Theory of Computing Systems
https://doi.org/10.1007/s00224-023-10154-8

A Closer Look at the Expressive Power of Logics Based on Word Equations

Joel Day[1] · Vijay Ganesh[2] · Nathan Grewal[2] · Matthew Konefal[1] · Florin Manea[3]

Accepted: 13 November 2023

Abstract
Word equations are equations $\alpha \doteq \beta$ where α and β are words consisting of letters from some alphabet Σ and variables from a set X. Recently, there has been substantial interest in the context of string solving in logics combining word equations with other kinds of constraints on words such as (regular) language membership (regular constraints) and arithmetic over string lengths (length constraints). We consider the expressive power of such logics by looking at the set of all values a single variable might take as part of a satisfying assignment for a given formula. Hence, each formula-variable pair defines a formal language, and each logic defines a class of formal languages. We consider logics arising from combining word equations with either length constraints, regular constraints, or both. We also consider word equations with visibly pushdown language membership constraints as a generalisation of the combination of regular and length constraints. We show that word equations with visibly pushdown membership constraints are sufficient to express all recursively enumerable languages and hence satisfiability is undecidable in this case. We then establish a strict hierarchy involving the other combinations. We also provide a complete characterisation of when a thin regular language is expressible by word equations (alone) and some further partial results for regular languages in the general case.

Keywords Word equations · String constraints · String solving · Regular languages · Expressibility

1 Introduction

Logical theories based on strings (or words) over a finite alphabet have been an important topic of study for decades [39]. Connections to arithmetic (see e.g. [43]) and interest in fundamental questions from algebra about free groups and semigroups underpinned interest in theories involving concatenation and equality. These two elements combined lead to word equations: a word equation is an equality of the form

Extended author information available on the last page of the article

Published online: 11 December 2023

$\alpha \doteq \beta$, where α and β are terms obtained by concatenating variables and concrete words over some finite alphabet. For example, if x and y are variables, and our alphabet is $\Sigma = \{a, b\}$, then $xaby \doteq ybax$ is a word equation. Its solutions are substitutions for the variables unifying the two sides: $x \to bb$, $y \to b$ would be one such solution in the previous example.

The existential theory of a finitely generated free monoid Σ^* consists of formulas made up of Boolean combinations of word equations. In fact, the problem of deciding whether a formula in this fragment is true is equivalent to determining satisfiability of word equations, since any such formula can be transformed into a single word equation without disrupting satisfiability (see [32, 39]). It was originally hoped that the problem of deciding if a word equation has a solution could facilitate an undecidability proof for Hilbert's famous Tenth Problem by providing an intermediate step between Diophantine equations and the computations of Turing Machines. Famously, however, this endeavour failed when Makanin showed in 1977 that satisfiability of word equations can be decided algorithmically [40].

Since then, several improvements to the algorithm proposed by Makanin have been discovered. Two decades later, Plandowski [42] was the first to show that the problem could be solved in PSPACE, and this has later been refined to nondeterministic linear space by Jeż via the Recompression technique [30]. It was shown in [44] and [17] (see also Chapter 12 of [39]) that the problem remains decidable even when the variables are constrained by regular languages, limiting the possible substitutions. On the other hand, if length constraints (requiring that some pairs of variables are substituted for words of the same length) are permitted, then it remains a long-standing open problem as to whether or not the problem is decidable. Recalling the earlier example of a word equation $xaby \doteq ybax$, we might ask whether solutions exist such that x and y have the same length, which in this case is clearly not possible due to the resulting alignment of the ab and ba factors.

Word equations and logics involving them (or strings more generally) have remained a topic of interest within the Theoretical Computer Science community, in particular due to their fundamental role within Combinatorics on Words and Formal Languages, and more recently due to interest from the Formal Methods community. The latter can be attributed to increasing popularity and influence of software tools called string-solvers, which seek to algorithmically solve constraint satisfaction problems involving strings. In this setting, a string constraint is a property or piece of information about an unknown string and the string solvers try to determine whether strings exist which satisfy combinations of string constraints of various types. Word equations, regular language membership, and relations between lengths are all among the most prominent building blocks of string constraints, and when combined are sufficient to model several others. For example, the "$\mathsf{substring}(x, y)$" constraint expressing that x occurs somewhere inside y can be modelled by the word equation $y \doteq z_1 x z_2$, where z_1, z_2 are additional variables, while the "$\mathsf{index\_of}(x, y)$" constraint returning the position

of an occurrence of x in y can be modelled by using the length of z_1 in the previous equation.

Another application is Database theory where string-solvers are also useful, e.g. for evaluating path queries in graph databases [7, 21] and in connection with document spanners [22, 23]. Recently, a finite-model version of the theory of concatenation was considered in this context [24].

