V_1 a \cdots V_s)$ and $(U_1 b \cdots U_s, V_1 b \cdots V_s)$. Then any solution must align all as (bs) between U_i and U_{i+1} against the corresponding as (bs) between V_i and V_{i+1}: otherwise a corresponding a and b from two equations are aligned against the same constant from one of the original equations. Note, this reduction remains valid when additionally regular constraints are introduced, such constraints are properly defined below.

Lastly, we always assume that the involution on Ω is without fixed points: otherwise for a variable X such that $\overline{X} = X$ we can introduce a fresh variable X', set $\overline{X} = X'$ and add an equation $X = X'$, which ensures that $\sigma(X) = \overline{\sigma(X)}$. In this way we can avoid some case distinctions.

Constraints

Let $\mathcal{C}$ be a class of formal languages, then a system of *word equations with constraints in* $\mathcal{C}$ is given by a finite list $(U_i, V_i)_i$ of word equations and a finite list of constraints of type $X \in L$ (resp. $X \notin L$) where $X \in \Omega$ and $L \subseteq A^*$ with $L \in \mathcal{C}$. For a solution we now additionally demand that $\sigma(X) \in L$ (resp. $\sigma(X) \notin L$) for all constraints.

Here, we focus on rational and recognizable (or regular) constraints and we assume that the reader is familiar with basic facts in formal language theory. The classes of rational and recognizable subsets are defined for every monoid M [9], and they are incomparable, in general. *Rational* sets (or languages) are defined inductively as follows.

- All finite subsets of M are rational.
- If $L_1, L_2 \subseteq M$ are rational, then the union $L_1 \cup L_2$, the concatenation $L_1 \cdot L_2$, and the generated submonoid L_1^* are rational.

A subset $L \subseteq M$ is called *recognizable*, if there is a homomorphism ρ to some finite monoid E such that $L = \rho^{-1}\rho(L)$. We also say that ρ (or E) *recognizes* L in this case. Kleene's Theorem states that in finitely generated free monoids both classes coincide, and we follow the usual convention to call a rational subset of a free monoid *regular*. If M is generated by some finite set $\Gamma \subseteq M$ (as it always the case in this paper) then every rational set is the image of a regular set L under the canonical homomorphism from Γ^* onto M; and every recognizable set of M is rational. (These statements are trivial consequences of Kleene's Theorem.) Therefore, throughout we assume that a rational (or regular) language is specified by a nondeterministic finite automaton, *NFA* for short.

Consider a list of k regular languages $L_i \subseteq A^*$ each of them being specified by some NFA with m_i states. The disjoint union of these automata yields a single NFA with $m = m_1 + \cdots + m_k$ states which accepts all L_i by choosing appropriate initial and final sets for each L_i; we may assume that the NFA has state set $\{1, \ldots m\}$. Then each constant $a \in A$

defines a Boolean $m \times m$ matrix $\tau(a)$ where the entry (p, q) is 1 if (p, a, q) is a transition and 0 otherwise. This yields a homomorphism $\tau : A^* \to \mathbb{B}^{m\times m}$ such that τ recognizes L_i for all $1 \leq i \leq k$. Moreover, for each i there is a row vector $I_i \in \mathbb{B}^{1\times n}$ and a column vector $F_i \in \mathbb{B}^{n\times 1}$ such that we have $w \in L_i$ if and only if $I_i \cdot \tau(w) \cdot F_i = 1$.

For a matrix P we let P^T be its transposition. There is no reason that $\tau(\overline{a}) = \tau(a)^T$, hence τ is not necessarily a homomorphism which respects the involution. So, as done in [4], we let $\mathbb{M}_{2m} \subseteq \mathbb{B}^{2m\times 2m}$ denote the following monoid with involution:

$$\mathbb{M}_{2m} = \left\{ \begin{pmatrix} P & 0 \\ 0 & Q \end{pmatrix} \;\middle|\; P, Q \in \mathbb{B}^{m\times m} \right\} \text{ with } \overline{\begin{pmatrix} P & 0 \\ 0 & Q \end{pmatrix}} = \begin{pmatrix} Q^T & 0 \\ 0 & P^T \end{pmatrix}.$$

Define $\rho(a) = \left(\begin{smallmatrix} \tau(a) & 0 \\ 0 & \tau(\overline{a})^T \end{smallmatrix}\right)$. Then the homomorphism $\rho : A^* \to \mathbb{M}_{2m}$ respects the involution. Moreover ρ recognizes all L_i and $\overline{L_i} = \{\overline{w} \mid w \in L_i\}$.

Consider regular constraints $X \in L$ and $X \notin L'$. As ρ recognises both L and L', the conditions $\sigma(X) \in L$ and $\sigma(X) \notin L'$ are equivalent to $\rho(\sigma(X)) \in \rho(L)$ and $\rho(\sigma(X)) \notin \rho(L')$. As the image of ρ is a subset of $\mathbb{M}_{2m}$, there are only finitely many elements in it. Thus all regular constraints on X boil down to restrictions of possible values of $\rho(\sigma(X))$. To be more precise, if all positive constraints on X are $(L_i)_{i\in I}$ and all negative are $(L'_i)_{i\in I'}$, all those constraints are equivalent to

$$\rho(\sigma(X)) \in \bigcap_{i\in I} \rho(L_i) \cap \bigcap_{i\in I'} (\mathbb{M}_{2m} \setminus \rho(L'_i)) \ .$$

Thus, as a preprocessing step our algorithm guesses the $\rho(\sigma(X))$, which we shall shortly denote as $\rho(X)$, moreover this guess needs to satisfy

- $\rho(\overline{X}) = \overline{\rho(X)}$
- $\rho(X) \in \rho(L)$ for each positive constraint L on X;
- $\rho(X) \notin \rho(L')$ for each negative constraint L' on X.

In the following we are interested only in solutions for which $\rho(\sigma(X)) = \rho(X)$. Note that, as ρ is a function, each solution of the original system corresponds to a solution for exactly one such guess, thus we can focus on generating the solutions for this restricted problem.

We now give a precise definition of the main problem we are considering in the rest of the paper:

Definition 1. An *equation E with constraints* is a tuple $E = (A, \Omega, \rho; U = V)$ containing the following items:

- An alphabet of constants with involution A.
- An alphabet of variables with involution without fixed points Ω.
- A mapping $\rho : A \cup \Omega \to \mathbb{M}_{2m}$ such that $\overline{\sigma(x)} = \sigma(\overline{x})$ for all $x \in A \cup \Omega$.
- The word equation $U = V$ where $U, V \in (A \cup \Omega)^*$.

A *solution* of E is a homomorphism $\sigma : (A \cup \Omega)^* \to A^*$ leaving the constants from A invariant such that the following conditions are satisfied:

$$\begin{aligned} \sigma(U) &= \sigma(V)\,, \\ \overline{\sigma(X)} &= \sigma(\overline{X}) \quad \text{for all} \quad X \in \Omega, \\ \rho(\sigma(X)) &= \rho(X) \quad \text{for all} \quad X \in \Omega. \end{aligned}$$

The *input size* of E is given by $\|E\| = |A| + |\Omega| + |UV| + m$.

In the following, when this does not cause a confusion, we denote both the size of the instance and the length of the equation by n. Note that we can always increase the size of the equation by repeating it several times.

The measure of size of the equation is accurate enough with respect to polynomial time and/or space. For example note that if an NFA has m states then the number of transitions is bounded by $m\,|A|$. Note also that $|A|$ can be much larger than the sum over the lengths of the equations plus the sum of the number of states of the NFAs in the lists for the constraints.

As already noted, by a convention, when a variable X and its involution $\overline{X}$ are not present in the equation, each solution assigns ϵ to both X and $\overline{X}$. In particular, this assignment should satisfy the constraint, i.e., $\rho(X) = \rho(\epsilon)$ for each variable not present in the solution. Note that the input equation can have variables that are not present in the equation and have constraints other than $\rho(X) = \rho(\epsilon)$, however, such a situation can be removed by a simple preprocessing: we check (in PSPACE) whether there is a word satisfying the given set of constraints, if not, we reject, if yes then we remove the variable from the instance.

Equations during the algorithm.. During the procedure we will create various other equations and introduce new constants. Still, the original alphabet A never changes and new constants shall represent words in A^*. As a consequence, we will work with equations over $B \cup \Omega$, where B is the smallest alphabet containing A and all constants in $UV\overline{UV}$. We shall call such B the *alphabet of* (U, V). Note that $|B| \leq |A| + 2\,|UV|$ and we therefore we can ignore $|B|$ for the complexity.

Ideally, a solution of (U, V) assigns to variables words over the alphabet of (U, V), call it B. However, as our algorithm transforms the equations and solutions, it is sometimes more convenient to allow also solutions that assign words from some $B' \supset B$. A solution is *simple* if it uses only constants from B, by default we consider simple solutions. Whenever we consider a non-simple solution, we explicitly give the alphabet over which this is a solution.

To track the meaning of constants outside A, we additionally require that a solution (over an alphabet B') supplies some homomorphism $h : B' \mapsto A^*$, which is constant on A and compatible with ρ, in the sense that $\rho(b) = \rho(h(b))$ for all $b \in B$. (Due to its nature, we also assume that $h(b)$ contains at least two constants for $b \in B' \setminus A$.) Thus, in the following, a solution is a pair (σ, h). In particular, given an equation (U, V) the $h(\sigma(U))$ corresponds to a solution of the original equation. Note, that h is a homomorphism with respect to the involution, i.e., we assume that $h(\overline{a}) = \overline{h(a)}$.

A *weight* of a solution (σ, h) of an an equation (U, V) is

$$\mathrm{w}(\sigma, h) = |U| + |V| + \sum_{X \in \Omega} |UV|_X \, |h(\sigma(X))| \quad , \tag{1}$$

where $|UV|_X$ denotes the number of occurrences of X in U and V together. The main property of such defined weight is that it decreases during the run of the algorithm, using this property we shall guarantee a termination of the algorithm: each next equation in the sequence will have a smaller weight, which ensures that we do not cycle.

Given a non-simple solution (σ, h) we can replace all constants $c \notin B$ (where B is the alphabet of the equation) in all $\sigma(X)$ by $h(c)$ (note, that as $\rho(c) = \rho(h(c))$, the $\rho(X)$ is preserved in this way). This process is called a *simplification* of a solution and the obtained substitution σ' is a *simplification* of σ. It is easy to show that σ' is a solution and that $h(\sigma'(U)) = h(\sigma(U))$, so in some sense both σ and σ' represent the same solution of the original equation. Lastly, σ and σ' have the same weight, see Lemma 1. Thus, in some sense we can always simplify the solution.

Lemma 1. *Suppose that (σ, h) is a solution of the equation (U, V). Then the simplification (σ', h) of (σ, h) is also a solution of (U, V), $h(\sigma'(U)) = h(\sigma(U))$ and $\mathrm{w}(\sigma', h) = \mathrm{w}(\sigma, h)$.*

Proof. Let B be the alphabet of the equation and B' the alphabet of the solution σ. Consider any constant $b \in B' \setminus B$. As it does not occur in the equation, all its occurrences in $\sigma(U)$ and $\sigma(V)$ come from the variables, i.e., from some $\sigma(X)$. Then replacing all occurrences of b in each $\sigma(X)$ by the same string w preserves the equality of $\sigma(U) = \sigma(V)$, thus σ' is also a solution. Since we replace some constants b with $h(b)$ (and $h \circ h = h$), clearly $h(\sigma(X)) = h(\sigma'(X))$ for each variable, in particular, the weight contributed by each variable occurrence does not change. Furthermore, as $\rho(c) = \rho(h(c))$ we have that $\rho(\sigma(X)) = \rho(\sigma'(X))$. Thus, $h(\sigma'(U)) = h(\sigma(U))$ and $\mathrm{w}(\sigma', h) = \mathrm{w}(\sigma, h)$, as claimed. □

As a final note observe that h is a technical tool used in the analysis, it is not stored, nor transformed by the algorithm, nor it is used in the graph representation of all solutions.

1.2. Word equations with rational constraints over free groups.

By $F(\Gamma)$ we denote the free group over a finite set Γ. We let $A = \Gamma \cup \Gamma^{-1}$. Set also $\overline{x} = x^{-1}$ for all $x \in F(\Gamma)$. Thus, in (free) groups we identify x^{-1} and $\overline{x}$. By a classical result of Benois [1] rational subsets of $F(\Gamma)$ form an effective Boolean algebra. That is: if L is rational and specified by some NFA then $F(\Gamma) \setminus L$ is rational; and we can effectively find the corresponding NFA. There might be an exponential blow-up in the NFA size, though. This is the main reason to allow negative constraints $X \notin L$, so we can avoid explicit complementation.

Proposition 1 ([4])**.** *Let $F(\Gamma)$ be a free group and $A = \Gamma \cup \Gamma^{-1}$ be the corresponding set with involution as above. There is a polynomial time transformation which takes as input a system $\mathcal{S}$ of equations (and inequalities) with rational constraints over $F(\Gamma)$ and outputs a word equation with regular constraints $\mathcal{S}'$ over A which is solvable if and only if $\mathcal{S}'$ is solvable in $F(\Gamma)$.*

More precisely, let $\varphi : A^* \to F(\Gamma)$ be the canonical morphism of the free monoid with involution A^* onto the free group $F(\Gamma)$. Then the set of all solutions for $\mathcal{S}'$ is bijectively mapped via $\sigma' \mapsto \varphi \circ \sigma'$ onto the set of all solutions of $\mathcal{S}$.

Proposition 1 in particular shows that the description of all solutions of a system of equations and inequalities (with rational constraints) over a free group can be efficiently reduced to solving the corresponding task for word equations with regular constraints in a free monoid with involution. For convenience of the reader let us remark that the proof of Proposition 1 is fairly straightforward. It is based on the fact that $XYZ = 1$ in $F(\Gamma)$ is equivalent with the existence of words $P, Q, R \in A^*$ such that $X = P\overline{Q}$, $Y = Q\overline{R}$, and $Z = R\overline{P}$. Indeed, if $XYZ = 1$ in $F(\Gamma)$ then we can represent X, Y, and Z by reduced words and the existence of P, Q, R follows because $F(\Gamma)$ is a free group. The other direction is trivial and holds for non reduced words as well.

Input size.

The input size for the reduction is given by the sum over the lengths of the equations and inequalities plus the size of Γ plus the sum of the number of states of the NFAs in the lists for the constraints. As in the case of word equations over free monoid, the measure is accurate enough with respect to polynomial time and or space. Note that $|\Gamma|$ can be much larger than the sum over the lengths of the equations and inequalities plus the sum of the number of states of the NFAs in the lists for the constraints. We encode $X \neq 1$ by a rational constraint, which introduces an NFA with $2\,|\Gamma| + 1$ states. Since $|\Gamma|$ is part of the input, this does not cause any problem. The output size remains at most quadratic in the input size.

1.3. Existential theory for free groups

We can easily extend the algorithm for word equations over free groups with rational constraints to the existential theory of free groups with rational constraints. As a first step note that we can eliminate the disjunction by non-deterministic guesses. Secondly, as the singleton $\{1\} \subseteq F(\Gamma)$ is, by definition, rational, the set $F(\Gamma) \setminus \{1\}$ is rational, too. Therefore an inequality $U \neq V$ can be handled by a new fresh variable X and writing $U = XV \;\&\; X \in F(\Gamma) \setminus \{1\}$ instead of $U \neq V$.

