On the Complexity of Solving Restricted Word Equations

Florin Manea[1], Dirk Nowotka[1], and Markus L. Schmid[2]

[1]Kiel University, Department of Computer Science, D-24098 Kiel, Germany, , {flm,dn}@zs.uni-kiel.de
[2]Fachbereich 4 – Abteilung Informatikwissenschaften, Universität Trier, 54286 Trier, Germany, mschmid@uni-trier.de

May 24, 2018

Abstract

We investigate the complexity of the solvability problem for restricted classes of word equations with and without regular constraints. The solvability problem for unrestricted word equations remains NP-hard, even if, on both sides, between any two occurrences of the same variable no other different variable occurs; for word equations with regular constraints, the solvability problems remains NP-hard for equations whose sides two sides share no variables or with two variables, only one of which is repeated. On the other hand, word equations with only one repeated variable (but an arbitrary number of variables) and at least one non-repeated variable on each side, can be solved in polynomial-time.

1 Introduction

A *word equation* is a symbolic equality $\alpha = \beta$, such that α and β are words over an alphabet $\Sigma \cup X$, where Σ is a finite alphabet of *constants* and $X = \{x_1, x_2, x_3, \ldots\}$ is an enumerable set of *variables*. A *solution* to a word equation $\alpha = \beta$ is a morphism $h : (\Sigma \cup X)^* \to \Sigma^*$ that satisfies $h(\alpha) = h(\beta)$ and $h(b) = b$ for every $b \in \Sigma$. For example, $x\mathtt{ab}y = \mathtt{b}yx\mathtt{a}$ is a word equation with variables x, y, constants $\mathtt{a}, \mathtt{b}$ and h with $h(x) = \mathtt{bab}$, $h(y) = \mathtt{aba}$ is a solution, since $h(x\mathtt{ab}y) = \mathtt{babababa} = h(\mathtt{b}yx\mathtt{a})$.

The *solvability problem* for word equations, i. e., to decide whether or not a given word equation has a solution, has a long history with the most prominent landmark being Makanin's algorithm [12] from 1977, which showed the solvability problem to be decidable (see Chapter 12 of [11] for a survey). While the complexity of Makanin's original algorithm was very high, it is nowadays known that the solvability problem is in PSPACE (see [9, 14]) and NP-hard (in fact, it is even believed to be in NP). Word equations with only a single variable can be solved in linear time [8] and equations with two variables can be solved in time $\mathcal{O}(n^5)$ [3]; it is not known whether there exist polynomial-time algorithms solving word equations with k variables, for $k \geq 3$.

If we require $\beta \in \Sigma^*$, i. e., only one side of the equation is allowed to contain variables, then we obtain the *pattern matching problem with variables* (or simply *matching problem*, for short), where the term *pattern* refers to the part α that can contain variables. The matching problem is NP-complete and, compared to the solvability problem for word equations, many more tractability and intractability results are known (see [5, 6, 15]). More precisely, while restrictions of numerical parameters (e. g., number of variables, number of occurrences per variable, length of substitution words, alphabet size, etc.) make the problem either polynomial-time solvable in a trivial way (e. g., if the number of variables is bounded by a constant) or result in strongly restricted, but still NP-complete variants (see [5]), structural restrictions of the pattern (e. g., of the order of the variables) are more promising and can yield rich classes of patterns for which the matching problem can be solved in polynomial-time (see [15]). For example, the matching problem remains NP-complete if $|\Sigma| = 2$, every variable has at most two occurrences in α and every variable can only be replaced by the empty word or a single symbol. Nevertheless, efficient polynomial-time algorithms exist (see [4]), if the patterns are *regular* (i. e., every variable has at most one occurrence), the patterns are *non-cross* (i. e., between any two occurrences of the same variable x no other variable different from x occurs) or the patterns have a bounded *scope coincidence degree* (i. e., the maximum number of scopes of variables that overlap is bounded, where the scope of a variable is the interval in the pattern where it occurs).

Technically, all these results can be seen as tractability and intractability results for restricted variants of the solvability problem for word equations (in fact, as it seems, all NP-hardness lower bounds for restricted variants of the solvability problem in the literature are actually NP-hardness lower bounds for the matching problem). However, these results are disappointing in terms of how much they provide us with a better understanding of the complexity of word equations, since in the matching problem the most crucial feature of word equations, of having variables on both sides, is missing.

The aim of this paper is to look into the complexity of the solvability problem for word equations, whose hardness is *not* derived from the hardness of the matching problem. In particular, we investigate whether the structural restrictions mentioned above, which are beneficial for the matching problem, can be extended, with a comparable positive impact, to classes of word equations that have variables on both sides. We pay special attention to *regular constraints*, i. e., each variable x is accompanied by a regular language L_x from which $h(x)$ must be selected in a solution h. While Makanin's algorithm still works in the presence of regular constraints, it turns out that for more restricted classes of equations, the addition of regular constraints can drastically increase the complexity of the solvability problem.

2 Definitions

Let Σ be a finite alphabet of *constants* and let $X = \{x_1, x_2, x_3, \ldots\}$ be an enumerable set of *variables.* For any word $w \in (\Sigma \cup X)^*$ and $z \in \Sigma \cup X$, we denote by $|w|_z$ the number of occurrences of z in w, by $\mathsf{var}(w)$ the set of variables occurring in w and, for every i, $1 \leq i \leq |w|$, $w[i]$ denotes the symbol at position i in w. A morphism $h : (\Sigma \cup X)^* \to \Sigma^*$ with $h(a) = a$ for every $a \in \Sigma$ is called a *substitution.* A *word equation* is a tuple $(\alpha, \beta) \in (\Sigma \cup X)^+ \times (\Sigma \cup X)^+$ (for the sake of convenience, we also write $\alpha = \beta$) and a *solution* to a word equation (α, β) is a substitution h with $h(\alpha) = h(\beta)$, where $h(\alpha)$ is the *solution word* (*defined by h*). A word equation is *solvable* if there exists a solution for it and the *solvability problem* is to decide for a given word equation whether or not it is solvable.

A word $\alpha \in (\Sigma \cup X)^*$ is usually called *pattern*, and $L(\alpha) = \{h(\alpha) \mid h \text{ is a substitution}\}$ is the *pattern language of α*. We say that α is *regular*[1], if, for every $x \in \mathsf{var}(\alpha)$, $|\alpha|_x = 1$; e.g., $\mathtt{a}x_1\mathtt{ba}x_2\mathtt{c}x_3\mathtt{bca}x_4\mathtt{a}x_5\mathtt{bb}$ is regular. The word α is *non-cross* if between any two occurrences of the same variable x no other variable different from x occurs, e. g., $\mathtt{a}x_1\mathtt{ba}x_1x_2\mathtt{a}x_2x_2x_3x_3\mathtt{b}x_4$ is non-cross, whereas $x_1\mathtt{b}x_1x_2\mathtt{ba}x_3x_3x_4x_4\mathtt{bc}x_2$ is not. A word equation (α, β) is regular or non-cross, if both α and β are regular or both α and β are non-cross, respectively.