A wealth of string-solvers is now available [1, 2, 8, 10, 31, 33, 41, 46], with a variety being optimized for specific applications, alongside those intended as being more general-purpose (see also [6, 26] for an overview). However, the underlying task of determining the satisfiability of string constraints remains a challenging problem and implementations rely heavily on search heuristics.

Motivated in part by the applications in string-solving, and by the desire to make progress on seemingly very difficult open theoretical problems, various results exist which investigate the computability and/or complexity of the satisfiability problem for combinations of string constraints. The works [25, 34–37] identify restrictions on word equations which result in a decidable satisfiability problem even when length constraints are present. Several further ways of augmenting word equations (i.e., additional predicates or constraints on the variables), are discussed and shown to be undecidable in [11–14, 16, 27, 28]. An immediate consequence of [43] is that allowing arbitrary existential and universal quantification of variables leads to an undecidable theory, and in fact this holds even for very restricted cases with a single quantifier alternation and a constant number of quantifiers (see [19, 20]).

Nevertheless, despite progress on satisfiability problems such as those mentioned above, and while the expressive power and computational properties of prominent language classes such as the regular and context free languages are well understood, little is known about the true expressive power of word equations and of string logics involving word equations in conjunction with other common types of string constraints. This is both a barrier to settling open problems involving satisfiability problems, such as for word equations with length constraints, and also a limit in terms of general understanding in the context of string solving: often simply finding a solution to one constraint is not enough and the set of solutions must be considered more generally in order to account for other constraints which might be present, or to determine that no solution exists.

In [11, 18], it was shown that, on the one hand, length is not definable using equality and concatenation alone, and, on the other hand, that if predicates are present which facilitate the comparison of the number of occurrences of at least two different letters, then connections to arithmetic over natural numbers and Diophantine sets can be made which lead to undecidable satisfiability problems. Karhumäki, Mignosi and Plandowski [32] considered explicitly the question of which formal languages are expressible as the set of solutions to a word equation, projected onto a single variable. Their techniques can be used to show that several simple languages like $\{a^n b^n \mid n \in \mathbb{N}\}$ and $\{a, b\}^* c$ are not expressible. However, they do not consider additional constraints, and thus their results in many cases are not directly applicable to our setting.

Our Contributions

We consider the question of expressibility of formal languages in the sense of [32] in a number of logics, introduced in detail in Section 2, which involve word equations alongside some of the most commonly associated constraints. The logics, summarised below are all quantifier-free and consist of the typical Boolean connectives $\wedge$, $\vee$ and $\neg$, and different combinations of word equations and other kinds of string constraints.

- WE - word equations only
- WE + REG - word equations and regular language membership (regular constraints)
- WE + LEN - word equations and linear arithmetic over lengths of variables (length constraints)
- WE + LEN + REG - word equations, regular language membership, and linear arithmetic over lengths of variables
- WE + VPL - word equations and visibly pushdown language membership

In Section 3, we consider the relationships between the classes of languages expressible in each of the logics listed above. For each logic $\mathfrak{T}$, denote the corresponding classes of languages expressible in that logic by $\mathcal{L}(\mathfrak{T})$. From existing results we can infer that $\mathcal{L}(\text{WE}) \subset \mathcal{L}(\text{WE} + \text{REG}), \mathcal{L}(\text{WE} + \text{LEN})$. In particular, the tools developed in [32] are sufficient since this strict inclusion requires showing inexpressibility for word equations only. Moreover, it is easily seen that $\mathcal{L}(\mathfrak{T})$ contains only recursively enumerable languages for each of the listed logics $\mathfrak{T}$. In order to settle the other relationships, we adapt and extend the tools from [32] in order to work in the context of word equations combined with additional constraints such as regular language membership and length arithmetic. By doing so, we are able to completely characterise the relationships and provide the following strict hierarchy, in which $\mathcal{L}(\text{WE} + \text{REG})$ and $\mathcal{L}(\text{WE} + \text{LEN})$ are incomparable:

$$\mathcal{L}(\text{WE}) \subset \begin{matrix}\mathcal{L}(\text{WE+LEN}),\\ \mathcal{L}(\text{WE+REG})\end{matrix} \subset \mathcal{L}(\text{WE+LEN+REG}) \subset \mathcal{L}(\text{WE+VPL}) = \text{RE}.$$

The inclusion $\mathcal{L}(\text{WE} + \text{LEN} + \text{REG}) \subset \text{RE}$ is relevant to the open problem regarding the decidability status for satisfiability in this logic, which is of importance in the field of string solving. In particular, if the inclusion were not strict, but rather an equality, then satisfiability would necessarily be undecidable. Similarly, there are many recursively enumerable languages which, if shown to be expressible in WE + LEN or WE + LEN + REG, would result in the same negative result e.g. by allowing a reduction from Hilbert's 10th problem, motivating a need for techniques such as ours for showing inexpressibility in this case.