1.4. Linear Diophantine systems

We shall consider linear Diophantine systems with solutions over natural numbers. Formally, such a system is given by an $m \times n$ matrix A with coefficients in $\mathbb{Z}$ and an $m \times 1$ vector $b \in \mathbb{Z}^m$. We write $Ax = b$ and its set of solutions is given by the set $\{x \in \mathbb{N}^n \mid Ax = b\}$. We say that $Ax = b$ is *satisfiable* over $\mathbb{N}$ if the set of solutions is non-empty. Note that while we could also allow inequalities, a system of inequalities $Ax \geq b$ can be reduced to equalities by introducing fresh variables y and rewriting the system as $Ax - y = b$. Looking for solutions in $\mathbb{N}^n$ makes the problem NP-hard. Actually, we use the following well-known proposition.

Proposition 2. *The following two problems are* NP*-complete.*
Input. *$Ax = b$ where $A \in \mathbb{Z}^{n \times n}$ and $b \in \mathbb{Z}^{n \times 1}$ and coefficients are written in binary.*
Question 1. *Is the set $\{x \in \mathbb{N}^n \mid Ax = b\}$ non-empty?*
Question 2. *Is the set $\{x \in \mathbb{N}^n \mid Ax = b\}$ infinite?*

Proof. The NP-completeness of the first problem is standard, see e.g., [12]. It can be reduced to the second problem by adding an equation $y - z = 0$ where y, z are fresh variables. A possible reduction of the second problem to satisfiability is as follows. Given an equation $Ax = b$ we create a system $Ax = b \;\&\; Ax' = b$, where $x = (x_1, \ldots, x_n)$ and $x' = (x'_1, \ldots, x'_n)$ use disjoint sets of variables. Then we add equations $x' = x + y$ where y uses fresh variables. This guarantees that $x' \geq x$. Finally, we add an equation $x'_1 + \cdots + x'_n = x_1 + \cdots + x_n + z + 1$. This guarantees that $x' > x$, in the sense that at least one of the inequalities $x'_i \geq x_i$ is strict. If the new system is satisfiable then $Ax = b$ has infinitely many solutions $x + k(x' - x)$ with $k \in \mathbb{N}$. Conversely, if $Ax = b$ has infinitely many solutions then there must exist solutions x and x' with $x < x'$ due to Dickson's Lemma [3], and they satisfy the created system. □

2. Graph representation of all solutions

In this section we give an overview of the graph representation of all solutions and the way such a representation is generated as well as a detailed description of the graph representation of all solution of a word equation with constraints. This description is devised so that it is (together with Section 1) a citable reference, in particular, it is supposed to be usable without reading the actual construction and the proof of its correctness. It will include all the necessary definitions, though. The actual construction and the proof of correctness is given in Section 3.

2.1. Transforming the equation

By an *operator* we denote a function that transforms substitutions (for variables). All our operators have simple description: $\sigma'(X)$ is usually obtained from $\sigma(X)$ by morphisms, appending/prepending constants, etc. In particular, they have a polynomial description. We usually denote them by φ and their applications by $\varphi[\sigma]$.

Recall that the instance size is n, so in particular the input equation is of length at most n and has at most n variables.

Definition 2. A word equation (U, V) with constraints is *proper* if

- in total U and V have at most cn^2 occurrences of constants;
- in total U and V have at most n occurrences of variables;
- there is a homomorphism $h : B \mapsto A^+$ that is compatible with ρ, that is $\rho(h(b)) = \rho(b)$ for each $b \in B$, where B is the alphabet of (U, V) and A is the alphabet of the input equation.

A possible constant is $c = 16$ as we will see later.

Concerning the homomorphism h, note that we do not want to consider equations containing constants that cannot represent strings in the input alphabet. For the input equation we may assume $A = B$ and therefore we can take h as the identity. In particular, the input equation is proper. Note that it is essential that h is also a homomorphism for the involution.

The main technical result of the paper states that:

Lemma 2. *Suppose that (U_0, V_0) is a proper equation with $|U_0| > 1$ or $|V_0| > 1$ over an alphabet B_0 which has a simple solution (σ_0, h_0). Then there exists a proper equation (U_1, V_1) over an alphabet B_1 and a family of operators Φ such that*

- *There is $\varphi \in \Phi$ and a solution (σ_1', h_1) of (U_1, V_1) over $B_0 \cup B_1$ such that*
  - $\sigma_0 = \varphi[\sigma_1']$
  - *σ_1 is a simplification of σ_1'*
  - $h_0(\sigma_0(U_0)) = h_1(\sigma_1(U_1)) = h_1(\sigma_1'(U_1))$.

  Furthermore, $\mathrm{w}(\sigma_0, h_0) > \mathrm{w}(\sigma_1, h_1) = \mathrm{w}(\sigma_1', h_1)$.
- *If (σ_1', h_1') is a solution of (U_1, V_1) (over an arbitrary alphabet) and $\varphi \in \Phi$ then (σ_0', h_0') is a solution of (U_0, V_0), where $\sigma_0' = \varphi[\sigma_1']$ and h_0' is some homomorphism compatible with ρ.*
- *Φ has a polynomial-size descriptions.*

Given (U_0, V_0), all possible equations (U_1, V_1) (for all possible solutions σ_0) can be produced in PSPACE.

Discussion. The exact definition of allowed families of operators Φ is deferred to Section 2.2, for the time being let us only note that Φ has polynomial description (which can be read from (U_0, V_0) and (U_1, V_1)), may be infinite and its elements can be efficiently listed, (in particular, it can be tested, whether Φ is empty, finite or infinite).

Concerning the difference between σ_1 and σ_1': while we know that $\sigma_0 = \varphi[\sigma_1']$ we cannot guarantee that (σ_1', h_1) is simple, so the claim of the Lemma 2 does not apply to σ_1'. However, when we take a simplification (σ_1, h_1) of σ_1', the claim of Lemma 2 does apply. Moreover, σ_1 and σ_1' represent the same solution $h_1(\sigma_1(U_1)) = h_1(\sigma_1'(U_1))$ of the original equation, so nothing is lost in the simplification. Alternatively, we could impose the condition that the solution (σ_1', h_1) is simple however then we cannot assume that $\sigma_0 = \varphi_1[\sigma_1']$, we can only guarantee that $h_0(\sigma_0(U_0)) = h_1(\sigma_1'(U_1))$. The authors realise, that both possible choices make the details of many proofs more complicated, but in any case this is a technical detail that should not bother the reader.

Getting back to the solutions, an equation in which both U_0 and V_0 have length 1 has easy to describe solutions:

- if U_0, V_0 are the same constant then the equation has exactly one solution, in which every variable is assigned ϵ (recall our convention that a variable not present in the equation is assigned ϵ);
- if U_0, V_0 are both variables, say X and Y, then if $\rho(X) \neq \rho(Y)$ then there is no solution, otherwise any σ that assigns ϵ to other variables and w to X, Y, where $\rho(w) = \rho(X)$, is a solution;

- if U_0 is a constant and V_0 a variable, say a and X, then if $\rho(a) \neq \rho(X)$ then there is no solution, otherwise there is a unique solution, which assigns a to X and ϵ to all other variables.

In this way all solutions of the input equation (U, V) are obtained by a path from (U, V) to some satisfiable (U_i, V_i) satisfying $|U_i| = |V_i| = 1$ and the solution of (U, V) is a composition of operators from the families on the path applied to the solution of (U_i, V_i). Note that there may be several ways to obtain the same solution, using different paths in the graph.

2.2. Construction of the solution graph

Using Lemma 2 one can construct in PSPACE a graph like representation of all solutions of a given word equation: for the input equation (U, V) we construct a directed graph $\mathcal{G}$ which has nodes labelled with proper equations. Then for such node, say labelled with (U_0, V_0), such that $|U_0| > 1$ or $|V_0| > 1$ we use Lemma 2 to list all equations (U_1, V_1) such that $(U_0, V_0), (U_1, V_1)$ satisfy the claim of Lemma 2 for some solution of (U_0, V_0). For each such equation we put the edge $(U_0, V_0) \to (U_1, V_1)$ and annotate it with the appropriate family of operators Φ. We lastly remove the nodes that are not reachable from the starting node and those that do not have a path to an ending node.

In this way we obtain a finite description of all solution of a word equation with regular constraints.

Theorem 1. *There exists effectively a* PSPACE *transducer working as follows.*
Input. *A word equation with regular constraints over a free monoid with involution.*
Ouput. *A finite graph representation of all solutions of the equation.*

Using Proposition 1 we obtain a similar claim for word equation with rational constraints over a free group.

Corollary 1. *There exists effectively a* PSPACE *transducer working as follows.*
Input. *A word equation with rational constraints over a free group.*
Ouput. *A finite graph representation of all solutions of the equation.*

2.2.1. Families of operators

Let us now describe the used family of operators. Given an edge $(U, V) \to (U', V')$ the class Φ of operators is defined using:

- A linear Diophantine system of polynomial size in parameters $\{x_X, y_X\}_{X \in \Omega}$.
- A set $\{s_X, s'_X\}_{X \in \Omega}$ of strings, length of string s_X (s'_X) may depend on x_X (y_X, respectively): it may use *one* expression of the form $(ab)^{x_X}$ ($(ab)^{y_X}$, respectively) or a^{x_X} (a^{y_X}, respectively) when $a = b$. The constants a, b need to be from alphabet B, which is the alphabet of constants used in (U, V) plus the constants used by the input equation. Each s_X and s'_X is of polynomial length (we treat $(ab)^{x_X}$ as having description of $\mathcal{O}(1)$ size).

- A set of $E_1, \ldots, E_k$ of strings which may use expressions $(ab)^{x_X}$ and $(ab)^{y_X}$ (or a^{x_X} and a^{y_X}, when $a = b$), similarly to s_X and s'_X, k is of polynomial size and each E_i has polynomial-size description. There are corresponding constants c_{E_1}, c_{E_2}, ..., c_{E_k} that occur in (U', V') but not in (U, V).

Note that the Diophantine system may be empty and some of $\{s_X, s'_X\}_{X\in\Omega}$ may be ϵ or not dependent on parameters. On the other hand, each E_i consists of at least two constants.

Particular operator $\varphi \in \Phi$ corresponds to a solution $\{\ell_X, r_X\}_{X\in\Omega}$ of the system of Diophantine equations. Given such a solution, the corresponding operator φ acts on the substitutions for variables as follows: It first replaces each constant c_{E_i} with strings E_i in which all x_X and y_X are replaced with numbers ℓ_X and r_X. Then it prepends to $\sigma(X)$ the s_X in which parameter x_X is replaced with ℓ_X, then appends s'_X in which parameter y_X is replaced with r_X.

Example 2. Consider an equation $abXZXXXd = XabdYYZ'$ and an equation $c_1Zc_2d = c_1dc_2Z'$. The linear Diophantine system is $x_X + 1 = x_X + 1, 3x_X = 2x_Y$; the strings are $s_X = (ab)^{x_X}, s_Y = (ab)^{x_Y}$ and $s'_X = s'_Y = \epsilon$. There are two expressions $E_1 = (ab)^{x_X+1}$ and $E_2 = (ab)^{3x_X}$, they correspond to c_1 and c_2, respectively. The operator φ replaces c_1 with $(ab)^{\ell_X+1}$ and c_2 with $(ab)^{3\ell_X}$, prepends $(ab)^{\ell_X}$ to the (empty) substitution for X and $(ab)^{\ell_X}$ to the (empty) substitution for Y.

Consider again $abXZXXXd = XabdYYZ'$ and the equation $abXcXbXbXc = XbacYbYc$. The Diophantine system is empty. The strings are $s_X = s_Y = \epsilon, s'_X = s'_Y = b$, $s_Z = s_{Z'} = d$ and $s'_Z = s'_{Z'} = \epsilon$; there is one expression: bd. The operator replaces c with bd, appends b to substitutions for X and Y and d to (empty) substitutions for Z, Z'.

3. Compression step

In this section we describe procedures that show the claim of Lemma 2. In essence, for a word equation (with constraints) (U, V) with a solution σ we want to *compress* the word $\sigma(U)$ directly on the equation, i.e., without the knowledge of the actual solution. In case of the free monoid (without involution) [14], the 'compression' essentially is a replacement of all substrings ab with a single constant c. However, due to the involution (and possibility that $a = b$) the compressions in case of free monoid with involution are more involved: we replace the *ab-blocks*, as defined later in this section, see Definition 4. To do this, we sometimes need to modify the equation (U, V).

The crucial observation is that some of such compressions guarantee that if the compressed equation is proper then so is the obtained one, see Lemma 8. Moreover, each compression step decreases the weight of the corresponding solution, which guarantees a termination of the whole process.

3.1. Transforming solutions and inverse operators.

As we want to describe the set of all solutions, ideally there should be a one-to-one correspondence between the solutions before and after the application of used subprocedures. However, as those subprocedures are non-deterministic and the output depends on

the non-deterministic choices, the situation becomes a little more involved. What we want to guarantee is that no solution is 'lost' in the process and no solution is 'gained': given a solution for some non-deterministic choices we transform the equation into another one, which has a 'corresponding' solution and we know a way to transform this solution back into the original equation. Furthermore, when we transform back in this way any solution of the new equation, we obtain a solution of the original equation, i.e., we do not gain solutions.

As already noticed, to ease the presentation, the solutions of the new equation may use constants outside the alphabet of the new equation. To be more precise, they can 'inherit' some constants from the previous solution and therefore use also the constants that occurred in the previous equation.

Definition 3 (Transforming the solution)**.** Given a (nondeterministic) procedure we say that it *transforms the equation* (U, V) *with a solution* (σ, h) if

- (U, V) is proper and (σ, h) is non-empty;
- Based on the nondeterministic choices and equation (U, V) we can define a family of operators Φ, called the *corresponding family of inverse operators*.
- For some nondeterministic choices the procedure returns an equation (U', V') with a nonempty solution (σ', h') and for some operator $\varphi \in \Phi$ we have $\varphi[\sigma'] = \sigma$. Furthermore, $h(\sigma(U)) = h'(\sigma'(U'))$ and $\mathrm{w}(\sigma', h') \leq \mathrm{w}(\sigma, h)$ and if $(U, V) \neq (U', V')$ then this inequality is in fact strict.

  In such a case we also say that this procedure *transforms* (U, V) *with* (σ, h) *to* (U', V') *with* (σ', h').
- For every equation (U', V') that can be returned by this procedure applied on (U, V) and any its solution (σ', h') and for every operator $\varphi \in \Phi$ the $(\varphi[\sigma'], h_0)$ is a solution of (U, V) for some homomorphism $h_0 : B \mapsto A^+$ compatible with ρ, where B is the alphabet of $\varphi[\sigma'](U)$.

If a procedure transforms every proper equation with every nonempty solution then we say that it *transforms solutions*.