An equation (α, β) is *variable disjoint* if $\mathsf{var}(\alpha) \cap \mathsf{var}(\beta) = \emptyset$.

For a word equation $\alpha = \beta$ and an $x \in \mathsf{var}(\alpha\beta)$, a *regular constraint* (*for x*) is a regular language L_x and a solution h for $\alpha = \beta$ *satisfies* the regular constraint L_x if $h(x) \in L_x$. The solvability problem for word equations with regular constraints is to decide on whether an equation $\alpha = \beta$ with regular constraints L_x, $x \in \mathsf{var}(\alpha\beta)$, given as NFA, has a solution that satisfies all regular constraints. The *size* of the regular constraints is the sum of the number of states of the NFA. If the regular constraints are all of the form Γ^*, for some $\Gamma \subseteq \Sigma$, then we call them *word equations with individual alphabets.* A word equation $\alpha = \beta$ along with an $m \in \mathbb{N}$ is a *bounded word equation.* The problem of *solving a bounded word equation* is then to decide on whether there exists a solution h for $\alpha = \beta$ with $|h(x)| \leq m$ for every $x \in \mathsf{var}(\alpha\beta)$.

[1]The usage of the term regular in this context has historical reasons: the matching problem has been first investigated in terms of pattern languages, which are regular languages if α is regular.

3 Regular and Non-Cross Word Equations

For the matching problem, the restriction of regularity implies that every variable has only one occurrence in the equation, which makes the solvability problem trivial (in fact, it boils down to the membership problem for a very simple regular language). However, word equations in which both sides are regular can still have repeated variables, although the maximum number of occurrences per variable is 2 (i. e., regular equations are restricted variants of quadratic equations (see, e. g., [16])) and these two occurrences must occur on different sides. Unfortunately, we are neither able to show NP-hardness nor to find a polynomial-time algorithm for the solvability problem of regular word equations.

Open Problem 1. *Can regular word equations be solved in polynomial-time?*

As we shall see later, solving a system of two regular equations is NP-hard (Corollary 1), solving regular equations with regular constraints is even PSPACE-complete (Theorem 3), and solving bounded regular equations or regular equations with individual alphabets is NP-hard (Corollaries 3 and 4, respectively), as well.

On the positive side, it can be easily shown that regular word equations can be solved in polynomial-time, if we additionally require them to be variable disjoint (no variable is repeated in the whole equation). More precisely, in this case, we only have to check emptiness for the intersection of the pattern languages described by the two sides of the equations (which are regular languages).

Next, we show that polynomial-time solvability is still possible if at most one variable is repeated, and each side contains at least one non-repeating variable.

Theorem 1. *Word equations with only one repeated variable, and each side containing at least one non-repeating variable, can be solved in polynomial time.*

Proof. Let us consider a word equation $\alpha = \beta$ such that α and β contain the repeated variable x and α contains, in order from left to right, the variables $y_1, \dots, y_k$ (each appearing exactly once) while β contains, in order from left to right, the variables $z_1, \dots, z_p$ (each appearing exactly once), with $k, p \geq 1$. We can also assume without losing generality that at least one of α and β starts with a variable and that they have an empty common prefix; also, assume that at least one of α and β ends with a variable and that they have an empty common suffix. For simplicity, if the equation $\alpha = \beta$ has a solution, let w_t be the image of the variable t under the solution-substitution h, for $t \in \{x\} \cup \{y_1, \dots, y_k\} \cup \{z_1, \dots, z_p\}$. Let us sketch the strategy we use to solve such equations, and show it works in polynomial time.

For the ease of presentation, we treat first the case $k, p \geq 2$. Let α' be the prefix of α occurring before y_1 and let β' be the prefix of β occurring before z_1. Let α'' be the suffix of α occurring after y_k and let β'' be the suffix of β occurring after z_p.

Let us assume that α' is empty. If neither of α'' and β'' is empty we proceed as in the next case (when neither α' nor β' is empty), and consider the mirrored equation $\alpha^R = \beta^R$ instead of the original one (where w^R is the mirror image of the word w). So we assume that at least one of α'' and β'' is empty. If α'' is empty we solve the equation as follows: $w_{y_1} = h(w)$, where w is the prefix of β occurring before z_p, $w_{z_p} = h(v)$, where v is the factor of α occurring between y_1 and y_k, and $w_{y_k} = h(\beta'')$, while the other variables are defined arbitrarily. If β'' is empty then we only define $w_{y_1} = h(w)$, where w is defined as above, and $w_{z_p} = h(v y_k \alpha')$, again v is defined as above; the other variables are defined arbitrarily. The case when β' is empty can be treated analogously: just exchange α and β.

We now look at the case when neither α' nor β' is empty. Let us assume that α' starts with x (the case when β' starts with x is similar: we exchange α and β to reach the case when the prefix of the left hand side starts with x).

If β' contains an x, then we let u be the prefix of β' that contains no variable. We get that (in a solution) w_x can either be a prefix of u or it can be expressed as $r^s r'$ where r is a primitive prefix of u and r' is a prefix of r and $s \geq 1$ (r is uniquely determined: its length is the period of u, and r is a prefix of u). Let s_0 be the smallest s such that $|r^s| > \max\{|\alpha|, |\beta|\}$. For each w_x prefix of u or $w_x \in \{r^s r' \mid s \leq s_0, r' \text{ prefix of } r\}$, we replace each occurrence of x in α and β by w_x and check if the obtained equation has a solution. For $s \geq s_0$ we do the following. For each r' prefix of r we replace each occurrence of x by $r^s r'$ (symbolically, not explicitly, as we do not know s). However, this still allows us to compare which of the prefixes β' and α' is shorter and to reduce the equations such that at least on of them starts with a variable (if the shorter of β' and α' is also a prefix of the longer one). We do the same with α'' and β'' and then obtain a new equation (γ, δ), that only contains y and z variables and where the former occurrences of x were replaced by $r^s r'$. Such an equation has always a solution. For instance, if $\gamma = y_1 v y_k v'$ where v' contains only constants, and $\delta = w' z_1 w z_p$, where w' contains only constants. then $w_{y_1} = h(w' z_1 w)$, while $w_{z_p} = h(v y_k v')$, and all the other images can be defined arbitrarily. The other cases can be treated analogously.

If β' contains no x, then w_x is either a prefix of β' (and we treat this case exactly like the above when w_x was a prefix of u) or has β' as a prefix. We replace everywhere the occurrences of x by $\beta' x$ to get a new equation; we then reduce the β' occurring at the beginning of both sides of this equation to get another equation (γ, δ), where both γ and δ start with variables y and, respectively, z. Now we can solve the mirrored equation just like the above.

The cases $k \geq 2, p = 1$ and $k = p = 1$ can be solved analogously: we just first try to reduce the prefixes that occur before the first y and z variables, respectively, as well as the suffixes that occur after the last y and z variables, respectively. If this is possible, we can easily define a substitution for the y and z variables. The cases when $k = 1$ and $p \geq 1$ can be solved by exchanging the sides of the equation. $\square$

If we allow an arbitrary number of occurrences of each variable, but require them to be sorted on both sides on the equation, where the sorting order might be different on the two sides, then we arrive at the class of non-cross word equations. The matching problem for non-cross patterns can be solved efficiently but, as we shall see next, for non-cross equations, the solvability problem becomes NP-hard.