The equivalence $\mathcal{L}(\text{WE} + \text{VPL}) = \text{RE}$ is also worthy of further comment. Standard proofs can be adapted to show that the logic WE + CF combining word equations with deterministic context free language membership, can express all recursively enumerable languages, and it is well known in the string solving community that this combination induces an undecidable satisfiability problem. This is unsurprising since intersection-emptiness is undecidable for deterministic context free languages. On

the other hand, WE + LEN + REG is powerful enough to express many common string constraint types, but the decidability of satisfiability is unknown. Visibly pushdown languages, although not a class commonly used explicitly in string constraints, offer an appealing intermediate logic to study. Firstly, when considered in isolation, they offer very good computational properties: they have many of the desirable closure (union, intersection, complement) and algorithmic properties of the regular languages, in contrast to many other classes of languages falling between regular and context free. Secondly, they directly generalise the regular languages, but with sufficient memory capabilities to model length comparisons, and thus when combined with word equations, directly generalise the combination of length constraints and regular constraints. Unfortunately, our result is negative in the sense that there is no hope of a decidable satisfiability problem for word equations with visibly pushdown language membership constraints, as this combination is expressive enough to capture all recursively enumerable languages. Nevertheless, this result provides a tighter upper limit on the combinations of constraints for which satisfiability is undecidable, and moreover does this for a class of language membership constraints for which intersection-emptiness is decidable.

In addition to the inexpressibility results in Section 3 and the resulting relations between the classes of expressible languages, we are also able to show undecidability of the following problem in several cases:

Given a language L expressible in one of the logics $\mathfrak{T}_1$ listed above, and given a less expressive logic $\mathfrak{T}_2$, is L expressible in $\mathfrak{T}_2$?

In this context, L is given as a formula ψ from $\mathfrak{T}_1$ and a variable x in ψ. The question then asks whether ψ can be simplified to a weaker logic (or set of string constraints) without affecting the set of possible assignments for the variable x. This problem is therefore of interest in practical string solving applications, where much of the complexity arises from dealing with complex combinations of constraints, and removing one type of constraints from a formula can be an effective preprocessing step.

In Section 4 we concentrate again on word equations which are not extended by other kinds of constraints, or in other words, on the logic WE. In particular, we consider the relationship between the class of languages expressible in WE and the class of regular languages. It was already shown that these classes are incomparable in [32]. We show firstly, and perhaps rather surprisingly, that it is undecidable whether a language expressed in WE is regular. This provides a negative result in the same vein as those at the end of Section 3, and related to the simplification problem mentioned above. It provides some evidence of the complexity of the class of languages expressed by word equations, or at least of this means of representing them.

We then turn our attention to the converse problem of when a regular language is expressible. In this case, the representation (i.e. a finite automaton or regular expression) is arguably much simpler and there is more reason to expect a positive result. Although we are not able to settle the problem completely, we divide our analysis into two cases depending on whether or not the language in question is "thin" (so, whether there is a word which does not appear as a factor of any word in the language). Due to technical reasons related to possible representations of the language and to the tools

we use to show inexpressibility, the "thin" case seems the easier to address and indeed we are able to give a complete characterisation of when a thin regular language is expressible in WE. This in turn gives a positive result for the corresponding decision problem. When the language is not thin, we are able to provide some partial results which shed some light on which kinds of languages are (not) expressible in WE, how techniques for showing inexpressibility can be applied in this general setting, and the difficulties inherent to settling the problem completely.

This work extends the conference paper [15]. In particular, Sections 1-3 build on work presented in [15] by adding and updating proofs and explanations which were omitted partially or entirely in that work. Sections 4.1 and 4.2 are entirely new to the present work.

2 Preliminaries

Let $\mathbb{N} = \{1, 2, 3, \ldots\}$ and $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$. The integers are denoted by $\mathbb{Z}$. An alphabet $\Sigma = \{a_1, a_2, \ldots, a_n\}$ is a set of symbols, or letters. We denote by Σ^* the set of all words obtained by concatenating letters from Σ including the empty word, which we denote ε. In other words, Σ^* is the free monoid generated by Σ together with the operation of concatenation. Similarly, Σ^+ is the free semigroup $\Sigma^* \backslash \{\varepsilon\}$. For a word $w = uvx$, where $u, v, x \in \Sigma^*$, we say that u is a prefix of w, v is a factor of w and x is a suffix of w. If u (resp. v, x) is not equal to w, then it is a proper prefix (resp. factor/suffix). The length of a word w is written $|w|$. For words $u, v \in \Sigma^*$ we denote their concatenation either by $u \cdot v$ or simply as uv. For $w \in \Sigma^*$ and i, j satisfying $1 \leq i \leq j \leq |w|$, we denote by $w[i]$ the i^{th} letter of w and by $w[i : j]$ the factor $w[i]w[i+1] \ldots w[j-1]$. For $1 \leq i \leq |w|$, we call i a position of w, and associate the position with the letter $w[i]$. We denote n repetitions of a word w by w^n. A word w is primitive if w cannot be written in the form $w = x^n$ where $x \in \Sigma^+$ and $n \neq 1$. For each word $w \in \Sigma^+$ there is a unique primitive word u such that $w = u^n$ for some $n \in \mathbb{N}$. We call u the primitive root of w. Two words $w_1, w_2 \in \Sigma^*$ are conjugate if there exist $u, v \in \Sigma^*$ such that $w_1 = uv$ and $w_2 = vu$. The words w_1 and w_2 commute if $w_1 w_2 = w_2 w_1$.