Example 3. Consider a procedure that can replace X with aX (for any constant a) and Y by bY (also for any constant b). Then it transforms the equation $X = Y$ with a solution $\sigma(X) = cc, \sigma(Y) = cc$ and any h: The inverse operator φ prepends a to $\sigma(X)$ and b to $\sigma(Y)$, where a and b are constants that were introduced by the procedure. If (σ', h') is a solution of the obtained equation then $(\varphi[\sigma'], h')$ is a solution of the original equation; note that when $a \neq b$ then the obtained equation does not have solutions at all, but this is fine with the definition. Moreover, for the nondeterministic choice in which we pop c from both X and Y, the obtained equation has a solution $\sigma'(X) = \sigma'(Y) = c$, for which $\sigma = \varphi[\sigma']$.

On the other hand, this procedure does not transform solutions: a solution $\sigma(X) = \sigma(Y) = c$ cannot be transformed, as we do not allow empty solutions. We can modify the procedure, though: it can either replace X with aX or with a, similarly for Y. In our case

$X = Y$ with $\sigma(X) = \sigma(Y) = c$ is transformed into $c = c$ with a solution $\sigma(X) = \sigma(Y) = \epsilon$. It is easy to see that this modified procedure transforms solutions.

Discussion. Note that both the (U', V') and Φ depend on the nondeterministic choices, so it might be that for different choices we can transform (U, V) to (U', V') (with a family Φ') and to (U'', V'') (with a family Φ'').

In many cases, Φ consists of a single operator φ, in such a case we call it the *corresponding inverse operator* furthermore, in some cases φ does not depend on (U, V).

Note that when (U, V) with a (σ, h) is transformed into (U', V') with (σ', h') then the simplification σ'' of σ' (recall that a simplification replaces all constants $b \notin B'$ that do not occur in the equation (U', V') by $h'(b)$ in all $\sigma'(X)$) is also a solution of (U', V') and moreover $h'(\sigma''(U')) = h'(\sigma'(U'))$, so it corresponds to the same original solution of the input equation as $(\sigma'(U'), h')$ see Lemma 1, and so consequently the same as (σ, h), since $h'(\sigma(U')) = h(\sigma(U))$.

Clearly, composition of two operations that transform the equations also transforms the equations (although the description of the family of inverse operators may be more complex).

As a last comment, observe that when we take an arbitrary solution (σ', h') and operator φ then we cannot guarantee that there is some h for which $h'(\varphi[\sigma'](U)) = h'(\sigma'(U'))$: imagine we can replace factor $ab\overline{a}$ with a single constant c while $h'(c) = a'b'$, so there is no way to reasonably define $h(a)$ and $h(b)$. Thus we can take any h for $\varphi[\sigma']$, and we know that one exists by the assumption that (U, V) is proper.

3.2. ab-blocks.

In an earlier paper using the recompression technique [14] there were two types of compression steps: compression of pairs ab, where $a \neq b$ were two different constants, and compression of maximal factor a^ℓ (i.e., ones that cannot be extended to the right, nor left). In both cases, such factors were replaced with a single fresh constant, say c. While the actual replacement was performed only on the equation (U, V) implicitly it was performed also on the solution $\sigma(U)$ as well.

The advantage of such compression steps was that the replaced factors were non-overlapping, in the sense that when we fixed a pair (or block) to be compressed, each constant in a word w belonged to at most one replaced factor.

We would like to use similar compression rules also for the case of monoids with involution, however, one needs to take into the account that when w is replaced with a constant c, then also $\overline{w}$ should be replaced with $\overline{c}$. The situation gets complicated, when some of constants in w are fixed-points for the involution, i.e., $\overline{a} = a$. In the worst case, when $\overline{a} = a$ and $\overline{b} = b$ the occurrences of ab and $\overline{ab} = ba$ are overlapping, so the previous approach no longer directly applies. (Even if we start with a situation such that $a \neq \overline{a}$ for all $a \in A$, as it is the case for free groups, fixed points in larger alphabets are produced during the algorithm.)

Intuitively, when we want to compress ab into a single constant, also $\overline{b}\overline{a}$ needs to be replaced. Furthermore, if factors s and s' are to be replaced and they are overlapping, we should replace their union with a single constant. Lastly, the factors to be replaced naturally

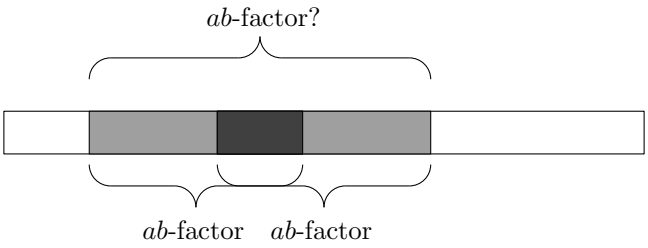

Figure 1: To prove Lemma 3 we need to show that a union of two overlapping ab-factors is also an ab-factor.

fall into *types*, depending on whether the first constant of the factor is a or $\overline{b}$ and the last b or $\overline{a}$.

These intuitions lead to the following definition of ab-blocks (for a fixed pair of constants ab) and their types.

Definition 4. Depending on a and b, *ab-blocks* are

1. If $a = b$ then there are two types of ab-blocks: a^i for $i \geq 2$ and $\overline{a}^i$ for $i \geq 2$.
2. If $a \neq b$, $\overline{a} \neq a$ and $\overline{b} \neq b$ then ab and $\overline{ab} = \overline{b}\overline{a}$ are the two types of ab-blocks.
3. If $a \neq b$, $\overline{a} = a$ and $\overline{b} \neq b$ then ab, $\overline{ab} = \overline{b}a$ and $\overline{b}ab$ are the three types of ab-blocks.
4. If $a \neq b$, $\overline{a} \neq a$ and $\overline{b} = b$ then ab, $\overline{ab} = b\overline{a}$ and $ab\overline{a}$ are the three types of ab-blocks.
5. If $a \neq b$, $\overline{a} = a$ and $\overline{b} = b$ then the $(ba)^i$, $a(ba)^i$, $(ba)^ib$ and $(ab)^i$ (where in each case $i \geq 1$) are the four types of ab-blocks.

An occurrence of an ab-block in a word is an *ab-factor*, it is *maximal*, if it is not contained in any other ab-factor.

The ab-blocks of the first and last kind are called *nontrivial*, as they can have arbitrary large length, the other are called trivial, as they can have at most 3 constants.

Note that for the purpose of this definition when $a = \overline{a}$ we treat a and $\overline{a}$ as the same constant, even if for some syntactic reason we write a and $\overline{a}$.

The following fact is a consequence of the definitions of maximal ab-blocks and shows the correctness of the definition.

Lemma 3. *For any word $w \in B^*$ and two constants $a, b \in B$, maximal ab-factors in w do not overlap.*

Proof. For the proof it is enough to show that when two ab-factors s and s' are overlapping then their union (i.e., the smallest factor that encompasses them both) is also an ab-factor, see Figure 1. Due to case distinction it follows by a case by case analysis according to Definition 4.

If $a = b$ and $a \neq \overline{a}$ then as ab and $\overline{b}\overline{a}$ have no common constants, two overlapping ab-factors are both factors consisting of repetitions of the same constant and so also their union is an ab-factor. If $\overline{a} = a$ then $ab = \overline{b}\overline{a}$ and so the same argument as before applies.

If $a \neq b$, $\overline{a} \neq a$ and $\overline{b} \neq b$ then two overlapping ab-factors need to be the same factor (note that here we do not exclude the case $a = \overline{b}$).

If $a \neq b$, $\overline{a} = a$ and $\overline{b} \neq b$ then consider two different overlapping ab-factors. As all constants in any ab-block are different, if s and s' are of the same type and overlapping then they are in fact the same factor. If the factors ab and $\overline{b}a$ overlap then their union is $\overline{b}ab$, which is also an ab-factor. If factors ab and $\overline{b}ab$ overlap, then the latter contains the former; the same applies to the factors $\overline{b}a$ and $\overline{b}ab$.

If $a \neq b$, $\overline{a} \neq a$ and $\overline{b} = b$ then the analysis is symmetric to the one given above.

In the last case, when $a \neq b$, $\overline{a} = a$ and $\overline{b} = b$ observe that a factor is an ab-factor if and only if it has length at least 2 and consists solely of alternating constants a and b. Thus also a union of two overlapping ab-factors is an ab-factor. □

Given a set of ab-blocks we perform the compression by replacing maximal ab-factors from this set. For consistency, we assume that such a set is involution closed.

Definition 5 (S-reduction)**.** For a fixed ab and an involution-closed set of ab-blocks S the *S-reduction* of the word w is the word w' in which all maximal factors $s \in S$ are replaced by a new constant c_s, where $\overline{c_s} = c_{\overline{s}}$. The inverse operation is an *S-expansion*.

There are the following observations.

- The S-expansion is a function on B^*, using Lemma 3 we obtain that also S-reduction is a function on B^* as well.
- The S-reduction introduces new constants to B, we extend ρ to it in a natural way: $\rho(c_s) = \rho(s)$.
- We let $c = \overline{c}$ if and only if $s = \overline{s}$. In this way constants may become fixed points for the involution. For example, $a\overline{a}$ is an $a\overline{a}$-block for $a \neq \overline{a}$. If $a\overline{a}$ is compressed into c then $c = \overline{c}$.
- It might be that after S-reduction some constant c in the solution is no longer in B (as it was removed from the equation). In such a case the corresponding solution will not be simple, this is described in more detail later on.

Example 4. Consider the case in which $a = \overline{a}$, $b = \overline{b}$ and $d \neq \overline{d}$. Take a word $\overline{d}ababababadbaba\overline{d}abab\overline{d}ad$. Consider the ab, then $\overline{d}(aba)d(ababa)d(baba)\overline{d}(abab)\overline{d}ad$ lists the maximal ab-factors. The factors $aba, ababa, baba, abab$ are replaced with new constants c_1, c_2, c_3, c_4, obtaining $\overline{d}c_1dc_2dc_3\overline{d}c_4\overline{d}ad$. Then $\overline{c}_1 = c_1$, $\overline{c}_2 = c_2$ and $\overline{c}_3 = c_4$.

Consider now ad. The maximal factors are $(\overline{d}a)b(ad)abab(ad)baba(\overline{d}a)bab(\overline{d}ad)$. The factors ad, $\overline{d}a$ and $\overline{d}ad$ are replaced with c_1, c_2, c_3, where $\overline{c}_1 = c_2$ and $\overline{c}_3 = c_3$. In this way we obtain $c_2bc_1ababc_1babac_2babc_3$.

3.3. Crossing and noncrossing factors

Performing the S-reduction is easy, if all maximal factors from S are wholly contained within the equation or within the substitution for a variable: in such a case we perform the S-reduction separately on the equation and on each substitution for a variable (the latter is done implicitly). It looks non-obvious, when part of some factor s is within the substitution for the variable and part in the equation. Let us formalise those notions.

Definition 6. For a word equation (U, V) an ab-factor is *crossing* in a solution σ if it does not come from U (V, respectively), nor from any $\sigma(X)$ for an occurrence of a variable X; ab is *crossing* in a solution σ, if some ab-factor is crossing. Otherwise ab is *non-crossing* in σ.

Note that as $\overline{b}\overline{a}$ is an ab-block, it might be that ab is crossing because of a factor $\overline{b}\overline{a}$.

Example 5. Consider an equation $abXcdYY = XabcdXXX$ and a solution $\sigma(X) = abab$ and $\sigma(Y) = ababab$. Then after the substitution left-hand side is equal to $ab(ab)cd(ababab)(ababab)$. The first factor ab is explicit, second: implicit, the first factor ba is crossing, factor cd is explicit; first factor aba is crossing. In total, cd is non-crossing, ba, bc, da are crossing. If $a \neq \overline{a}$ or $b \neq \overline{b}$ then ab is non-crossing; if $a = \overline{a}$ and $b = \overline{b}$ then ab is crossing (as $\overline{ba} = \overline{a}\overline{b} = ab$ is).

By guessing all $X \in \Omega$ with $\sigma(X) = \epsilon$ (and removing them) we can always assume that $\sigma(X) \neq \epsilon$ for all X. In this case crossing ab's can be alternatively characterized in a more operational manner.

Lemma 4. *Let $\sigma(X) \neq \epsilon$ for all X. Then ab is crossing in σ if and only if one of the following holds:*

- *aX or $\overline{aX} = \overline{X}\overline{a}$, for an unknown X, occurs in U or V and $\sigma(X)$ begins with b (so $\sigma(\overline{X})$ ends with $\overline{b}$)* or
- *Xb or $\overline{Xb} = \overline{b}\,\overline{X}$, for an unknown X, occurs in U or V and $\sigma(X)$ ends with a (so $\sigma(\overline{X})$ begins with $\overline{a}$)* or
- *XY or $\overline{XY} = \overline{Y}\,\overline{X}$, for unknowns X, Y, occurs in U or V and $\sigma(X)$ ends with a while $\sigma(Y)$ begins with b (so $\sigma(\overline{Y})$ ends with $\overline{b}$ and $\sigma(\overline{X})$ begins with $\overline{a}$).*

Proof. Suppose that ab is crossing, which means that there is some ab-factor that is crossing. By definition it means that it does not come from one occurrence of a variable, nor from equation. Thus one of its constants comes from a variable and the other from the equation or from a different variable (note that as the solution is non-empty, that is $\sigma(Z) \neq \epsilon$ for any variable Z, the considered constants and variables are neighbouring in the word). Case inspection implies that one of the conditions listed in the lemma describes this situation.

So suppose that ab satisfies one of the conditions in the lemma. Then clearly ab is crossing: for instance, if aX occurs in the equation and $\sigma(X)$ begins with b then the ab-factor formed by this a and the first b in $\sigma(X)$ is crossing, other cases are shown in the same way. □

Since a crossing ab can be associated with an occurrence of a variable X, it follows that the number of crossing abs is linear in the number of occurrences of variables.

Lemma 5. *Let σ be a solution of an equation (U, V) with at most n occurrences of variables. Then there are at most $4n$ different crossing words in σ.*

Note that in particular Lemma 5 applies to proper equations.

Proof. Firstly observe that if $\sigma(X) = \epsilon$ for any variable, then we can remove all Xes from the equation and this does not influence whether ab is crossing or not and can only reduce the number of occurrences of variables. Thus we can assume that σ is non-empty, in the sense that $\sigma(X) \neq \epsilon$ for every variable present in the equation. So in the remaining part we assume that the assumptions of Lemma 4 are satisfied.

By Lemma 4 when ab is crossing, then one can associate ab (or $\overline{b}\overline{a}$) with an occurrence of a variable and its first or last constant. There are at most n occurrences of variables, so $2n$ occurrences with distinguished first or last constant and we have also two options of associating (ab or $\overline{b}\overline{a}$), so $4n$ possibilities in total, which yields the claim. □

3.4. Reduction for non-crossing ab.

When ab is non-crossing in the solution σ we can make the compression for all ab-factors that occur in the equation (U, V) on $\sigma(U)$ by replacing each ab-factor in U and V. The correctness follows from the fact that each maximal factor of ab-block in $\sigma(U)$ and $\sigma(V)$ comes either wholly from U (V, respectively) or from $\sigma(X)$. The former are replaced by our procedure and the latter are replaced implicitly, by changing the solution. Thus it can be shown that the solutions of the new and old equation are in one-to-one correspondence, i.e., are transformed by the procedure.