Theorem 2. *Solving non-cross word equations is* NP*-hard.*

We prove this theorem by a reduction from a graph problem, for which we first need the following definition.

Let $\mathcal{G} = (V, E)$ be a graph with $V = \{t_1, t_2, \ldots, t_n\}$. A vertex s is the *neighbour* of a vertex t if $\{t, s\} \in E$ and the set $N_{\mathcal{G}}[t] = \{s \mid \{t, s\} \in E\} \cup \{t\}$ is called the (*closed*) *neighbourhood* of t. If, for some $k \in \mathbb{N}$, every vertex of $\mathcal{G}$ has exactly k neighbours, then $\mathcal{G}$ is *k-regular*. A *perfect code* for $\mathcal{G}$ is a subset $C \subseteq V$ with the property that, for every $t \in V$, $|N_{\mathcal{G}}[t] \cap C| = 1$. Next, we define the problem to decide whether a 3-regular graph has a perfect code, which is NP-complete (see [10]):

3-Regular Perfect Code (3RPerCode)
Instance: A 3-regular graph $\mathcal{G}$.
Question: Does $\mathcal{G}$ contain a perfect code?

We now define a reduction from 3RPerCode. To this end, let $\mathcal{G} = (V, E)$ be a 3-regular graph with $V = \{t_1, t_2, \ldots, t_n\}$ and, for every i, $1 \leq i \leq n$, N_i is the neighbourhood of t_i. Since the neighbourhoods play a central role, we shall define them in a more convenient way. For every r, $1 \leq r \leq 4$, we use a mapping $\wp_r : \{1, 2 \ldots, n\} \to \{1, 2 \ldots, n\}$ that maps an $i \in \{1, 2 \ldots, n\}$ to the index of the r^{th} vertex of neighbourhood N_i, i.e., for every i, $1 \leq i \leq n$, $N_i = \{t_{\wp_1(i)}, t_{\wp_2(i)}, t_{\wp_3(i)}, t_{\wp_4(i)}\}$. Obviously, the mappings $\wp_r$, $1 \leq r \leq 4$, imply a certain order on the vertices in the neighbourhoods, but, since our constructions are independent of this actual order, any order is fine.

We transform $\mathcal{G}$ into a word equation with variables $\{x_{i,j} \mid 1 \leq i, j \leq n\} \cup \{y_i, y'_i \mid 1 \leq i \leq n\}$ and constants from $\Sigma = \{\star, \diamond, \bar{\diamond}, \odot, \#, \mathsf{a}\}$. For every i, j, $1 \leq i, j \leq n$, the variable $x_{i,j}$ represents $t_i \in N_j$. For every i, $1 \leq i \leq n$, we define

$$\begin{aligned} \alpha_i &= x_{\wp_1(i),i} \ldots x_{\wp_4(i),i}, & \alpha'_i &= \# \, \mathsf{a}^8 \, \# \, \# \,, \\ \beta_i &= \mathsf{a}, & \beta'_i &= y_i \, \# (x_{i,\wp_1(i)})^2 \ldots (x_{i,\wp_4(i)})^2 \, \# \, y'_i, \qquad \text{and} \end{aligned}$$

$$\begin{array}{ccccccccccccc} u & = & \alpha_1 & \star & \ldots & \star & \alpha_n & \star & \odot & \bar{\diamond} & \alpha'_1 & \diamond & \ldots & \diamond & \alpha'_n \,, \\ v & = & \beta_1 & \star & \ldots & \star & \beta_n & \star & \odot & \bar{\diamond} & \beta'_1 & \diamond & \ldots & \diamond & \beta'_n \,. \end{array}$$

Proposition 1. *The words u and v are non-cross and can be constructed from $\mathcal{G}$ in polynomial time.*

Proof. In order to see that u and v are non-cross it is sufficient to note that, for every i, j, $1 \leq i \leq j \leq n$, $i \neq j$, the factors α_i and β'_i are non-cross and, furthermore, $\mathsf{var}(\alpha_i) \cap \mathsf{var}(\alpha_j) = \emptyset$ and $\mathsf{var}(\beta'_i) \cap \mathsf{var}(\beta'_j) = \emptyset$. It is obvious that $\mathcal{G}$ can be transformed into (u, v) in polynomial time. $\square$

Lemma 1. *The graph $\mathcal{G}$ has a perfect code if and only if (u, v) has a solution.*

Proof. For the sake of convenience, let $u = u_1 \odot u_2$ and $v = v_1 \odot v_2$. We start with the *only if* direction. For a perfect code C of $\mathcal{G}$, we construct a substitution h with $h(u) = h(v)$ in the following way. For every i, $1 \leq i \leq n$, we define $h(x_{i,\wp_r(i)}) = \mathsf{a}$, $1 \leq r \leq 4$, if $t_i \in C$, and $h(x_{i,\wp_r(i)}) = \varepsilon$, otherwise. Thus, for every i, $1 \leq i \leq n$, $h((x_{i,\wp_1(i)})^2 \ldots (x_{i,\wp_4(i)})^2) \in \{\mathsf{a}^8, \varepsilon\}$, which implies that $h(y_i)$ and $h(y'_i)$ can be defined such that $h(\beta'_i) = h(\alpha'_i)$. Consequently, $h(v_2) = h(u_2)$. Since C is a perfect code, for every i, $1 \leq i \leq n$, there is an r, $1 \leq r \leq 4$, such that $t_{\wp_r(i)} \in C$ and $t_{\wp_{r'}(i)} \notin C$, $1 \leq r' \leq 4$, $r \neq r'$. Therefore, $h(x_{\wp_1(i),i} x_{\wp_2(i),i} x_{\wp_3(i),i} x_{\wp_4(i),i}) = h(x_{\wp_r(i),i}) = \mathsf{a}$, which means that $h(\alpha_i) = h(\beta_i)$. Since this particularly implies $h(u_1) = h(v_1)$, we can conclude $h(u) = h(v)$.

In order to prove the *if* direction, we assume that there exists a solution h.

Claim: If $h(u_1) = h(v_1)$ and $h(u_2) = h(v_2)$, then $\mathcal{G}$ has a perfect code.

Proof of Claim: From $h(u_1) = h(v_1)$, we can directly conclude that, for every i, $1 \leq i \leq n$, $h(\alpha_i) = \beta_i$, which means that exactly one of the variables $x_{\wp_1(i),i}, x_{\wp_2(i),i}, x_{\wp_3(i),i}, x_{\wp_4(i),i}$ is mapped to a, while the others are mapped to ε. From $h(v_2) = h(u_2)$ it follows that, for every i, $1 \leq i \leq n$, $h(\beta'_i) = \alpha'_i$. Next, we observe that, for every i, $1 \leq i \leq n$, due to the symbols $\#$ in β'_i and α'_i, $h((x_{i,\wp_1(i)})^2 \ldots (x_{i,\wp_4(i)})^2) \in \{\mathsf{a}^8, \varepsilon\}$. Since each of the variables $x_{i,\wp_1(i)}, x_{i,\wp_2(i)}, x_{i,\wp_3(i)}, x_{i,\wp_4(i)}$ are mapped to either a or ε, this implies that either all of these variables are erased or all of them are mapped to a. Let C be the set of exactly the vertices $t_i \in V$ for which $h(x_{i,\wp_1(i)}) = h(x_{i,\wp_2(i)}) = h(x_{i,\wp_3(i)}) = h(x_{i,\wp_4(i)}) = \mathsf{a}$. For every neighbourhood $V_j = \{t_{\wp_1(j)}, t_{\wp_2(j)}, t_{\wp_3(j)}, t_{\wp_4(j)}\}$, $1 \leq j \leq n$, $h(x_{\wp_1(j),j}\, x_{\wp_2(j),j}\, x_{\wp_3(j),j}\, x_{\wp_4(j),j})$ is mapped to a, which implies that for some r, $1 \leq r \leq 4$, $h(x_{\wp_r(j),j}) = \mathsf{a}$; thus, $t_{\wp_r(j)} \in C$. Furthermore, $h(x_{\wp_{r'}(j),j}) = \varepsilon$, $1 \leq r' \leq 4$, $r \neq r'$, which means that $t_{\wp_{r'}(j)} \notin C$, $1 \leq r' \leq 4$, $r \neq r'$. Consequently, C is a perfect code. This concludes the proof of the Claim.

It remains to show that a solution h necessarily satisfies $h(u_1) = h(v_1)$ and $h(u_2) = h(v_2)$. Let w be the solution word of h. We first recall that, since $v_1, u_2 \in \Sigma^*$, $h(v_1) = v_1$ and $h(u_2) = u_2$, which particularly means that $v_1 \odot$ is a prefix and $\odot\, u_2$ is a suffix of w. If $|w|_\odot = 1$, then $w = v_1 \odot u_2$ and therefore $h(u_1) = h(v_1)$ and $h(u_2) = h(v_2)$. If, on the other hand, $|w|_\odot \geq 2$, then $w = v_1 \odot \gamma \odot u_2$. If $\gamma = \varepsilon$, then $w = v_1 \odot \odot u_2$, which is a contradiction, since w must contain the factor $\star \odot \bar{\diamond}$. From $h(u_2) = u_2$ and $h(v_1) = v_1$ it follows that $h(u_1) = v_1 \odot \gamma =$ and $h(v_2) = \gamma \odot u_2$. The factor v_2 starts with an occurrence of $\bar{\diamond}$ and since γ is a non-empty prefix of $h(v_2)$, this means that $|\gamma|_{\bar{\diamond}} = k \geq 1$. Moreover, γ is also a suffix of $h(u_1)$ and since $|u_1|_{\bar{\diamond}} = 0$, this implies that there are variables $z_1, z_2, \ldots, z_\ell \in \mathsf{var}(u_1)$, $1 \leq \ell \leq k$, with $\sum_{i=1}^{\ell} |h(z_i)|_{\bar{\diamond}} \geq k$. Since each of these variables z_i, $1 \leq i \leq \ell$, is repeated twice in v_2 and since $|v_2|_{\bar{\diamond}} = 1$, we can conclude that $|h(v_2)|_{\bar{\diamond}} \geq 2k + 1$. In the suffix $\odot\, u_2$ of $h(v_2)$, there is only one occurrence of $\bar{\diamond}$, which implies that $|\gamma|_{\bar{\diamond}} \geq 2k$. Since $k \geq 1$, this is clearly a contradiction to $|\gamma|_{\bar{\diamond}} = k$. $\square$

The equation obtained by the reduction from above has the form $u_1 \odot u_2 = v_1 \odot v_2$, where in a solution h, $h(u_1) = h(v_1)$ and $h(u_2) = h(v_2)$. In order to achieve this synchronisation between the two left parts and between the two right parts, we need to repeat variables in v_2. However, we can as well represent $u_1 \odot u_2 = v_1 \odot v_2$ as a system of two equations $u_1 = v_1$ and $u_2 = v_2$ and, since the synchronisation of the left parts and the right parts is now enforced by the fact that we regard them as two separate equations, we can get rid of the repeated variables in v_2, which makes the two equations regular.

Corollary 1. *The problem of checking solvability of a system of 2 regular word equations $\alpha_1 = \beta_1$, $\alpha_2 = \beta_2$ with $\beta_1, \beta_2 \in \Sigma^*$ is* NP*-hard.*

Proof. Following the reduction used in the proof of Theorem 2, let $u_1 = \alpha_1 \star \ldots \star \alpha_n$, $v_1 = \beta_1 \star \ldots \star \beta_n$, $u_2 = \alpha'_1 \diamond \ldots \diamond \alpha'_n$ and $v_2 = \beta'_1 \diamond \ldots \diamond \beta'_n$ be defined in the same way as in the proof of Lemma 1, with the only difference that $\alpha'_i = \# \, \mathtt{a}^4 \, \# \, \#$ and $\beta'_i = y_i \, \# \, x_{i,\wp_1(i)} \ldots x_{i,\wp_4(i)} \, \# \, y'_i$. Obviously, u_1, u_2, v_1 and v_2 are regular. Analogously to the proof of Lemma 1, it can now be easily verified that the system $u_1 = v_1$, $u_2 = v_2$ has a solution if and only if $\mathcal{G}$ has a perfect code. □

We conclude this section by stressing the fact that the non-cross equation from the reduction above is "almost regular", i. e., one side is regular, while for the other the maximum number of occurrences per variable is 2. However, we were not able to get rid of these repeated variables, which suggests that a hardness reduction for the regular case needs to be substantially different.

4 Word Equations with Regular Constraints

In practical scenarios, it seems rather artificial that we only want to find just any solution for a word equation and we are fine with whatever sequence of symbols the variables will be substituted with. It is often more realistic that the variables have a well-defined domain from which we want the solution to select the words. This motivates the addition of regular constraints to word equations, as defined in Section 2, for which we investigate the solvability problem in this section.

As mentioned in Section 1, regular constraints can be easily incorporated into algorithms for the general solvability problem. However, while it is open whether solving general word equations is hard for PSPACE, for word equations with regular constraints, this can be easily shown, even for regular equations.

Theorem 3. *Solving word equations with regular constraints is* PSPACE*-complete, even for regular equations.*

Proof. We can reduce the PSPACE-hard intersection emptiness problem for NFA, i. e., deciding for given NFA M_i, $1 \leq i \leq n$, whether or not $\bigcap_{i=1}^{n} L(M_i) = \emptyset$. To this end, let $M_1, \ldots, M_n$ be NFA over some alphabet Σ with $\# \notin \Sigma$. We define $\alpha = x_1 \# x_2 \# \ldots \# x_{n-1}$ and $\beta = x_2 \# x_3 \# \ldots \# x_n$, and we define the regular constraints $L_{x_i} = L(M_i)$. We note that the equation $\alpha = \beta$ is regular.