Algorithm 1 $\mathsf{CompNCr}(U, V, ab, \rho)$ Reduction for a non-crossing ab

```
S ← all maximal ab-factors in U and V and their involutions   ▷ S needs to be involution closed
for s ∈ S do
    let c_s be a fresh constant
    if s = s̄ then
        let c_s̄ denote c_s
    else
        let c_s̄ be a fresh constant
    replace each maximal ab-factor s (s̄) in U and V by c_s (c_s̄, respectively)
    set ρ(c_s) ← ρ(s) and ρ(c_s̄) ← ρ(s̄)
return (U', V')
```

To show that $\mathsf{CompNCr}(U, V, ab, \rho)$ transforms (U, V) with (σ, h), for which ab is non-crossing, we should define a corresponding solution (σ', h') of the obtained (U', V') as well as an inverse operator φ. Intuitively, they are defined as follows (let as in $\mathsf{CompNCr}(U, V, ab, \rho)$

the S be the set of all maximal ab-blocks in (U, V) and their involutions and let $\mathsf{CompNCr}(U, V, ab, \rho)$ replace $s \in S$ by c_s):

- $\sigma'(X)$ is obtained by replacing each $s \in S$ by c_s
- h' is h extended to new constants by setting $h'(c_s) = h(s)$
- the operator $\varphi_{\{c_s \to s\}_{s \in S}}$ is the the S-expansion, i.e., in each $\sigma(X)$ it replaces each c_s by s, for all $s \in S$.

Note that the defined operator is in the class defined in Section 2.2: all s_X, s'_X are ϵ while $E_1, \ldots, E_k$ are exactly the elements of S and thus new constants c_{E_i} are the constants c_s.

Lemma 6. *Let ab be non-crossing in a solution σ and let $\mathsf{CompNCr}(U, V, ab, \rho)$ compute a set of ab-blocks S in (U, V) and replace $s \in S$ by c_s. Then $\mathsf{CompNCr}(U, V, ab, \rho)$ transforms (U, V) with (σ, h) to (U', V') with (σ', h'), where h' is defined as above and $\varphi_{\{c_s \to s\}_{s \in S}}$ is the inverse operator. Moreover, $\sigma'(U')$ is an S-reduction of $\sigma(U)$.*

Proof. We define a new solution σ' by replacing each maximal factor $s \in S$ in any $\sigma(X)$ by c_s. Note that in this way $\rho(\sigma(X)) = \rho(\sigma'(X))$, as for each s we define $\rho(c_s) \leftarrow \rho(s)$. As all constants $\{c_s\}_{s \in S}$ are fresh, they do not occur in $\sigma(U)$ and so $\sigma(X)$ can be recreated from $\sigma'(X)$ by replacing each c_s by s, formally $\sigma = \varphi_{\{c_s \to s\}_{s \in S}}[\sigma']$, as claimed. This is a solution of (U', V'): consider any maximal ab-factor s in $\sigma(U)$ (or $\sigma(V)$):

- If it came from the equation then it was replaced by $\mathsf{CompNCr}(U, V, ab, \rho)$.
- If it came from a substitution for a variable and s
  - is in S then it was replaced implicitly in the definition of σ';
  - is not in S then it is left as it was.
- It cannot be crossing, as this contradicts the assumption.

Thus, $\sigma'(U')$ is obtained from $\sigma(U)$ by replacing each maximal ab-factor $s \in S$ by c_s (recall that by Lemma 3 all such maximal factors are disjoint), and the same applies to $\sigma'(V')$, in particular we obtain that $\sigma'(U') = \sigma'(V')$ and that this is an S-reductions of $\sigma(U) = \sigma(V)$.

We define h' simply by extending h to new constant c_s in a natural way: $h(c_s) = h(s)$ for each $s \in S$; note that such defined h' is compatible, since $\rho(c_s) = \rho(s)$ and h is known to be compatible. Furthermore $h'(\sigma'(U')) = h(\sigma(U))$, as $\sigma'(U')$ is obtained from $\sigma(U)$ by replacing each maximal factor $s \in S$ by c_s and by definition $h'(c_s) = h(s)$ and on all other constants they are equal.

We now must show that if (σ'', h'') is any solution of (U', V') then $\sigma'''(U) = \sigma'''(V)$, where $\sigma''' = \varphi_{\{c_s \to s\}_{s \in S}}[\sigma'']$: observe that $\varphi_{\{c_s \to s\}_{s \in S}}[\sigma''](U)$ is obtained from $\sigma''(U')$ by replacing each c_s by s: each constant c_s in U' was obtained from s in U, so in the other direction we replace c_s by s and for a constant c_s in $\sigma''(X)$ this follows by the definition of $\varphi_{\{c_s \to s\}_{s \in S}}$. The same applies to $\varphi_{\{c_s \to s\}_{s \in S}}[\sigma''](V)$, we obtain that indeed $\varphi_{\{c_s \to s\}_{s \in S}}[\sigma''](U) =$

$\varphi_{\{c_s \to s\}_{s\in S}}[\sigma''](V)$. So it is left to show that there is a h''' for constants present in $\sigma'''(U)$. We know that there is such a homomorphism for the alphabet of (U, V) (as it is a proper equation) and for other constants we can use the homomorphism h'', by the form of $\varphi_{\{c_s \to s\}_{s\in S}}$ there are no other constants in $h'''(U)$.

Concerning the weight, observe first that $h'(\sigma'(X)) = h(\sigma(X))$:

- if c_s replaced s then $h'(c_s) = h(s)$;
- for every preserved constant a it holds that $h'(a) = h(a)$.

Moreover, the number of occurrences of each variable is the same in (U, V) and in (U', V'), so the contribution to weight from variables is the same in both cases. Clearly we have $|U'| + |V'| \leq |U| + |V|$, so the weight does not increase. Furthermore, if $(U, V) \neq (U', V')$ then at least one factor was replaced in the equation and so $|U'| + |V'| < |U| + |V|$ and so the weight decreases. □

Example 6. Consider an equation $abXYY = Xab\overline{XXX}$, with $a \neq \overline{a}$ and $b \neq \overline{b}$. It has a solution $\sigma(X) = abab$ and $\sigma(Y) = \overline{b}\overline{a}\overline{b}\overline{a}\overline{b}\overline{a}$. It is easy to see that ab is noncrossing. After $\mathsf{CompNCr}(U, V, ab, \rho)$ the obtained equation is $cXYY = Xc\overline{XXX}$ and it has a solution $\sigma(X) = cc$ and $\sigma(Y) = \overline{ccc}$.

3.5. Reduction for crossing ab.

Since we already know how to compress a non-crossing ab, a natural way to deal with a crossing ab is to 'uncross' it and then compress using CompNCr. To this end we pop from the variables the whole parts of maximal ab-blocks which cause this block to be crossing. Afterwards all maximal ab-blocks are noncrossing and so they can be compressed using CompNCr.

3.5.1. Idea (example)

As an example consider an equation $abaXaXaXa = aXabYbYbY$, let $a = \overline{a}$ an $b = \overline{b}$ so that the ab-blocks are nontrivial. For simplicity, let us for now ignore the constraints. Also, let us focus on the solutions of the form $X \in b(ab)^{\ell_X}$ and $Y = (ab)^{\ell_Y}a$; clearly, ab is crossing in this solution. So we 'pop' from X the $b(ab)^{\ell_X}$ and $(ab)^{\ell_Y}a$ from Y (and remove those variables). After the popping this equation is turned into $(ab)^{3\ell_X+4}a = (ab)^{\ell_X+3\ell_Y+4}a$, for which ab is noncrossing. Thus solutions of the original equation (of the prescribed form $X = b(ab)^{\ell_X}$ and $Y = (ab)^{\ell_Y}a$) correspond to the solutions of the Diophantine equation: $3\ell_X + 4 = \ell_X + 3\ell_Y + 4$. This points out another idea of the popping: when we pop the whole part of block that is crossing, we do not immediately guess its length, instead we treat the number of repetitions of ab in the variables (here: ℓ_X and ℓ_Y) as *parameters*, identify ab-blocks of the same length (and type) and only afterwards verify, whether our guesses were correct. The verification is formalised as a linear system of Diophantine equations (here: $3\ell_X + 4 = \ell_X + 3\ell_Y + 4$) in parameters (here: ℓ_X and ℓ_Y). We can check solvability (and compute a minimal solution) in NP (so in particular in PSPACE), see e.g., [12]. (For a more accurate estimation of constants see [4]). Each solution of the Diophantine system

corresponds to one 'real' set of lengths of ab-blocks popped from variables. Now we replace equation $(ab)^{3\ell_X+4}a = (ab)^{\ell_X+3\ell_Y+4}a$ with $ca = ca$, which has a unique solution $\sigma(X) = \sigma(Y) = \epsilon$. Of course there is no single inverse operator, instead, they should take into the account the system $3\ell_X + 4 = \ell_X + 3\ell_Y + 4$. And it is so, for each solution (ℓ_X, ℓ_Y) of the Diophatine equation there is one inverse operator, which first replaces c with $(ab)^{\ell_X+3\ell_Y+4}$ and then appends $b(ab)^{\ell_X}$ to the substitution for X and $(ab)^{\ell_Y}a$ to the substitution for Y.

There are some details that were ignored in this example: during popping we need to also guess the types of the popped blocks and whether the variable should be removed (as now it represents ϵ) or not. Furthermore, we also need to calculate the transition of the popped ab-block, which depends on the number of repetitions of ab (i.e., on particular ℓ_X, ℓ_Y, etc.). Now, when we look at $\rho(ab)$, $\rho(ab)^2$, ... then starting from some (at most exponential) value it becomes periodic, the period is also at most exponential; to be more precise, we can guess the period p that is at most exponential and verify the guess by checking that $\rho(ab)^{2p} = \rho(ab)^p$. Then either ℓ_X is smaller than p or $\rho(ab)^{\ell_X} = \rho(ab)^{p+\ell_X \mod p}$. In both cases to calculate $\rho(ab)^{\ell_X}$ it is enough to know $\ell_X \mod p$, which we can guess, as it is at most exponential. The appropriate conditions are also written as Diophantine equations and added to the constructed linear Diophantine system which in total has polynomial size, as its coefficients are written in binary.

3.5.2. Detailed description

A full description is available also as a pseudocode, see Algorithm 2. The proof of correctness is provided in Lemma 7.

Idempotent power. In the preprocessing, when the ab-blocks are nontrivial (i.e., when $a = b$ *or* $a \neq b$ and $a = \overline{a}$ and $b = \overline{b}$)[2] we guess (some) idempotent power p of $\rho(ab)$ (when $a \neq b$) or $\rho(a)$ (when $a = b$, in the following we consider only the former case) in $\mathbb{M}_{2m}$, i.e., p such that $\rho(ab)^{2p} = \rho(ab)^p$. It is easy to show that there is such $p \leq |\mathbb{M}_{2m}| \leq 2^{4m^2}$, so we can restrict the guess so that the binary notation of p is of polynomial size. We verify the guess by computing $\rho(ab)^{2p}$ and $\rho(ab)^p$ in time $\mathsf{poly}(\log p, m)$ (the powers are computed be iterated squaring of the matrices) and checking, whether $\rho(ab)^{2p} = \rho(ab)^p$. Note that as p is also an idempotent power for $\overline{ab}$, ba and $\overline{ba}$: the first is clear, as $(\overline{ab})^k = \overline{(ab)^k}$; for the second, there are two cases: if $a = b$ then this is trivial, as $ab = ba$, if not then $a = \overline{a}$ and $b = \overline{b}$ and so $ba = \overline{ab}$; the third case is a combination of the two previous ones.

Popping and transitions. Now for every variable X we guess whether $\sigma(X)$ begins (and ends) with an ab-factor or a single-constant suffix (prefix, respectively) of an ab-factor; to simplify the notation, the *ab-prefix* of $\sigma(X)$ is the maximal prefix of $\sigma(X)$ that is also a (proper) suffix of some ab-factor; define the *ab-suffix* in a symmetric way. Note that an ab-prefix of $\sigma(X)$ may be empty, may consist of a single-constant (i.e., $\overline{a}$ or b, which are the possible final constants of an ab-factor) or have more constants in which case it is also an ab-factor

[2]Note that in the trivial case the maximal ab-blocks do not include any exponentiation of ab or a, and this is the reason, why they are "trivial".

itself. Thus, for each X we guess its ab-prefix s_X and and ab-suffix s'_X; note that $\overline{s_X} = s'_{\overline{X}}$, so we make the guesses consistent. We then left pop s_X and right-pop s'_X from X, i.e., we replace X with $s_X X s'_X$, when $\sigma(X) \neq s_X s'_X$ or with $s_X s'_X$, when $\sigma(X) = s_X X s'_X$. Consider s_X, if it includes a parameter x_X then it is of one of the forms: $s_X = a^{x_X}$ or $s_X = \overline{a}^{x_X}$ for some $x_X \geq 2$ (when $a = b$) or $s_X \in \{(ab)^{x_X}, (ab)^{x_X}b, b(ab)^{x_X}, (ba)^{x_X}\}$, for some $x_X \geq 1$ (when $a = \overline{a}, b = \overline{b}$ and $a \neq b$). Similar observation can be made for ab-suffix of X, i.e., s'_X, which uses y_X instead of x_X. We treat x_X, y_X as *parameters* denoting integers whose values are to be established later on (we do not use name *variables*, as this is reserved for variables representing words). Note that $x_X = y_{\overline{X}}$, so we add appropriate equations to the constructed system of Diophantine equations D. Eventually, we fix the values of x_X and y_X, say to ℓ_X and r_X. Then $s_X[\ell_X]$ denotes a string s_X in which we substituted a number ℓ_X for parameter x_X and so $s_X[\ell_X]$ is a string of a well-defined length (the same applies to $s'_X[r_X]$). If s_X does not depend on x_X then also $s_X[\ell_X]$ is a string that does not depend on ℓ_X, still we use this notion to streamline the presentation.

We now establish the new transitions by X as well as the the transitions of the popped blocks; if x_X and y_X are defined, the transition depends on them. Consider a transition $\rho(ab)^{x_X}$. Recall that p is the idempotent power for $\rho(ab)$ (and so also of $\rho(\overline{b}\overline{a})$). Let ℓ'_X be x_X modulo p, which is formalised by adding $x_X = k_{x_X}p + \ell'_X$ to D, where k is a new parameter and ℓ'_X is guessed. Then if $x_X < p$ (equivalently: $x_X = \ell'_X$) then $\rho(ab)^{x_X} = \rho(ab)^{\ell'_X}$; otherwise, when $x_X \geq p$ (equivalently: $x_X > \ell'_X$) then $\rho(ab)^{x_X} = \rho(ab)^{p+\ell'_X}$, as p is an idempotent power. Since ℓ'_X is already fixed, we guess, which of the cases holds, add an appropriate condition ($x_X = \ell'_X$ or $x_X > \ell'_X$) to D and calculate the value of $\rho(ab)^{x_X}$, which is $\rho(ab)^{\ell'_X}$ or $\rho(ab)^{p+\ell'_X}$. We perform similar operations for y_x, with r'_X as a modulo reminder. As a last step we also guess the new transitions for variables, the new $\rho(X')$ is such that $\rho(X) = \rho(s_X)\rho(X')\rho(s'_X)$.