If there exists a word $w \in \bigcap_{i=1}^{n} L(M_i)$, then h with $h(x_i) = w$, $1 \leq i \leq n$, is a solution for $\alpha = \beta$, since $h(\alpha) = (w\#)^{n-2} w = h(\beta)$, and, furthermore, h satisfies the regular constraints. Let h be a solution for $\alpha = \beta$ that satisfies the regular constraints. This implies that $h(x_1) \# h(x_2) \# \ldots \# h(x_{n-1}) = h(x_2) \# h(x_3) \# \ldots \# h(x_n)$ and, since $|h(x_i)|_{\#} = 0$, $1 \leq i \leq n$, $h(x_1) = h(x_2) = \ldots = h(x_n)$ follows. Thus, $h(x_1) \in \bigcap_{i=1}^{n} L(M_i)$. □

Further we show that word equations without repeated variables can be solved in polynomial time also when regular constraints are used.

Theorem 4. *Solving word equations with regular constraints and without repeated variables can be done in polynomial time.*

Proof. Let $\alpha = \beta$ be a word equation without repeated variables and let L_x, $x \in \mathsf{var}(\alpha\beta)$, be regular constraints. Let $K_\alpha = K_{\alpha,1} \cdot K_{\alpha,2} \cdots K_{\alpha,|\alpha|}$, where $K_{\alpha,i} = L_{\alpha[i]}$, if $\alpha[i] \in \mathsf{var}(\alpha)$, and $K_{\alpha,i} = \{\alpha[i]\}$, otherwise. Let K_β be defined analogously. Since $\mathsf{var}(\alpha) \cap \mathsf{var}(\beta)$, there is a solution for $\alpha = \beta$ that satisfies the regular constraints if and only if $K_\alpha \cap K_\beta \neq \emptyset$. NFA M_α and M_β for K_α and K_β, respectively, can be easily constructed from the NFA for L_x, $x \in \mathsf{var}(\alpha\beta)$. Moreover, the size of M_α and M_β is polynomial in $|\alpha\beta|$ and the size of the NFA for L_x, $x \in \mathsf{var}(\alpha\beta)$. Thus, we can construct an NFA M for $K_\alpha \cap K_\beta$ of polynomial size and check $L(M) = \emptyset$ in time that is polynomial in the number of states of M. □

Word equations with only one variable can be solved in linear time (see Jeż [8]); their solvability problem is still in P when regular constraints are added.

Theorem 5. *Solving word equations with regular constraints and with only one variable can be done in polynomial time.*

Proof. Let $\alpha = \beta$ be a word equation with only one variable x and with regular constraint L_x, represented by an NFA M_x. For every $u \in \Sigma^*$, let h_u be the substitution defined by $h(x) = u$. By [2,8,13], we can obtain in polynomial time an NFA M_S accepting $S = \{u \mid h_u(\alpha) = h_u(\beta)\}$. Indeed, we know that S may consist of an infinite family of the form $(pq)^+p$ where pq is a primitve word, which occurs as a prefix of α, and another $\mathcal{O}(\log n)$ solutions whose length is $\mathcal{O}(n)$, and we can obtain S in polynomial time. Then we just produce in polynomial time the NFA accepting $L_x \cap S$ and check if it accepts the empty language or not. $\square$

Word equations with two variables can be solved in polynomial-time (see [3]). We shall see next that for word equations with regular constraints this is no longer the case (assuming $\mathsf{P} \neq \mathsf{NP}$). More precisely, solving equations with two variables and regular constraints is NP-hard, even if only one variable is repeated and the equations are variable disjoint. Moreover, we can show that the existence of an algorithm solving word equations with two variables and with regular constraints in time $2^{o(n+m)}$ (where n is the length of the equation and m is the size of the regular constraints) is very unlikely, since it would refute the well-known *exponential-time hypothesis* (ETH, for short).

We conduct a linear reduction from 3-Sat to the problem of solving word equations with regular constraints.[2] Let $C = \{c_1, c_2, \ldots, c_m\}$ be a Boolean formula in conjunctive normal form (CNF) with 3 literals per clause over the variables $\{v_1, v_2, \ldots, v_n\}$. We first transform C into a CNF C' such that C is satisfiable if and only if C' has an assignment that satisfies exactly one literal per clause (in the following, we call such an assignment a *1-in-3 assignment*). For every i, $1 \leq i \leq m$, we replace $c_i = \{y_1, y_2, y_3\}$ by 5 new clauses

$$\{y_1, z_1, z_2\}, \{y_2, z_2, z_3\}, \{z_1, z_3, z_4\}, \{z_2, z_5, z_6\}, \{y_3, z_5\},$$

where z_i, $1 \leq i \leq 6$, are new variables.[3] We note that C' has $5m$ clauses and $n + 6m$ variables. Next, we obtain C'' from C' by adding, for every i, $1 \leq i \leq n$, a new clause $\{v_i, \widehat{v}_i\}$, where $\widehat{v}_i$ is a new variable, and we replace all occurrences of $\overline{v_i}$ (i. e., the variable v_i in negated form) by $\widehat{v}_i$. The following proposition is immediate.

Proposition 2. *There is a satisfying assignment for C if and only if C'' has a 1-in-3 assignment. Furthermore, C'' has no negated variables, C'' has $5m + n$ clauses and $2n + 6m$ variables.*

For the sake of convenience, we set $n' = 2n + 6m$, $m' = 5m + n$, $C'' = \{c'_1, c'_2, \ldots, c'_{m'}\}$ and let $\{v'_1, v'_2, \ldots, v'_{n'}\}$ be the variables of C''. Furthermore, for every i, $1 \leq i \leq n'$, let k_i be the number of occurrences of variable v'_i in C''.

Next, we transform C'' into a word equation with regular constraints as follows. Let $\Sigma = \{v'_1, v'_2, \ldots, v'_{n'}, \#\}$ and let the equation $\alpha = \beta$ be defined by $\alpha = (x_1\,\#)^{n'-1}\,x_1$ and $\beta = x_2$. For the variables x_1 and x_2, we define the following regular constraints over Σ:

$$L_{x_1} = \{w \mid |w| = m', w[i] \in c'_i, 1 \leq i \leq m'\},$$
$$L_{x_2} = \{u_1 \# u_2 \# \ldots \# u_{n'} \mid u_i \in (\Sigma \setminus \{\#\})^*, |u_i|_{v'_i} \in \{k_i, 0\}, 1 \leq i \leq n'\}.$$

Proposition 3. *There are DFA M_{x_1} and M_{x_2} accepting the languages L_{x_1} and L_{x_2}, respectively, with $5m + n + 2$ and $21m + 5n + 1$ states, respectively.*

Proof. To construct a DFA for L_{x_1} with $m' + 2 = 5m + n + 2$ states is trivial (note that we need a trap state). A DFA for L_{x_2} needs to count k_1 occurrences of v'_1 with allowing occurrences of $\Sigma \setminus \{v'_1, \#\}$ to be read in between, then the same happens with respect to v'_2 and so on. The automaton can move from one such counting part to the next with a transition that reads $\#$ and, as long as no occurrences of v'_i have been read (i. e., in the very first state of a counting part), the automaton can also read an occurrence of $\#$ and move to the next counting part, which implements the case that an u_i contains no occurrence of v'_i. Each counting part has $k_i + 1$ states and we need one trap state; thus, the automaton has $(\sum_{i=1}^{n'} k_i + 1) + 1 = 3m' + n' + 1 = 21m + 5n + 1$ states. $\square$

[2] In order to prove NP-hardness, a simpler production would suffice, but we need a linear reduction in order to obtain the ETH lower bound.

[3] Note that this is just the reduction used by Schaefer [17] in order to reduce 3-Sat to 1-in-3 3Sat. We recall it here to observe that this reduction is linear.

By definition, only NFA are required to represent the regular constraints, but our use of DFA here points out that the following hardness result (and the ETH lower bound) also holds for the case that we require the regular constraints to be represented by DFA. So the hardness of the problem does not result from the fact that NFA can be exponentially smaller than DFA.

Lemma 2. *The Boolean formula C'' has a 1-in-3 assignment if and only if $\alpha = \beta$ has a solution that satisfies the regular constraints L_{x_1} and L_{x_2}.*

Proof. We start with the *only if* direction. To this end, let $\pi : \{v'_1, v'_2, \dots, v'_n\} \to \{0, 1\}$ be a 1-in-3 assignment for C'', where, for every i, $1 \leq i \leq m'$, y_i is the unique variable with $y_i \in c'_i$ and $\pi(y_i) = 1$. Let h be a substitution defined by $h(x_1) = y_1 y_2 \dots y_{m'}$ and $h(x_2) = (h(x_1)\,\#)^{n-1}\, h(x_1)$. Obviously, h is a solution for $\alpha = \beta$, $h(x_1) \in L_{x_1}$ and, since every v'_i has either 0 occurrences in $h(x_1)$ (in case that $\pi(v'_i) = 0$) or k_i occurrences (in case that $\pi(v'_i) = 1$), also $h(x_2) \in L_{x_2}$.

For the *if* direction, let h be a solution for $\alpha = \beta$ that satisfies the regular constraints. Consequently, $h(x_1) = y_1 y_2 \dots y_{m'}$, where $y_i \in c'_i$, $1 \leq i \leq m'$, and, furthermore, for every i, $1 \leq i \leq n$, $|h(x_2)|_{v'_i} \in \{k_i, 0\}$. This directly implies that $\pi : \{v'_1, v'_2, \dots, v'_n\} \to \{0, 1\}$, defined by $h(v'_i) = 1$ if $|h(x_2)|_{v'_i} = k_i$ and $h(v'_i) = 0$ if $|h(x_2)|_{v'_i} = 0$, is a 1-in-3 assignment for C''. $\square$

The exponential-time hypothesis, mentioned above, roughly states that 3-Sat cannot be solved in subexponential-time. For more informations on the ETH, the reader is referred to Chapter 14 of the textbook [1]. For our application of the ETH, it is sufficient to recall the following result.

Theorem 6 (Impagliazzo et al. [7])**.** *Unless ETH fails,* 3-Sat *cannot be solved in time $2^{o(n+m)}$, where n is the number of variables and m is the number of clauses.*

The reduction from above implies that a subexponential algorithm for solving word equations with two variables and regular constraints can be easily turned into a subexponential algorithm for 3-Sat; thus, the existence of such an algorithm contradicts ETH.

Theorem 7. *Solving word equations with two variables and with regular constraints is* NP*-hard, even if only one variable is repeated and the equations are variable disjoint. Furthermore, unless ETH fails, such word equations cannot be solved in time $2^{o(n+m)}$ (where n is the length of the equation and m is the size of the regular constraints).*

Proof. Since the reduction from above can be carried out in time $\mathcal{O}(n + m)$ (where n and m is the number of variables and clauses, respectively), Lemma 2 together with the fact that 3-Sat is NP-hard shows that solving word equations with two variables and with regular constraints is NP-hard, even if only one variable is repeated and the equations are variable disjoint.

In order to prove the second statement, we assume to the contrary that there is an algorithm solving word equations over two variables with regular constraints in time $2^{o(n+m)}$, where n is the size of the equation and m is the sum of the states of the DFA. Then an algorithm that, given a 3-CNF formula with k variables and ℓ clauses, first carries out the reduction from above in order to obtain a word equation of size $4k + 12\ell$ with regular constraints of size $26\ell + 6k + 3$ and then solves the solvability problem for this word equation in time $2^{o(10k+38\ell+3)} = 2^{o(k+\ell)}$ is an algorithm that solves 3-Sat in time $2^{o(k+\ell)}$, where ℓ is the number of clauses and k is the number of variables. $\square$

4.1 Bounded Word Equations

We first note that bounded word equations can be considered as a special case of word equations with regular constraints, since the bound m functions as regular constraints of the form $\{w \in \Sigma^* \mid |w| \leq m\}$ for every variable. However, there is an important difference: the length of a binary encoding of m is logarithmic in the size of an NFA for $\{w \in \Sigma^* \mid |w| \leq m\}$; thus, NP-hardness of a class of bounded word equations does not necessarily carry over to word equations with regular constraints. The solvability problem for a class of bounded word equations is NP*-hard in the strong sense*, if the NP-hardness remains if m is given in unary. The following results are strongly connected to (but cannot be directly derived from) the results of [16].

Theorem 8. *Solving bounded word equations is* NP*-hard (in the strong sense), even for equations* $\alpha = \beta$ *satisfying* $|\mathsf{var}(\alpha)| = 1$*,* $\mathsf{var}(\alpha) \cap \mathsf{var}(\beta) = \emptyset$ *and* β *is regular.*

Proof. We reduce from the shortest common superstring problem, i. e., deciding for given $k \in \mathbb{N}$ and strings $v_1, v_2, \ldots, v_n \in \Sigma^*$ whether there is a string u with $|u| \leq k$ that contains each v_i as a factor. Let $v_1, v_2, \ldots, v_n \in \Sigma^*$, $k \in \mathbb{N}$ be an instance of the shortest common superstring problem. Furthermore, let $\#$ be a new symbol, i. e., $\# \notin \Sigma$. We construct a word equation $\alpha = \beta$, where

$$\begin{array}{lllllllll} \alpha & = & x & \# & x & \# & \ldots & \# & x\,, \\ \beta & = & y_1 v_1 y'_1 & \# & y_2 v_2 y'_2 & \# & \ldots & \# & y_n v_n y'_n\,. \end{array}$$

Furthermore, we let k be the upper bound on the substitution word lengths.

If there exists a word $w \in \Sigma^*$ with $|w| \leq k$ and, for every i, $1 \leq i \leq n$, $w = u_i v_i u'_i$, then we define a substitution h by $h(x) = w$, $h(y_i) = u_i$ and $h(y'_i) = u'_i$, $1 \leq i \leq n$. Obviously, h satisfies the length bound and, for every i, $1 \leq i \leq n$, $h(x) = h(y_i v_i y'_i)$; thus, $h(\alpha) = h(\beta)$.