Identical blocks. As we know the types of ab-factors in the equation (note that they do not depend on particular values of $\{x_X, y_X\}_{X\in\Omega}$), we can calculate the maximal ab-factors in the equation (as well as their types), even though x_X and y_X are not yet known. Denote those maximal ab-factors by $E_1, \ldots, E_\ell$, note that they may use $(ab)^{x_X}$ or $(ab)^{y_X}$, similarly as $\{s_X, s'_X\}_{X\in\Omega}$, we do not impose the condition that one E_i uses at most one such an expression (in fact, as seen in the example in Section 3.5.1, it can use several such expressions). Similarly as in case of $\{s_X, s'_X\}_{X\in\Omega}$, by $E_i[\{\ell_X, r_X\}_{X\in\Omega}]$ we denote E_i in which parameters x_X and y_X were replaced with numbers ℓ_X and r_X, for each variable X. Concerning their lengths, denote by e_i the length for E_i. Since the popped factors have lengths that are linear in x_X or y_X for some X the lengths $e_1, e_2, \ldots, e_\ell$ are also linear in $\{x_X, y_X\}_{X\in\Omega}$. It is easy to see (Lemma 7) that ℓ is polynomial in the size of the equation and so are the descriptions of each e_i. For the future reference, by $e_i[\{\ell_X, r_X\}_{X\in\Omega}]$ we denote the evaluation of expression e_i when x_X is substituted by ℓ_X and y_X is substituted by r_X, for each $X \in \Omega$. Clearly $|E_i[\{\ell_X, r_X\}_{X\in\Omega}]| = e_i[\{\ell_X, r_X\}_{X\in\Omega}]$.

Consider all maximal ab-factors of the same type, we guess the order between their lengths, i.e., we guess which of them are equal and what is the order between groups of expressions denoting equal lengths. We write the corresponding conditions into the system

of equations; formally we divide these expressions into groups $\mathcal{E}_1$, $\mathcal{E}_2$, ..., $\mathcal{E}_k$, one group contains only factors of the same type and its elements correspond to factors of the same length. For each group $\mathcal{E} = \{E_{i_1}, E_{i_2}, \ldots, E_{i_\ell}\}$ we add equations $e_{i_1} = e_{i_2}$, $e_{i_2} = e_{i_3}$, ..., $e_{i_{\ell-1}} = e_{i_\ell}$, which ensure that indeed those factors are of the same length. Then for all groups $\mathcal{E}_1$, $\mathcal{E}_2$, ..., $\mathcal{E}_k$ of factors of the same type we guess the relation between the lengths of factors between the groups, i.e.,. for each two groups $\mathcal{E}_i$ and $\mathcal{E}_j$ we choose elements from the group, say E_i and E_j, and add the appropriate of the inequalities $e_i < e_j$ or $e_i > e_j$ to the system. Intuitively, we can rule out the possibility that $e_i = e_j$, as we put e_i and e_j in different groups

Satisfiability of the system of Diophantine equations and inequalities. As the constructed Diophantine system D is of polynomial size, its satisfiability is in NP, see e.g., [12], so in particular in PSPACE. So we check the satisfiability of D: if it is not satisfiable, we terminate.

Replacement. When the system D is satisfiable, we replace all ab-factors in one group by a new constant, i.e., blocks $\{E_{i_1}, E_{i_2}, \ldots, E_{i_k}\}$ are replaced with a constant $c_{E_{i_1}}$ (the choice of E_{i_1} is arbitrary), obtaining the new equation.

Family of inverse operators. The corresponding family of inverse operators is defined in terms of system D, the popped prefixes and suffixes $\{s_X, s'_X\}_{X\in\Omega}$ and the maximal blocks $E_1, E_2, \ldots, E_k$ (replaced with constants $c_{E_1}, c_{E_2}, \ldots, c_{E_k}$), call this class $\Phi_{D,\{s_X,s'_X\}_{X\in\Omega},E_1,\ldots,E_k}$. For each solution $\{\ell_X, r_X\}_{X\in\Omega}$ of D the family contains an operator $\varphi_{\{\ell_X,r_X\}_{X\in\Omega}}$. The action of such an operator (on X) is as follows: it first replaces each constant c_{E_i} with ab-block $E_i[\{\ell_X, r_X\}_{X\in\Omega}]$ (so of length $e_i[\{\ell_X, r_X\}_{X\in\Omega}]$). Afterwards, we append $s_X[\ell_X]$ and prepend $s'_X[r_X]$ to the substitution for X.

Note that this family of operators is of the form promised in Section 2.2.1.

Algorithm 2 $\mathsf{CompCr}(U, V, ab, \rho)$ Compression of ab-blocks for a crossing ab

$p \leftarrow$ idempotent power of $\rho(ab)$ in $\mathbb{M}_{2m}$ ▷ Guess and verify when needed.
▷ It is the same for $\overline{ab}, ba, \overline{ba}$.
for $X \in \Omega$ **do**
  non-deterministically guess ab-prefix s_X and ab-suffix s'_X of $\sigma(X)$
  ▷ May depend on a parameter x_X and y_X, respectively
  ensure that $\overline{s_X} = s'_{\overline{X}}$ and $\overline{s'_X} = s_{\overline{X}}$
  add equations $x_X = y_{\overline{X}}$ and $y_X = x_{\overline{X}}$ to D ▷ They need to be equal
  add $x_X \geq 1$ (or $x_X \geq 2$) to D, when applicable
  ▷ Depending on s_X and whether $a = b$
  ▷ $x_X \geq 1$ for $a \neq b$, $\overline{a} = a$, $\overline{b} = b$ and $x_X \geq 2$ for $a = b$:
  replace each X with $s_X X s'_X$
  nondeterministically guess the reminders ℓ'_X and r'_X of x_X and y_X by p
  add $x_X = k_{x_X} p + x'_X$, $y_X = k_{y_X} p + r'_X$ to D
  nondeterministically guess, whether $x_X < p$ or not
  if $x_X < p$ **then**
    add $x_X = \ell'_X$ to D, calculate $\rho_s \leftarrow \rho(s_X[x_X])$
  else
    add $x_X > \ell'_X$ to D, calculate $\rho_s \leftarrow \rho(p + s_X[x_X])$
  ▷ Perform similar actions for s'_X, y_X and $\rho_{s'}$
  guess ρ_X such that $\rho(X) = \rho_s \rho_X \rho_{s'}$
  update $\rho(X) \leftarrow \rho_X$
  Guess nondeterministically, whether $\sigma(X) = \epsilon$ ▷ Make the same guess for $\sigma(X)$ and $\sigma(\overline{X})$
  if $\sigma(X) = \epsilon$ and $\rho(X) = \rho(\epsilon)$ **then** ▷ Make the same guess for $\sigma(X)$ and $\sigma(\overline{X})$
    remove X from the equation
let $E_1, E_2, \ldots, E_\ell$ be the maximal ab-factors in (U, V) and $e_1, e_2, \ldots, e_\ell$ their lengths
partition $\{E_1, E_2, \ldots, E_\ell\}$ into groups $\{\mathcal{E}_1, \ldots, \mathcal{E}_k\}$, ▷ Guess the partition
▷ Each group has ab-factors of the same type
for each group $\mathcal{E}_{i_j} = \{E_{i_1}, E_{i_2}, \ldots, E_{i_\ell}\}$ **do**
  add equations $\{e_{i_1} = e_{i_2}, e_{i_2} = e_{i_3}, \ldots, e_{i_{\ell-1}} = e_{i_\ell}\}$ to D
for different groups $\mathcal{E}_i$ and $\mathcal{E}_j$ of ab-factors of the same type **do**
  take any $e_i \in \mathcal{E}_i$, $e_j \in \mathcal{E}_j$, add one of inequalities $\{e_i < e_j\}$ or $\{e_i > e_j\}$ to D
non-deterministically check, whether D has a solution ▷ In NP
for each part $\mathcal{E}_i = \{E_{i_1}, E_{i_2}, \ldots, E_{i_\ell}\}$ **do**
  let $c_{E_{i_1}}$ be an unused constant ▷ Choice of E_{i_1} is arbitrary
  replace blocks $E_{i_1}, E_{i_2}, \ldots, E_{i_\ell}$ by $c_{E_{i_1}}$

Inverse operator for a particular solution. For a solution (σ, h) consider the run of $\mathsf{CompCr}(U, V, ab, \rho)$ in which the non-deterministic choices are done according to σ: i.e., we guess $\{s_X, s'_X\}_{X \in \Omega}$ such that $s_X[\ell_X]$ is the ab-prefix of $\sigma(X)$ and $s'_X[r_X]$ is the ab-suffix of $\sigma(X)$ for appropriate

values $\{\ell_X, r_X\}_{X\in\Omega}$ (when $\sigma(X)$ is a single ab-block then we divide it into s_X and s'_X in an arbitrary way, but we make the guesses consistent for X and X', for instance we can guess we guess $s'_X = \epsilon$, $s_{\overline{X}} = \epsilon$ and $s'_{\overline{X}} = \sigma(\overline{X})$); note that such values can be read out of $\sigma(X)$: we know that its ab-prefix and ab-suffix are of the same type as s_X and s'_X, so they are equal to $s_X[\ell_X]$ and $s'_X[r_X]$ for some values ℓ_X and r_X (when whole $\sigma(X)$ is an ab-block then there is more then one choice for ℓ_X and r_X). We partition the arithmetic expressions according to σ, i.e., E_i and E_j (of the same type) are in one group if and only if $E_i[\{\ell_X, r_X\}_{X\in\Omega}] = E_j[\{\ell_X, r_X\}_{X\in\Omega}]$, note that those are the ab-factors created for σ after popping the ab-prefix and ab-suffix from each variable. In particular their lengths are equal, i.e., $e_i[\{\ell_X, r_X\}_{X\in\Omega}] = e_j[\{\ell_X, r_X\}_{X\in\Omega}]$. Additionally, for two different groups $\mathcal{E}_i$ and $\mathcal{E}_j$ of expressions of the same type we add an equation $e_i < e_j$ for some $e_i \in \mathcal{E}_i$ and $e_j \in \mathcal{E}_j$ if and only if $e_i[\{\ell_X, r_X\}_{X\in\Omega}] < e_j[\{\ell_X, r_X\}_{X\in\Omega}]$ (and in the other case we add the converse inequality $e_i > e_j$).

Then $\{\ell_X, r_X\}_{X\in\Omega}$ is a solution of a constructed system D and $\mathsf{CompCr}(U, V, ab, \rho)$ transforms (U, V) with (σ, h) and $\varphi_{\{\ell_X, r_X\}_{X\in\Omega}} \in \Phi_{D,\{s_X, s'_X\}_{X\in\Omega}, E_1, \ldots, E_k}$ is the corresponding inverse operator. Concerning homomorphism h', we extend h to new constants by setting $h'(c_{E_i}) = h(E_i[\{\ell_X, r_X\}_{X\in\Omega}])$.

It remains to formally state and prove the above intuitions.

Lemma 7. $\mathsf{CompCr}(U, V, ab, \rho)$ *transforms solutions. Let D be the system returned by* $\mathsf{CompCr}(U, V, ab, \rho)$ *for the corresponding non-deterministic choices, and let X left-popped s_X and right-popped s'_X and let expressions in groups $\mathcal{E}_1, \ldots, \mathcal{E}_k$ be replaced with constants $c_{E_1}, \ldots, c_{E_k}$. Then the family $\Phi_{D,\{s_X, s'_X\}_{X\in\Omega}, E_1, \ldots, E_k}$ is the corresponding family of operators.*

Furthermore $\sigma'(U')$ is an S-reduction of $\sigma(U)$, for a set S that contains all ab-blocks that have explicit or crossing maximal factors in $\sigma(U)$ and their involutions (and perhaps some more ab blocks).

Proof. Let us focus on a proper equation (U, V). As a first step, we shall show that indeed all maximal blocks have lengths that are arithmetic expressions in $\{x_X, y_X\}_{X\in\Omega}$, there are polynomially (in n) many such lengths and that each of them of them is also of polynomial size. Consider, what constants can be included in a maximal ab-factor. Before any popping there are $\mathcal{O}(|U| + |V|)$ constants in the equation. There are at most $2(|U| + |V|)$ popped factors (two for each occurrence of a variable) and each of their lengths is at most $1 + 2x_X$ (or $1 + 2y_X$), when it depends on a parameter, or 2, when it does not. Hence, the total sum of lengths is at most $\mathcal{O}(|U| + |V|)$ plus $2\sum_{X\in\Omega}(x_X + y_X)$. Since the equation is proper, the former is quadratic in n, the latter is linear in n, as we count one expression $(ab)^{x_X}$ as if it had a constant length. Now, every popped s_X (and s'_X) goes into exactly one maximal ab-factor, so indeed the lengths are expressions linear in $\{x_X, y_X\}_{X\in\Omega}$.

We now show that if (U, V) has a solution (σ, h) then for appropriate non-deterministic choices we transform it into (U', V') with (σ', h') and the inverse operator is in the defined family. So consider such a solution (σ, h). As already noted, consider the non-deterministic guesses of $\mathsf{CompCr}(U, V, ab, \rho)$ that are consistent with (σ, h), i.e., for each variable X let its ab-prefix and ab-suffix be $s_X[\ell_X]$ and $s'_X[r_X]$ (note that it may be that s_X does not depend on the parameter x_X, or s'_X on y_X, it may be that one of them is ϵ; if whole $\sigma(X)$

is an ab-block then we split between s_X and s'_X arbitrarily). Note that such s_X, ℓ_X, s'_X and r_X exist, as various s_X and s'_X define all types of ab-prefix and ab-suffix of $\sigma(X)$ and particular ab-prefixes and suffixes are obtained by substituting ℓ_X and r_X for x_X and y_X, respectively. Note that when $\sigma(X)$ is an ab-block (or a single constant a or b) then the division of $\sigma(X)$ into ab-prefix and ab-suffix is not unique. We can make it in an arbitrary way, as long as it is consistent for $\sigma(X)$ and $\sigma(\overline{X})$. Let $\mathsf{CompCr}(U, V, ab, \rho)$ guess those s_X and s'_X. Let also $\mathsf{CompCr}(U, V, ab, \rho)$ remove X from the equation only when this is needed, i.e., $\sigma(X) = s_X[\ell_X]s'_X[r_X]$.

Consider the equation (U_1, V_1) obtained from the equation calculated so far by $\mathsf{CompCr}(U, V, ab, \rho)$ by substituting $\{\ell_X, r_X\}_{X\in\Omega}$ for $\{x_X, y_X\}_{X\in\Omega}$. Then it has a solution (σ_1, h), where $\sigma(X) = s_X[\ell_X]\sigma_1(X)s'_X[r_X]$. Moreover, $\sigma(U) = \sigma_1(U_1)$. This is easy to see: $s_X[\ell_X]$ and $s'_X[r_X]$ are the ab-prefix and ab-suffix of $\sigma(X)$ (by their definition) and we replace X by $s_X[\ell_X]Xs'_X[r_X]$ (or $s_X[\ell_X]s'_X[r_X]$).

Let $E_1, \ldots, E_\ell$ be the maximal ab-factors calculated by the $\mathsf{CompCr}(U, V, ab, \rho)$. Then in (U_1, V_1) the maximal ab-factors are $E_1[\{\ell_X, r_X\}_{X\in\Omega}], \ldots, E_\ell[\{\ell_X, r_X\}_{X\in\Omega}]$ and have lengths $e_1[\{\ell_X, r_X\}_{X\in\Omega}]$, $\ldots$, $e_\ell[\{\ell_X, r_X\}_{X\in\Omega}]$: the s_X and s'_X were chosen so that they are of the type of the ab-prefix and ab-suffix of $\sigma(X)$ and $s_X[\ell_X]$ and $s'_X[r_X]$ are the ab-prefix and ab-suffix of $\sigma(X)$.

Lastly, the ab is non-crossing in (U_1, V_1) in σ_1: suppose that it is not. As we assumed that we removed X when $\sigma(X) = s_X[\ell_X]s'_X[r_X]$ then this means that σ_1 is non-empty and so we can apply Lemma 4. As the cases listed in the lemma are symmetric, suppose that aX occurs in (U_1, V_1) and $\sigma_1(X)$ begins with b. If $s_X = \epsilon$ then this is a contradiction, as then $\sigma(X)$ also begins with b and so we guessed the ab-prefix of $\sigma(X)$ incorrectly. Thus $s_X \neq \epsilon$ and so also $s_X[\ell_X] \neq \epsilon$. If $s_X[\ell_X]$ consists of at least two constants then it is an ab-factor and it overlaps with the ab-factor consisting of the last constant of $s_X[\ell_X]$ and the following b and so by Lemma 3 the $s_X[\ell_X]b$ is also an ab-factor, which contradicts the choice of s_X. If s_X is a single constant, i.e., a, then we clearly guessed incorrectly: $s_X[\ell_X]\sigma_1(X)$ begins with ab and so also $\sigma(X)$ begins with ab, thus we should have popped an ab-factor from it and not a single a. The other cases are shown in the same way.

At this moment $\mathsf{CompCr}(U, V, ab, \rho)$ also calculates the transitions ρ_s, $\rho_{s'}$ as well as adds some equations to the system; we focus on ρ_s, the actions for $\rho_{s'}$ are symmetrical. Let it make the following choices: firstly, it correctly guesses the remainder ℓ'_X of x_X modulo p, i.e., it guesses $\ell'_X = \ell_X \mod p$. In particular, the equation $x_X = k_{x_X}p + \ell'_X$ is satisfiable. If $\ell_X < p$ then let it guess that $x_X < p$. Then ℓ_X satisfies the added equation $x_X = \ell'_X$. Then $\rho(s_X[\ell_X]) = \rho(s_X[\ell'_X])$ and this is the value substituted for ρ_s, so $\rho_s = \rho(s_X[\ell_X])$. On the other hand, if $\ell_X \geq p$ then the added inequality $x_X > \ell'_X$ is satisfied by ℓ_X. Moreover, as p is an idempotent power for $\rho(ab)$ (or $\rho(a)$, when $a = b$; for the simplicity of presentation in the following we consider only the former case), we know that $\rho(s_X[\ell_X]) = \rho(s_X[kp + \ell'_X]) = \rho(s_X[p + \ell'_X])$, thus ρ_s has the value $\rho(s_X[\ell_X])$. The same analysis applies to the actions that calculate $\rho_{s'} = \rho(s'_X[r_X])$. Hence there is a transition $\rho_X = \rho(\sigma_1(X))$ such that $\rho_s\rho_X\rho_{s'} = \rho(X)$, we guess this value for ρ_X.

Consider the equations and inequalities on $e_1, \ldots, e_k$ added to D. An equality $e_i = e_j$ is added if and only if $E_i[\{\ell_X, r_X\}_{X\in\Omega}] = E_j[\{\ell_X, r_X\}_{X\in\Omega}]$ (which implies that $e_i[\{\ell_X, r_X\}_{X\in\Omega}] =$

$e_j[\{\ell_X, r_X\}_{X\in\Omega}]$) and we consider the choices in which inequality $e_i < e_j$ is added only when the corresponding blocks are of the same type and $e_i[\{\ell_X, r_X\}_{X\in\Omega}] < e_j[\{\ell_X, r_X\}_{X\in\Omega}]$. Thus $\{\ell_X, r_X\}_{X\in\Omega}$ satisfy those equations and inequalities as well.

Let us now investigate the replacement of ab-factors by $\mathsf{CompCr}(U, V, ab, \rho)$. Recall that we take the non-deterministic choices in which it assigns E_i and E_j into the same group if and only if $E_i[\{\ell_X, r_X\}_{X\in\Omega}] = E_j[\{\ell_X, r_X\}_{X\in\Omega}]$ and they represent factors of the same type. Then the corresponding ab-factors in (U_1, V_1) are equal. Hence the action of $\mathsf{CompCr}(U, V, ab, \rho)$ are equivalent to $\mathsf{CompNCr}(U_1, V_1, ab, \rho)$ (up to naming of the new constants), recall that we already showed that ab is non-crossing in σ_1. Lemma 6 guarantees that when ab is non-crossing in σ then $\mathsf{CompNCr}(U_1, V_1, ab, \rho)$ transforms the solution σ and the inverse operator replaces constants c_{E_i} with the corresponding blocks of length $e_i[\{\ell_X, r_X\}_{X\in\Omega}]$. So let (U_1, V_1) with (σ_1, h_1) be transformed to (U', V') with (σ', h'), as guaranteed Lemma 6. By the same lemma we know what is the inverse operator that transforms (U', V') with (σ', h') to (U_1, V_1) with (σ_1, h_1). From previous considerations we also know what is the inverse operator that transforms (U_1, V_1) with (σ_1, h_1) to (U, V) with (σ, h). It is easy to see that their composition is exactly $\varphi_{\{\ell_X, r_X\}_{X\in\Omega}}$ from $\Phi_{D,\{s_X, s'_X\}_{X\in\Omega}, E_1, \ldots, E_k}$. As $\{\ell_X, r_X\}_{X\in\Omega}$ is a solution of D, this shows the the appropriate inverse operator indeed is in $\Phi_{D,\{s_X, s'_X\}_{X\in\Omega}, E_1, \ldots, E_k}$.

To show that $\sigma' U'$ is an S-reduction of $\sigma(U)$, we can use Lemma 6: note that $\sigma_1(U_1) = \sigma(U)$ so it is enough to show that $\sigma' U'$ is an S-reduction of $\sigma_1(U_1)$. But Lemma 6 says that $\mathsf{CompNCr}(U_1, V_1, ab, \rho)$ performs a reduction for a set of all explicit ab-blocks on $\sigma_1(U_1)$. Since each crossing or explicit ab-block in (U, V) for σ is explicit in (U_1, V_1) for σ_1, we get the claim.

Concerning the weight, note that Lemma 6 shows that $\mathrm{w}(\sigma', h') \leq \mathrm{w}(\sigma_1, h_1)$ and the inequality is strict if $(U', V') \neq (U_1, V_1)$. Similarly, since $\sigma(X) = s_X[\ell_X]\sigma(X)s'_X[r_X]$ and each X was replaced with $s_X[\ell_X]Xs'_X[r_X]$, each popped $s_X[\ell_X]$ contributed $2|h_1(s_X[\ell_X])|$ to $\mathrm{w}(\sigma, h)$, while in $\mathrm{w}(\sigma_1, h_1)$ it contributes $|s_X[\ell_X]|$, the same applies to $s'_X[r_X]$. Thus $\mathrm{w}(\sigma_1, h_1) \leq \mathrm{w}(\sigma, h)$ and if any of s_X, s'_X is non-empty, the inequality is strict. In the end, $\mathrm{w}(\sigma', h') \leq \mathrm{w}(\sigma, h)$ and the equality happens only when no factor was replaced and nothing was popped, i.e., when $(U, V) = (U', V')$, as claimed.

We now move to the next part of the proof. Assume that (U, V) is turned into the equation (U', V') that has a solution (σ', h') and system D was created on the way; let also s_X and s'_X be popped to the left and right from X (any of those may be ϵ), finally, let blocks from partition parts $\mathcal{E}_1$, $\mathcal{E}_2$, …, $\mathcal{E}_k$ be replaced with constants c_{E_1}, c_{E_2}, …, c_{E_k}. We are to show that for any operator $\varphi \in \Phi_{D,\{s_X, s'_X\}_{X\in\Omega}, E_1, \ldots, E_k}$ it holds that $\varphi[\sigma'](U) = \varphi[\sigma'](V)$ and that there is some homomorphism h for $\varphi[\sigma'](U)$ compatible with ρ.

By definition, φ corresponds to some solution $\{\ell_X, r_X\}_{X\in\Omega}$ of D, let us fix this solution.

Consider the equation obtained by $\mathsf{CompCr}(U, V, ab, \rho)$ after popping constants but before replacement of ab-factors, i.e., the one using parameters $\{x_X, y_X\}_{X\in\Omega}$. Consider the equation obtained by substituting $\{\ell_X, r_X\}_{X\in\Omega}$ for those parameters, i.e., each s_X is replaced with $s_X[\ell_X]$ and each s'_X by $s'_X[r_X]$. Denote this equation by (U_1, V_1). If (σ_1, h) is a solution of (U_1, V_1) then (σ, h) is a solution of (U, V), where $\sigma(X)$ is obtained from $\sigma(X)$ by appending $s_X[\ell_X]$ and prepending $s'_X[r_X]$ to $\sigma_1(X)$:

- Since $\sigma(X) = s_X[\ell_X]\sigma_1(X)s'_X(r_X)$ and (U_1, V_1) was obtained by replacing X with $s_X[\ell_X]Xs'_X[r_X]$ (or $s_X[\ell_X]s'_X[r_X]$ and then $\sigma_1(X) = \epsilon$), we get that $\sigma(U) = \sigma_1(U_1)$ and similarly $\sigma(V) = \sigma_1(V_1)$.

- For the constraints: $\rho(\sigma_1(X_1)) = \rho_X$ calculated by $\mathsf{CompCr}(U, V, ab, \rho)$ and satisfying the condition $\rho(X) = \rho_{s_X}\rho_X\rho_{s'_X}$. As $\{\ell_X, r_X\}_{X\in\Omega}$ is a solution of D then $\mathsf{CompCr}(U, V, ab, \rho)$ calculated $\rho_s = \rho(s_X[\ell_X])$ and $\rho_{s'} = \rho(s'_X[r_X])$: the ρ_s and $\rho_{s'}$ do not depend on a particular solution of D, they are equal for all of them.

- For the involution, note that we assume that $\overline{\sigma_1(X)} = \sigma_1(\overline{X})$ and $\overline{s_X} = s'_{\overline{X}}$ and so we get that $\overline{\sigma(X)} = \sigma(\overline{X})$.

Concerning the homomorphism h, observe that σ and σ_1 use exactly the same constants out of A, so if h is compatible with σ_1, it is also with σ.

Note that the inverse operator transforming the solutions of (U_1, V_1) to solutions of (U, V) is a 'second part' of the inverse operator φ.

We now consider the 'first part' of the inverse operator. Consider an equation obtained from (U', V') by replacing every constant c_{E_i} by an ab-factor $E_i[\{\ell_X, r_X\}_{X\in\Omega}]$. Note that this is exactly the equation (U_1, V_1) considered before, i.e., the one that is obtained after popping $s_X[\ell_X]$ to the left and $s'_X[r_X]$ to the right from X, for each variable X. Change the solution of σ' in the same way, that is by replacing each constant c_{E_i} by an ab-factor $E_i[\{\ell_X, r_X\}_{X\in\Omega}]$, in this way we obtain a substitution σ_1. Observe that $\sigma_1(U_1) = \sigma_1(V_1)$, as each constant $c_{E_i} \in \sigma'(U')$ and $\sigma'(V')$ was replaced in the same way.

Concerning the constraints, we need to show that $\rho(E_i[\{\ell_X, r_X\}_{X\in\Omega}])$ is the same as $\rho(c_{E_i})$. This follows from the way (U', V') is constructed from (U, V). The constants a, b are from the alphabet of (U, V) and so we know their transition. The constructed system D ensures that $s_X[\ell_X]$ (and $\rho(s'_X[r_x])$) have always the same transition ρ_s and $\rho_{s'}$. Thus $\rho(c_{E_i})$ always has the transition as $\rho(E_i[\{\ell_X, r_X\}_{X\in\Omega}])$, regardless of the solution $\{\ell_X, r_X\}_{X\in\Omega}$.

Lastly, we show that there is a homomorphism h for the equation (U, V) defined for all constants in $\sigma(U)$ compatible with ρ. For the constants that are present in (U, V) we take any homomorphism compatible with (U, V), we know that it exists, as (U, V) is proper; for the constants that were present in (U', V') we similarly take a compatible h' and assign $h(c) = h'(c)$. Lastly, there are constants that came from decompressing c_{E_i} into ab-factors, but due to definition of φ, both a and b are present in (U, V) (or are from original alphabet). Hence h can be defined for each constant: either as any homomorphism for (U, V) or as any homomorphism on (U', V'). □

Main transformation.

The main procedure $\mathsf{TransformEq}(U, V)$ simply performs the ab-compression, the crossing or non-crossing variant, depending on whether ab is crossing or not (this is found out by a non-deterministic choice).

Algorithm 3 TransformEq(U, V)

choose ab, where a (b) is a constant from (U, V) or from B
if ab is crossing **then** ▷ Guess non-deterministically
    CompNCr(U, V, ab, ρ)
else
    CompCr(U, V, ab, ρ)
return (U, V)

The crucial property of TransformEq is that for an appropriate choice the resulting equation is proper (assuming that the input one was): if there is any noncrossing ab available then we choose it and performing the compression reduces the size of the equation. If all possible ab are crossing, we choose the one that occurs most often in the equation. Uncrossing introduces $2n$ constants to the equation, but the compression of a frequent pair removes at least as much.

Lemma 8. *Suppose that (U, V) is a proper equation such that $|U > 1|$ or $|V| > 1$ and it has a solution (σ, h). Then for appropriate choice of ab* TransformEq *transforms (U, V) with (σ, h) into (U', V') with (σ', h'), where (U', V') is again proper.*

Proof. Observe that the claim about the transformation holds by Lemma 6 and Lemma 7, what we need to show is that for some choice of ab the obtained equation is indeed proper.

If there is any non-crossing ab occurring in U or V then choose it: its compression will shorten U or V and then clearly also (U', V') is proper (moreover, h' for (U', V') exists by Lemma 6).

We show that when there is no non-crossing ab then there is at least one ab which is crossing: the assumption that $|U| > 1$ or $|V| > 1$ so one of those sides has size at least 2. If it has 2 consecutive constants, we have ab which cannot be non-crossing (by case assumption), so it is crossing. If there are no two such consecutive constants, we also have a crossing pair: take any constant (or last constant of the first occurrence of a variable) and the next constant, which comes from a variable. This ab is crossing.