Let h be a solution for $\alpha = \beta$ that satisfies the length bound. We observe that since $h(\beta)$ contains every v_i as a factor, also $h(\alpha) = h(x)\#h(x)\#\ldots\#h(x)$ contains every v_i as a factor and, furthermore, since $|v_i|_\# = 0$, $1 \leq i \leq n$, every v_i is also a factor of $h(x)$. Thus, $|h(x)| \leq k$ and $h(x)$ contains every v_i, $1 \leq i \leq n$, as a factor.

For the shortest common superstring problem, we can assume that $k \leq \sum_{i=1}^{n} |v_i|$, since otherwise $v_1 v_2 \ldots v_n$ would also be a solution. Consequently, we can assume that k is given in unary, so solving bounded word equations of the form mentioned in the statement of the theorem is NP-hard in the strong sense. □

Due to the strong NP-hardness in Theorem 8, we can conclude the following.

Corollary 2. *Solving word equations with regular constraints is* NP*-hard, even for equations* $\alpha = \beta$ *satisfying* $|\mathsf{var}(\alpha)| = 1$*,* $\mathsf{var}(\alpha) \cap \mathsf{var}(\beta) = \emptyset$ *and* β *is regular.*

By using 1 as the bound on the substitution words and by a minor modification of the reduction for Theorem 2, we can obtain a hardness reduction for bounded regular word equations.

Corollary 3. *Solving bounded regular word equations is* NP*-hard.*

Proof. We modify the reduction from Section 3 as follows. Let

$$u = \alpha_1 \star \ldots \star \alpha_n \star (\odot^k) \diamond \alpha'_1 \diamond \ldots \diamond \alpha'_n\,,$$
$$v = \beta_1 \star \ldots \star \beta_n \star (\odot^k) \diamond \beta'_1 \diamond \ldots \diamond \beta'_n\,,$$

where, α_i and β_i, $1 \leq i \leq n$, are defined as in the original reduction, $\alpha'_i = \diamond \# \mathtt{a}^4 \# \#$ and $\beta'_i = y_{i,1} y_{i,2} y_{i,3} y_{i,4} y_{i,5} \# x_{i,\wp_1(i)} \ldots x_{i,\wp_4(i)} \# y'_i$, $1 \leq i \leq n$, and $k = 13n$. As length bound on the substitution words, we define $m = 1$.

A perfect code for $\mathcal{G}$ translates into a solution for $u = v$ in an analogous way as in the proof of Lemma 1 (the only difference is that, because of the length bound 1, we need the 5 variables $y_{i,1} y_{i,2} \ldots y_{i,5}$ in order to simulate variable y_i).

For the other direction, we show that a solution h for $u = v$ that satisfies the length bound m must also satisfy $h(u_1) = h(v_1)$ and $h(u_2) = h(v_2)$, where $u_1 = \alpha_1 \star \ldots \star \alpha_n \star$, $u_2 = \diamond \alpha'_1 \diamond \ldots \diamond \alpha'_n$, $v_1 = \beta_1 \star \ldots \star \beta_n \star$ and $v_2 = \diamond \beta'_1 \diamond \ldots \diamond \beta'_n$. It follows then in the same way as in the proof of Lemma 1 that a perfect code for $\mathcal{G}$ exists.

We first note that any solution h for $u = v$ that satisfies the bound m must also satisfy $|h(u_1)| \leq 5n$ and $|h(v_2)| \leq 13n$. Since $w = (\odot^k) u_2$ is a constant suffix and $w' = v_1(\odot^k)$ is a constant prefix of $h(u) = h(v)$ and both have length at least $13n$, $h(u_1)$ must be a prefix of w' and $h(v_2)$ must be a suffix of w. Since $|w|_\star = 0$ and $|w|_\diamond = 0$, we can conclude that $|h(x_{i,j})|_\star = 0$ and $|h(x_{i,j})|_\diamond = 0$, $1 \leq i, j \leq n$. Moreover, $h(u_1)$ is a prefix of $v_1(\odot^k) = \mathtt{a}\star \ldots \star \mathtt{a}\star(\odot^k)$, which means that $|h(u_1)|_\odot \geq 1$ implies that $h(u_1) = \mathtt{a}\star \ldots \star \mathtt{a}\star(\odot^{k'})$, with $1 \leq k' \leq k$, which is a contradiction. Consequently $h(x_{i,j}) \in \{\mathtt{a}, \varepsilon\}$. This directly implies that $h(u_1) = h(v_1)$ and $h(u_2) = h(v_2)$. □

4.2 Individual Alphabets

The least restrictive regular constraints are probably constraint languages of the form Γ^* for some $\Gamma \subseteq \Sigma$, i. e., word equations with individual alphabets, which we shall investigate in this section.

We first note that if $|\Sigma| = 1$, then general word equations and word equations with individual alphabets coincide and, furthermore, the solvability problem for word equations can be solved in polynomial-time, if $|\Sigma| = 1$.

Theorem 9. *Solving word equations can be done in polynomial time if $|\Sigma| = 1$.*

Proof. Let $\Sigma = \{\mathtt{a}\}$ and let $\alpha = \beta$ be a word equation. Since $|\Sigma| = 1$, the order of the constants and variables in α and β does not matter and, furthermore, for every $x \in \mathsf{var}(\alpha) \cap \mathsf{var}(\beta)$, we can simply remove $\min\{|\alpha|_x, |\beta|_x\}$ occurrences of x from both α and β. Consequently, we can reduce $\alpha = \beta$ to an equation of the form $(\Pi_{i=1}^{\ell}(y_i)^{m_i})\mathtt{a}^k = (\Pi_{i=1}^{\ell'}(z_i)^{m'_i})\mathtt{a}^{k'}$, where all y_i, z_j, $1 \leq i \leq \ell$, $1 \leq j \leq \ell'$, are distinct variables. Moreover, we can also replace every $(y_i)^{m_i}$ and $(z_i)^{m'_i}$ by a single variable y_i and z_i and regular constraints $L_{y_i} = \{a^k \mid k \mod m_i = 0\}$ and $L_{z_i} = \{a^k \mid k \mod m'_i = 0\}$, respectively. This yields an equations with regular constraints and without repeated variables, which, according to Theorem 4, can be solved in polynomial-time. $\square$

However, if $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ and $\{\mathtt{a}\}$ is used as individual alphabet for all variables, then solving word equations becomes NP-hard again, simply because the matching problem is already NP-hard for this case (as can be easily concluded from the reduction of Lemma 5 in [6]).

By using individual alphabets, the reduction for Theorem 2 can be easily transformed to a hardness reduction for the solvability problem of regular equations with individual alphabets.