Since the equation (U, V) is proper, it has at most $m \leq 16n^2$ constants. Consider all crossing abs, let there be k of them, by earlier discussion we know that $k \geq 1$. By Lemma 5 we know that $k \leq 4n$. Consider, how many constants are covered by ab-factors in the equation, for each crossing ab. Each constant is covered by at least one such factor (as there are no non-crossing abs, such a factor may include constants from a variable), so there is an ab whose maximal factors cover at least m/k constants, choose such an ab. As each ab-factor consist of at least 2 constants, replacing a factor by a single constants removes at least half of constants in a factor. Thus during the CompCr(U, V, ab, ρ) at least $m/(2k)$ constants are removed from the equation. Note that there is no contradiction in assuming that at least half of the constants is removed and that each variable pops one constant to each side:

- if the ab factor is explicit, it will loose at least half of its constants;

- if it is implicit and was popped from a variable then it will be replaced with one constant and does not count towards the amount of compressed constants;
- if the ab factor is crossing and has length k then it is replaced with 1 constant, which we charge towards the number of popped constants, so we can think that all explicit constants from this factor were in fact compressed.

Thus the new equation has size at most

$$\begin{aligned} m - \frac{m}{2k} + 2n &= (1 - \frac{1}{2k})m + 2n \\ &\leq (1 - \frac{1}{8n})m + 2n \\ &\leq (1 - \frac{1}{8n}) \cdot 16n^2 + 2n \\ &\leq 16n^2 - 2n + 2n \\ &= 16n^2 \ , \end{aligned}$$

as claimed. □

Proof of Lemma 2 and generation of the graphs representation of all solutions

The Lemma 2 is just a reformulation of Lemma 8. To see this, we reformulate it in the language of transformation of solutions.

Lemma 9 (A modernised statement of Lemma 2)**.** *Suppose that (U_0, V_0) is a proper equation with $|U_0| > 1$ or $|V_0| > 1$ and a simple solution (σ_0, h_0). Then for some nondeterministic choices a run of* TransformEq *on (U_0, V_0) the returned equation (U_1, V_1) and the corresponding family of inverse operators Φ satisfy*

- *(U_1, V_1) is proper*
- *(U_0, V_0) with (σ_0, h_0) is transformed to (U_1, V_1) with (σ_1', h_1), Φ is the corresponding family of inverse operators and (σ_1, h_1) is a simplification of (σ_1', h_1).*

4. Running time for satisfiability

For word equations over the free monoid (without the regular constraints) the known algorithms [27, 14] (non-deterministically) verify the satisfiability in time polynomial in n and $\log N$, where N is the length of the length-minimal solution. It is the common belief that N is at most exponential in n, and should this be so, those algorithms would yield that WordEquation is in NP. While our algorithm works in polynomial space, so far a similar bound on its running time is not known.

4.1. Outline

The approach for the free monoid follows the lines similar to Lemma 8: whenever possible, we compress the non-crossing *ab*s, which shortens the equation (and does not increase the length of the solution). Thus there are only polynomially many such steps in a row. When only crossing *ab*s are available, instead of choosing one that occurs often in (U, V), we choose one which occurs often in $\sigma(U)$, which ensures that $|\sigma(U)|$ drops by a fraction $(1 - 1/8n)$. As $(1 - 1/8n)^{8n} < 1/e$, after $8n$ such compressions the length of the solution is reduced by a constant fraction $1/e$; so there are $\mathsf{poly}(n) \log N$ compressions in total.

To control both the size of the equation and solution, we alternatively reduce the size of one and the other. This affects the bound on the size of kept equations: their maximal length can be up to $32n^2$ instead of $16n^2$. We modify the definition of proper equations appropriately, so that it allows equations of length up to $32n^2$.

There is a slight problem, though: it might be that some *ab*-factors occur only in substitutions for variables (so they are non-crossing) and a compression for them is 'void': in the simplification we replace them back with two constants (or more).

However, when no constraints are allowed in the equations, the initial approach works: if s is a factor in $\sigma(U)$ for a length-minimal solution σ then either s is a factor of U or it has a crossing occurrence in σ. Otherwise we could remove all factors s from the solution, obtaining a shorter solution, which contradicts the length-minimality of σ.

The regular constraints make such an argument harder: when we cross out a factor s from $\sigma(X)$, the $\rho(\sigma(X))$ changes, which is not allowed. This can be walked around: instead of crossing s out we replace it with a single constant that has the same transition as s, i.e., $\rho(a) = \rho(s)$. To this end we *extend* the original alphabet: we add to the original alphabet A constants a_P for each $P \in \rho(A^+)$, where $\rho(A^+)$ denotes the image of A^+ by ρ, i.e., $\{\rho(w) \mid w \in A^+\}$. This set can be big, so we do not store it explicitly, instead we have a subprocedure that tests whether $P \in \rho(A^+)$.

There is another technical issue: as we often apply simplification we do not really know what happens with a length-minimal solution: it could be that we shorten the solution by performing the S-reduction but then the simplification of the solution enlarges the solution, negating the gain of the compression. However, this is not the case for an alphabet extended as above: for such alphabets a length-minimal solution is simple; note that, we need to tune the definition of length-minimal solutions: now it takes into the account also the weight of the solution as a secondary factor.

4.2. ρ-closed alphabets

We begin with the precise definition of the ρ-closure of the alphabet and then show that a word equation with constraints is satisfiable over A if and only if it is satisfiable over the ρ-closure of A.

Given a finite alphabet A with involution together with a homomorphism ρ from A to $\mathbb{M}_{2m}$ we say that an alphabet A is ρ-*closed* if $\rho(A) = \rho(A^+)$, i.e., for each word $w \in A^+$ there exists a constant a such that $\rho(w) = \rho(a)$.

Usually, an alphabet is not ρ-closed, however, we can naturally extend it with 'missing' constants: for an alphabet A define a *ρ-closure* $\mathrm{cl}_\rho(A)$ of A:

$$\mathrm{cl}_\rho(A) = A \cup \left\{ a_P \;\middle|\; \text{there is } w \in A^+ \text{ such that } \rho(w) = P \right\} \ ,$$

where each a_P is a fresh constant not in A, $\overline{a_P} = a_{P^T}$ and $a_P \neq a_{P'}$ when $P \neq P'$. It is easy to see that $\mathrm{cl}_\rho(A)$ is ρ-closed. Whenever clear from the context (or unimportant), we will drop ρ in the notation and talk about closure and cl.

Viewing the equation over A as an equation over $\mathrm{cl}(A)$ does not change the satisfiability.

Lemma 10. *Suppose that we are given a word equation (U, V) with regular constraints (defined using a homomorphism ρ) over a free monoid generated by A. Then (U, V) has a solution over A if and only if it has a solution when treated as an equation over the alphabet of constants $\mathrm{cl}_\rho(A)$.*

Note that the set of all solutions of the equation may be different for A and $\mathrm{cl}_\rho(A)$, but in this section we are interested only in the satisfiability.

Proof. If (σ, h) is a solution over A then (σ, h') is of course a solution over $\mathrm{cl}_\rho(A)$, where h' is h extended as an identity to $\mathrm{cl}(A) \setminus A$.

On the other hand, when (σ, h) is a solution over $\mathrm{cl}_\rho(A)$ then we can create a solution over A: for each $P \in \rho(A^+)$ choose a word w_P such that $\rho(w_P) = P$, moreover choose in a way so that $\overline{w_P} = w_{P^T}$, and replace every a_P in $\sigma(X)$ by w_P. Since constants a_P do not occur in the equation, it is routine to check that the obtained substitution is a solution (and since $\rho(w_P) = \rho(a_P)$, that all constraints are satisfied). Lastly, we restrict h. □

Oracles for $\mathrm{cl}(A)$. Note that the size of $\mathrm{cl}(A)$ may be much larger than $|A|$ (in fact, exponential in the input size). Thus we cannot store it explicitly, instead, whenever a constant from $\mathrm{cl}(A) \setminus A$ is introduced to the instance, we verify, whether it is indeed in $\mathrm{cl}(A)$, i.e., whether the corresponding transition matrix P is in $\rho(A^+)$. In general, such check can be performed in PSPACE (and in fact it is PSPACE-complete in some cases), but it can be performed more efficiently, when we know an upper-bound on $|\mathrm{cl}(A)|$.

Lemma 11. *It can be verified in* PSPACE*, whether $P \in \rho(A)$. Alternatively, this can be verified in* $\mathsf{poly}(|\rho(A^+)|, n)$ *time.*

Proof. The proof is standard.

Let $w_P = a_1 a_2 \cdots a_k$ be the shortest (non-empty) word such that $\rho(w_P) = P$. As $\mathbb{M}_{2m}$ has at most 2^{4m^2} elements, we have that $k \leq 2^{4m^2}$: if it were longer then $\rho(a_1 \cdots a_i) = \rho(a_1 \cdots a_j)$ for some $i < j$ and thus $\rho(a_1 a_2 \cdots a_k) = \rho(a_1 a_2 \cdots a_i a_{j+1} \cdots a_k)$, which cannot happen, as this word is shorter than w_P.

Thus in PSPACE we can non-deterministically guess the constants a_1, a_2, …, a_k and verify that indeed $\rho(a_1 a_2 \cdots a_k) = P$. Alternatively, we can deterministically list all elements of $\rho(A^+)$ in $\mathsf{poly}(|\rho(A^+)|, n)$ time. □

4.3. Length-minimal solution

We now give a proper definition of a length minimal solution: First, we compare the solutions (σ_1, h_1) and (σ_2, h_2) by $|\sigma_1(U)|$ and $|\sigma_2(U)|$ and if those are equal, by $\mathrm{w}(\sigma_1, h_1)$ and $\mathrm{w}(\sigma_2, h_2)$.

Definition 7 (Length-minimal solution)**.** A solution (σ_1, h_1) (of an equation (U, V)) is *length-minimal* if for every other solution (σ_2, h_2) of this equation either

- $|\sigma_1(U)| < |\sigma_2(U)|$ *or*
- $|\sigma_1(U)| = |\sigma_2(U)|$ and $\mathrm{w}(\sigma_1, h_1) \leq \mathrm{w}(\sigma_2, h_2)$.

The usual definition says that σ_1 is length-minimal, when for each other solution σ_2 we have $|\sigma_1(U)| \leq |\sigma_2(U)|$. Note that Definition 7 refines the usual one, in the sense that if a solution is length-minimal according to Definition 7, it is also length-minimal in the traditional sense, but not the other way around. Furthermore, for the input equation the h_1 is an identity on all constants in the solution, so our refined notion coincides with the traditional one.

Combining the notions of length-minimal solutions and ρ-closed alphabet we can show that indeed each ab that occurs in a solution needs to have an explicit or crossing factor in the equation.

Lemma 12. *Suppose that (U, V) is an equation over an alphabet $B \supseteq A$, where A is ρ-closed. Let (σ, h) be a length-minimal solution. If s is a maximal ab-factor of $\sigma(U)$ then there is a factor s that is explicit or crossing for σ. In particular, if ab is noncrossing and there is an ab-factor s in $\sigma(U)$ then there is an explicit maximal factor s in (U, V).*

Proof. Suppose that this is not the case, i.e., there is a maximal ab-factor s in $\sigma(U)$ and there are no explicit nor crossing factors s. Let $P = \rho(s)$ and take constants a_P and $\overline{a_P}$ ($a_P = \overline{a_P}$ if and only if $\overline{s} = s$). Create a new substitution σ' by replacing each maximal factor s and $\overline{s}$ in each $\sigma(X)$ by a_P and $\overline{a_P}$, respectively. By Lemma 3 the maximal ab-factors do not overlap, so such a replacement is well-defined. Moreover, our replacement respects the involution. It is easy to see that σ' is a solution: each replaced maximal factor s and $\overline{s}$ is wholly inside a substitution for a variable, so all of them were replaced; moreover, as the factors are maximal, by Lemma 3, they do not overlap. The constraints are satisfied, as $\rho(s) = P = \rho(a_P)$ and $\rho(\overline{s}) = P^T = \rho(\overline{a_P})$. Clearly h is compatible with σ', as each constant outside of A used by σ' is also used by σ. Lastly, $|\sigma'(U)| < |\sigma(U)|$, as we replaced at least one ab-factor. This shows that (σ, h) is not length-minimal, contradiction.

For the second claim, note that if ab is non-crossing then there cannot be a crossing ab-factor, so there needs to be an explicit one. □

4.4. Equations over ρ-closed alphabet

We are now ready to show that for ρ-closed alphabets $\mathsf{poly}(n, \log N)$ applications of TransformEq reduce it to a trivial equation (for appropriate non-deterministic choices). The proof follows in two steps: firstly, in Lemma 13, we show that when TransformEq transforms

a length-minimal solution then the obtained solution is simple. Since the inverse operator for TransformEq is a S-expansion for a set of ab-factors, we know that the obtained solution is significantly shorter than the one before TransformEq. Then, Lemma 14 shows that for appropriate choices the length of the length-minimal solution is reduced by a constant fraction $(1/e)$ each $\mathsf{poly}(n)$ application of TransformEq. As a wrap up, when we begin with a length-minimal solution (of length N) after at most $\mathsf{poly}(n, \log N)$ compressions we end up with a solution of length 1, which ends the proof.

Lemma 13 (cf. Lemma 9)**.** *Suppose that a proper equation (U_0, V_0) over an alphabet of constants $B \supseteq A$ that is ρ-closed has a length-minimal solution (σ_0, h_0) and that* TransformEq *transforms (U_0, V_0) with a (σ_0, h_0) into (U_1, V_1) with a solution (σ_1, h_1). Then (σ_1, h_1) is simple, in particular, $\sigma_0 = \varphi[\sigma_1]$, where φ is the corresponding inverse operator.*

Proof. Consider an application of TransformEq on (U_0, V_0). According to Lemma 9 for appropriate non-deterministic choices made by TransformEq we obtain an equation (U_1, V_1) and an operator φ and a solution (σ_1, h_1) such that $\sigma_0 = \varphi[\sigma'_1]$ and (σ_1, h_1) is a simplification of (σ'_1, h_1). We show that (σ'_1, h_1) is in fact simple, i.e., $(\sigma_1, h_1) = (\sigma'_1, h_1)$.

Suppose for the sake of contradiction that (σ'_1, h_1) is not simple. Hence (σ'_1, h_1) uses a constant b that does not occur in the alphabet of (U_1, V_1). Let $P = \rho(b)$; by definition of h_1, we know that $\rho(b) = \rho(h_1(b))$ and $h_1(b) \in A^+$. Since A is ρ-closed, there is a_P such that $\rho(b) = \rho(a_P) = P$. Create (σ''_1, h_1) by replacing each b and $\overline{b}$ in any $\sigma(X)$ by a_P and $\overline{a_P}$. It is easy to verify that (σ''_1, h_1) is a solution of (U_1, V_1). Recall that by definition of σ_1 and φ

$$\sigma_0 = \varphi[\sigma'_1] \ .$$

Recall also that φ may append and prepend constants to $\sigma(X)$ (independently of X and of the substitution) and it may replace some constants (outside of A) by longer factors, again independently of X and of the substitution. Consider now

$$\sigma''_0 = \varphi[\sigma''_1] \ .$$

We intend to show that (σ''_0, h_0) is a solution of (U_0, V_0) and that it contradicts the length-minimality of (σ_0, h_0). Since φ is the inverse operator, we know that $\sigma''_0(U_0) = \sigma''_0(V_0)$ and we need to show that h_0 is defined on any constant assigned by σ''_0 outside A and that it is compatible with ρ on such constants. Suppose that this is not the case; then the same constant is also used by σ_0: if this constant was used by σ''_1 and not replaced by φ then it cannot be a_P nor $\overline{a_P}$, as they are both in A. Hence this constant is also used by σ'_1 and as it is not replaced, it occurs also in σ_0. On the other hand, if this constant was appended or prepended to $\sigma''_1(X)$ by φ, then the same constant is appended or prepended to $\sigma'_1(X)$. Thus (σ''_0, h_0) is a solution of (U_0, V_0), it is left to show that (σ_0, h_0) is not length-minimal.