Corollary 4. *Solving regular word equations with individual alphabets is NP-hard.*

Proof. We modify the reduction from Section 3 as follows. Let

$$u = \alpha_1 \star \ldots \star \alpha_n \star \odot \diamond \alpha'_1 \diamond \ldots \diamond \alpha'_n \,,$$
$$v = \beta_1 \star \ldots \star \beta_n \star \odot \diamond \beta'_1 \diamond \ldots \diamond \beta'_n \,,$$

where, α_i, and β_i, $1 \leq i \leq n$, are defined as in the original reduction, $\alpha'_i = \diamond \# \mathtt{a}^4 \# \#$ and $\beta'_i = y_i \# x_{i,\wp_1(i)} \ldots x_{i,\wp_4(i)} \# y'_i$, $1 \leq i \leq n$. As individual alphabets, we choose $\{\mathtt{a}\}$ for the variables $x_{i,j}$, $1 \leq i, j, \leq n$, and $\{\#, \mathtt{a}\}$ for the variables y_i, y'_i, $1 \leq i \leq n$. We note that the solution h obtained from a perfect code in the proof of Lemma 1 is also a solution for the modified equation $u = v$ that also satisfies the individual alphabets. On the other hand, any solution h for $u = v$ that satisfies the individual alphabets satisfies in particular $|h(x)|_\star = |h(x)|_\diamond = |h(x)|_\odot = 0$, for every $x \in \mathsf{var}(uv)$, which implies $h(\alpha_1 \star \ldots \star \alpha_n \star) = h(\beta_1 \star \ldots \star \beta_n \star)$ and $h(\diamond \alpha'_1 \diamond \ldots \diamond \alpha'_n) = h(\diamond \beta'_1 \diamond \ldots \diamond \beta'_n)$ and therefore the existence of a perfect code for $\mathcal{G}$ follows in the same way as in the proof of Lemma 1. $\square$

5 Conclusions

We conclude this work by summarising our main results and by suggesting some further research directions.

Firstly, the polynomial-time decidability of the matching problem for non-cross patterns does not carry over to non-cross equations (which also means that the concept of the scope coincidence degree, briefly mentioned in Section 1, will not help, since it is a generalisation of the non-cross concept), while for regular equations, this is still open, which constitutes the most important question left open here.

If we allow regular constraints, it is possible to prove hardness results for strongly restricted variants of the solvability problem, often including the regular case. More precisely, for general regular constraints, the solvability problem is PSPACE-complete, even for regular equations (Theorem 3), and NP-hard for variable disjoint equations with only one repeated variable and two variables in total (Theorem 7). Especially this latter result points out a drastic difference in terms of complexity between general word equations and equations with regular constraints:

both the tractable cases of equations with only two variables or with only one repeated variable and at least one non-repeated variable on both sides (Theorem 1) become NP-hard if we allow regular constraints.[4] Moreover, the case with only one repeated variable remains intractable, even if the constraints are only bounding the length of the substitution words (Theorem 8). In particular, even if it turns out that, for some k, $k \geq 3$, or even for all constant k, general word equations with at most k variables can be solved in polynomial-time, Theorem 7 severely limits their practical application, since it shows that these polynomial-time algorithms cannot cope with regular constraints (unless $\mathsf{P} = \mathsf{NP}$).

As for regular equations, allowing a system of only two equations (and no further constraints), allowing bounds on the substitution words or allowing individual alphabets is enough to make the solvability problem NP-hard.

Our choice of restrictions for word equations is motivated by polynomial-time solvable cases of the matching problem. In order to obtain tractable classes of word equations, it might be worthwhile to strengthen the concept of non-cross and regularity by requiring $\alpha\beta$ to be regular or non-cross, instead of only requiring this for α and β separately. Another possible further restriction would be to require the order of the variables on the left and on the right side to be the same (e. g., $x_1\mathtt{ab}x_2\mathtt{c}x_3 = x_1\mathtt{c}x_3$ is *ordered regular*, while $x_1\mathtt{ab}x_2\mathtt{c}x_3 = x_3\mathtt{c}x_2$ is not). In this regard, it is interesting to note that the patterns produced by the reduction of Theorem 2 are not ordered non-cross (and not ordered regular for the corresponding corollaries), while Theorem 3, the PSPACE-completeness of solving word equations with regular constraints, also holds for ordered regular equations. Additionally requiring $\mathsf{var}(\alpha) = \mathsf{var}(\beta)$ for ordered regular equations would be a further restriction that might be useful.

Acknowledgements

We are indebted to Artur Jeż for valuable discussions. Markus L. Schmid gratefully acknowledges partial support for this research from DFG, that in particular enabled his visit at the University of Kiel. Florin Manea gratefully acknowledges the support for this research from the DFG project 218587403. Dirk Nowotka gratefully acknowledges the support for this research from the DFG project 181615770.

References

[1] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms.* Springer International Publishing AG Switzerland, 2015.

[2] R. Dabrowski and W. Plandowski. On word equations in one variable. *Algorithmica*, 60(4):819–828, 2011.

[3] R. Dąbrowski and W. Plandowski. Solving two-variable word equations. In *Proc. 31th International Colloquium on Automata, Languages and Programming, ICALP 2004*, volume 3142 of *LNCS*, pages 408–419, 2004.

[4] H. Fernau, F. Manea, R. Mercaş, and M.L. Schmid. Pattern matching with variables: Fast algorithms and new hardness results. In *Proc. 32nd Symposium on Theoretical Aspects of Computer Science, STACS 2015*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 302–315, 2015.

[5] H. Fernau and M. L. Schmid. Pattern matching with variables: A multivariate complexity analysis. *Information and Computation*, 242:287–305, 2015.

[6] H. Fernau, M. L. Schmid, and Y. Villanger. On the parameterised complexity of string morphism problems. *Theory of Computing Systems*, 2015. http://dx.doi.org/10.1007/s00224-015-9635-3.

[4]For the latter case, note that in the reduction of Theorem 7, we can add a non-repeated variable with regular constraint $\emptyset$ to the left side.

[7] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63:512–530, 2001.

[8] A. Jeż. One-variable word equations in linear time. *Algorithmica*, 74:1–48, 2016.

[9] A. Jeż. Recompression: A simple and powerful technique for word equations. *Journal of the ACM*, 63, 2016.

[10] J. Kratochvíl and M. Křivánek. On the computational complexity of codes in graphs. In *Proc. 13th Symposium on Mathematical Foundations of Computer Science, MFCS 1988*, volume 324 of *LNCS*, pages 396–404, 1988.

[11] M. Lothaire. *Algebraic Combinatorics on Words.* Cambridge University Press, Cambridge, New York, 2002.

[12] G.S. Makanin. The problem of solvability of equations in a free semigroup. *Matematicheskii Sbornik*, 103:147–236, 1977.

[13] S. Eyono Obono, P. Goralcik, and M. N. Maksimenko. Efficient solving of the word equations in one variable. In *Mathematical Foundations of Computer Science 1994, 19th International Symposium, MFCS'94, Kosice, Slovakia, August 22 - 26, 1994, Proceedings*, pages 336–341, 1994.

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

[15] D. Reidenbach and M. L. Schmid. Patterns with bounded treewidth. *Information and Computation*, 239:87–99, 2014.

[16] J. M. Robson and V. Diekert. On quadratic word equations. In *Proc. 16th Annual Symposium on Theoretical Aspects of Computer Science, STACS 1999*, volume 1563 of *Lecture Notes in Computer Science*, pages 217–226, 1999.

[17] T. J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th Annual ACM Symposium on Theory of Computing, STOC 1978*, pages 216–226. ACM, 1978.