Firstly, we show that $|\sigma_0(U_0)| \geq |\sigma''_0(U_0)|$. Consider that $\varphi[\sigma'_1]$ and $\varphi[\sigma''_1]$ and their action on substitution for X. Then φ prepends and appends the same strings to $\sigma_1(X)$ and $\sigma''_1(X)$, additionally, it replaces some constants (outside A) in $\sigma_1(X)$ and $\sigma''_1(X)$ in the same way in both of them. As $\sigma''_1(X)$ is obtained from $\sigma_1(X)$ by replacing b and $\overline{b}$ by $a_P, \overline{a_P} \in A$, so $h(\sigma''_0(X))$ and $h(\sigma_0(X))$ differ only in strings that replace b and $\overline{b}$: in the former those are

a_P and $\overline{a_P}$ while in the latter those are strings $h(b)$ and $h(\bar{b})$ (of lengths at least 2). Thus $|\sigma_0''(X)| \leq |\sigma_0(X)|$ and the inequality is strict when b or $\bar{b}$ is in $\sigma_1'(X)$. As this constant occurs in at least one $\sigma_1'(X)$, we conclude that σ_0 is not length-minimal. □

Lemma 14. *Suppose that a proper equation (U_0, V_0) over an alphabet of constants $B \supseteq A$ that is ρ-closed has a length-minimal solution of size N. Then after (appropriate) $\mathcal{O}(n^3)$ applications of* TransformEq *the obtained proper equation (U_k, V_k) has a solution of size at most N/e.*

Proof. Imagine the following process, see Algorithm 4. For an equation (U, V) and its length-minimal solution we guess, whether there is some non-crossing pair for them. If so, we make the compression of this pair. If not, we choose one of the crossing abs. In an alternating way we choose such an ab so that it covers many constants in $\sigma(U)$ or in (U, V), one of them *shortens the solution* and the other *shortens the equation*. Note that as in the Lemma 8 we can show that when there are no non-crossing abs in (U, V) then there is at least one crossing ab.

Algorithm 4 ShortenSol(U, V)

```
1:  odd ← 0
2:  while |U| > 1 or |V| > 1 do
3:      fix a length-minimal solution (σ, h)                    ▷ Mental experiment
4:      if there is non-crossing ab in (σ, h) occurring in (U, V) then
5:          CompNCr(U, V, ab, ρ)
6:      else
7:          if odd is even then                  ▷ ab that is shortening the solution
8:              choose ab covering most constants in σ(U)                  ▷ Guess
9:          else                                 ▷ ab that is shortening the equation
10:             choose ab covering most constants in (U, V)                ▷ Guess
11:         CompCr(U, V, ab, ρ)
12:         odd ← (odd + 1)  mod 2
```

We show that for appropriate choices all equations on the way are proper, we need to redefine the notion first, though: the size of the equation is at most $32n^2$, not $16n^2$ as before. Consider the run between two consecutive compressions of crossing abs and the size of the equation. It is not increased during each of the non-crossing compressions. It can be increased up to $2n$ constants (popped from a variable) for each crossing compression, so by $4n$ in total. Now the rest of the argument is as in Lemma 8, just with a larger constant, as we introduced $4n$ and not $2n$ constants.

So consider the sizes of the length-minimal solutions of the consecutive equations. We show, that

1. non-crossing compressions decrease the size of the equation by at least 1;
2. compressions do not increase the length of the length-minimal solution;

3. each crossing compression that reduce the size of the solution reduces it by a fraction at least $(1 - 1/8n)$

From 1 we obtain that there are at most $32n^2$ non-crossing compressions in between crossing compressions, so there are $64n^2 + 1$ compressions between two crossing compression that reduces the size of the solution. From 2 in between such two compressions the size of the length minimal solution does not increase. Thus, by 3, after $64n^2+2$ compressions the size of the length-minimal solution is reduced by $(1-1/8n)$. So after at most $8n(64n^2+2) = \mathcal{O}(n^3)$ the size of the length minimal solution is reduced by $(1 - 1/8n)^{8n} \leq 1/e$, as claimed.

Consider 1. By Lemma 12 if ab is non-crossing then there is an explicit ab-factor in the equation. Thus the non-crossing ab removes at least 1 constant from the equation.

Consider 2, let us look first at the non-crossing compression: then $\mathsf{CompNCr}(U, V, ab, \rho)$ transforms (U, V) with (σ, h) to (U', V') with (σ', h'), see Lemma 6. Since (σ, h) is length-minimal, by Lemma 13 the (σ', h') is simple and so in particular $|\sigma'(U')|$ upper-bounds the length of the length-minimal solution. From Lemma 6 we know that $\sigma'(U')$ is an S-reduction of $\sigma(U)$, thus $|\sigma(U)| \geq |\sigma(U')|$. A similar argument (using Lemma 7 instead of Lemma 6) applies to the crossing compression.

Consider 3, i.e., consider the application of the crossing compression that reduces the size of the solution. Consider all different crossing ab factors. By Lemma 5 there are at most $4n$ of them. Each constant of $\sigma(U)$ is covered by at least one constant of some maximal ab-factor, where ab is crossing, see Lemma 12 (recall that by case assumption there are no non-crossing abs). So there is ab whose factors cover at least $|\sigma(U)|/4n$ constants. Consider the equation that is obtained by compressing such an ab using $\mathsf{CompCr}(U, V, ab, \rho)$. Then by Lemma 7, for appropriate non-deterministic choices, the obtained equation (U', V') has a solution (σ', h') such that $\sigma'(U')$ is an S-reduction of $\sigma(U)$ for the set of all maximal ab-factors that are explicit or crossing in σ and their involutions (and perhaps some other ab-blocks). By Lemma 12 those are all maximal ab-factors. Thus each such factor is shortened by at least half, so in total at least $|\sigma(U)|/8n$ constants were removed.

$$|\sigma'(U')| \leq |\sigma(U)| - \frac{|\sigma(U)|}{8n} = \left(1 - \frac{1}{8n}\right)|\sigma(U)|.$$

Now, from Lemma 13 we know that (σ', h') is a simple solution and so in particular $|\sigma'(U')| \leq \left(1 - \frac{1}{8n}\right)|\sigma(U)|$ upper-bounds the length of the length-minimal solution. □

4.5. Running time for equations over groups

As a consequence of Lemma 13, we can verify the satisfiability of a word equation in free groups (without rational constraints) in (nondeterministic) time $\mathsf{npoly}(\log N, n)$, where N is the size of the length minimal solution.

Theorem 2. *The satisfiability of word equation over free group (without rational constraints) can be verified in* PSPACE *and at the same time* $\mathsf{npoly}(\log N, n)$ *time, where N is the size of the length-minimal solution of this equation.*

Proof. We reduce the problem in a free group to the corresponding one in a free semigroup, see Proposition 1. In this way we introduce regular constraint, and these are the only constraints in the problem. This constraint says that $a\overline{a}$ cannot be a factor of X, for any a. The NFA for this condition has $|\Gamma|^2 + 2$ states:

- sink (all transitions to itself)
- initial state
- a state (a, b), where a is the first constant of the word and b the last.

The transitions are obvious. It is easy to see that $\mathbb{M}_{2m}$ has $\mathcal{O}(|\Gamma|^2)$ elements. Thus the subprocedure for checking whether $P \in \rho(A^+)$ can be implemented in $\mathsf{poly}(n)$, see Lemma 11.

Concerning the problem in the free monoid, we first extend the alphabet A to $\mathrm{cl}(A)$. By Lemma 10 those problems are equisatisfiable. By Lemma 14 the length of the substitution for a variable drops by a constant fraction after $\mathcal{O}(n^3)$ applications of TransformEq (for appropriate non-deterministic choices), so there are only $\mathcal{O}(n^3 \log N)$ application of this procedure till the solution is reduced to a single constant, in which case also the sides of the equation have sizes 1. Clearly each such an application takes time polynomial in n (as all equation are proper by Lemma 8). □

5. Applications

Using the results above we obtain the following theorem:

Theorem 3. *It can be decided in* PSPACE *whether the input system with rational constraints has a finite number of solutions.*

Proof. To find out whether the equation has infinite number of solutions it is enough to find a path from a start node of the graph to a final node which either

- contains a loop *or*
- one of the edges of the path is labelled by a linear system of equations having infinite number of solutions *or*
- the final node has infinite number of solutions, which means that it is of the form (X, Y), $\rho(X) = \rho(Y)$ and there are infinitely many words w such that $\rho(w) = \rho(X)$.

The first condition is a simple reachability problem in a graph, which can be performed in NPSPACE, as the description of the nodes and edges are of polynomial size. The second condition can be verified in NP, see Proposition 2. The last condition can be easily verified in PSPACE. Since NPSPACE contains NP and is equal to PSPACE, the search of such a path can be done in PSPACE.

Now if none of those conditions is satisfied, the graph representation of all solutions is a finite DAG, for each edge the family of inverse operators is finite and each final node has finitely many solutions, which implies that there are only finitely many solutions in total (although we may counted some of them several times). □

[1] M. Benois. Parties rationelles du groupe libre. *C. R. Acad. Sci. Paris, Sér. A*, 269:1188–1190, 1969.
[2] F. Dahmani and V. Guirardel. Foliations for solving equations in groups: free, virtually free and hyperbolic groups. *J. of Topology*, 3:343–404, 2010.
[3] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, 35(4):413–422, 1913.
[4] V. Diekert, C. Gutiérrez, and Ch. Hagenah. The existential theory of equations with rational constraints in free groups is PSPACE-complete. *Information and Computation*, 202:105–140, 2005. Conference version in STACS 2001, LNCS 2010, 170–182, 2004.
[5] V. Diekert and M. Lohrey. Word equations over graph products. *IJAC*, 18(3):493–533, 2008.
[6] V. Diekert, Yu. Matiyasevich, and A. Muscholl. Solving word equations modulo partial commutations. *Theoretical Computer Science*, 224:215–235, 1999. Special issue of LFCS'97.
[7] V. Diekert and A. Muscholl. Solvability of equations in free partially commutative groups is decidable. *International Journal of Algebra and Computation*, 16:1047–1070, 2006. Journal version of ICALP 2001, 543–554, LNCS 2076.
[8] V. G. Durnev. Undecidability of the positive $\forall\exists^3$-theory of a free semi-group. *Sibirsky Matematicheskie Jurnal*, 36(5):1067–1080, 1995. In Russian; English translation: *Sib. Math. J., 36*(5), 917–929, 1995.
[9] S. Eilenberg. *Automata, Languages, and Machines*, volume A. Academic Press, New York and London, 1974.
[10] C. Gutiérrez. Satisfiability of word equations with constants is in exponential space. In *Proc. 39th Ann. Symp. on Foundations of Computer Science (FOCS'98), Los Alamitos (California)*, pages 112–119. IEEE Computer Society Press, 1998.
[11] C. Gutiérrez. Satisfiability of equations in free groups is in PSPACE. In *Proceedings 32nd Annual ACM Symposium on Theory of Computing, STOC'2000*, pages 21–27. ACM Press, 2000.
[12] J. E. Hopcroft and J. D. Ulman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.
[13] L. Ilie and W. Plandowski. Two-variable word equations. *Theoretical Informatics and Applications*, 34:467–501, 2000.
[14] A. Jeż. Recompression: a simple and powerful technique for word equations. In N. Portier and T. Wilke, editors, *STACS*, volume 20 of *LIPIcs*, pages 233–244, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
[15] O. Kharlampovich and A. Myasnikov. Irreducible affine varieties over a free group. II: Systems in triangular quasi-quadratic form and description of residually free groups. *J. of Algebra*, 200(2):517–570, 1998.
[16] O. Kharlampovich and A. Myasnikov. Elementary theory of free non-abelian groups. *J. of Algebra*, 302:451–552, 2006.
[17] A. Kościelski and L. Pacholski. Complexity of Makanin's algorithm. *J. ACM*, 43(4):670–684, 1996.
[18] G. 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).
[19] G. 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).
[20] G. S. Makanin. Decidability of the universal and positive theories of a free group. *Izv. Akad. Nauk SSSR*, Ser. Mat. 48:735–749, 1984. In Russian; English translation in: *Math. USSR Izvestija, 25*, 75–88, 1985.
[21] Yu. Matiyasevich. Some decision problems for traces. In S. Adian and A. Nerode, editors, *Proceedings of the 4th International Symposium on Logical Foundations of Computer Science (LFCS'97), Yaroslavl, Russia, July 6–12, 1997*, volume 1234 of *Lecture Notes in Computer Science*, pages 248–257, Heidelberg, 1997. Springer-Verlag. Invited lecture.
[22] Yu. V. Matiyasevich. *Hilbert's Tenth Problem.* MIT Press, Cambridge, Massachusetts, 1993.
[23] W. Plandowski. Satisfiability of word equations is in NEXPTIME. In *Proceedings of the Symposium on the Theory of Computing STOC'99*, pages 721–725. ACM Press, 1999.
[24] W. Plandowski. Satisfiability of word equations with constants is in PSPACE. *J. ACM*, 51:483–496,

2004.

[25] W. Plandowski. An efficient algorithm for solving word equations. In *Proceedings of the 38th Annual Symposium on Theory of Computing STOC'06*, pages 467–476. ACM Press, 2006.

[26] W. Plandowski. personal communication, 2014.

[27] W. Plandowski and W. Rytter. Application of Lempel-Ziv encodings to the solution of word equations. In K. G. Larsen et al., editors, *Proc. 25th International Colloquium Automata, Languages and Programming (ICALP'98), Aalborg (Denmark), 1998*, volume 1443 of *Lecture Notes in Computer Science*, pages 731–742, Heidelberg, 1998. Springer-Verlag.

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

[29] A. A. Razborov. On systems of equations in free groups. In *Combinatorial and Geometric Group Theory*, pages 269–283. Cambridge University Press, 1994.

[30] E. Rips and Z. Sela. Canonical representatives and equations in hyperbolic groups. *Inventiones Mathematicae*, 120:489–512, 1995.

[31] K. U. Schulz. Makanin's algorithm for word equations — Two improvements and a generalization. In K. U. Schulz, editor, *Word Equations and Related Topics*, volume 572 of *Lecture Notes in Computer Science*, pages 85–150, Heidelberg, 1991. Springer-Verlag.

[32] Z. Sela. Diophantine geometry over groups VIII: Stability. *Annals of Math.*, 177:787–868, 2013.