Given a set of variables $X = \{x_1, x_2, \ldots\}$ and an alphabet Σ, a word equation is a pair $(\alpha, \beta) \in (X \cup \Sigma)^* \times (X \cup \Sigma)^*$, usually written as $\alpha \doteq \beta$. A solution to a word equation is a substitution of the variables for words in Σ^* such that both sides of the equation become identical. Formally, we model solutions as morphisms. That is, we say a substitution is a (homo)morphism $h : (X \cup \Sigma)^* \to \Sigma^*$ satisfying $h(a) = a$ for all $a \in \Sigma$, and a solution to a word equation $\alpha \doteq \beta$ is a substitution h such that $h(\alpha) = h(\beta)$. We recall the following canonical lemmas concerning simple word equations. The first follows from the so-called Defect Theorem (see e.g. Theorem 1.2.5 and Corollary 1.2.6 in [38]).

Lemma 1 *([38]) Let x, y be variables and let $\alpha, \beta \in \{x, y\}^+$ be distinct words. If $h : \{x, y\}^* \to \Sigma^*$ is a solution to the word equation*

$$\alpha \doteq \beta,$$

then there exists $w \in \Sigma^*$ *such that* $h(x), h(y) \in \{w\}^*$. *Consequently, if* h *is a solution to* $\alpha \doteq \beta$ *then* $h(x), h(y)$ *commute.*

Lemma 2 *(Theorem 1.3.4 in* [38]*) Let* x, y, z *be variables. Then* $h : \{x, y, z\}^* \to \Sigma^*$ *is a solution to the word equation*

$$xz \doteq zy$$

if and only if either $h(x) = h(y) = \varepsilon$ *or there exist* $u, v \in \Sigma^*$ *and* $n \in \mathbb{N}_0$ *such that* $h(x) = uv$, $h(y) = vu$ *and* $h(z) = u(vu)^n$.

2.1 Logics Based on Word Equations

We refer to [29] for standard definitions and well-known results from formal language theory. We denote the classes of regular, decidable (recursive) and recursively enumerable languages as REG, REC and RE respectively. Following [32], given a word equation E and a variable x, the language expressed by x in E is the language $\{h(x) \mid h$ is a solution to $E\}$. If a language $L \subseteq \Sigma^*$ is expressed by some variable in some word equation, we say that L is expressible by word equations. Note that the language expressed is dependent on the underlying alphabet Σ, which may contain letters other than those explicitly present in the word equation. Generally, we are interested in a more general setting in which word equations may occur as atoms in a larger formula, possibly with other types of atoms providing further constraints on the variable. We define the following logics based on word equations:

Definition 1 *Let* WE *be the set of formulas adhering to the following syntax:*

- *A word equation is a* WE*-formula.*
- *For* WE*-formulas* ψ_1, ψ_2*, the Boolean combinations* $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$ *and* $\neg\psi_1$ *are all* WE*-formulas.*

Note that formulas in WE are quantifier-free. If quantifiers are permitted in addition, the resulting logic is often referred to as the theory of concatenation. Formulas in WE are evaluated using the natural semantics. That is, assignments h map variables to words in Σ^*. A subformula consisting of a single word equation E with variables $x_1, x_2, \ldots, x_k$ evaluates to true w.r.t. an assignment $h : \{x_1, x_2, \ldots, x_k\} \to \Sigma^*$ if h extends to a solution to E when interpreted as a substitution. Otherwise the subformula evaluates to false. Boolean combinations $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$ and $\neg\psi_1$ are all then evaluated in the usual way according to the truth values for ψ_1 and ψ_2.

For a WE-formula ψ containing a variable x, the language expressed by x in ψ is $\{h(x) \mid h$ is a satisfying assignment for $\psi\}$. If a language $L \subseteq \Sigma^*$ is expressed by a variable in some WE-formula, we say that L is expressible in WE.

Remark 1 *The class of languages expressible in* WE *(as well as classes arising from logics introduced later) is dependent on the underlying alphabet* Σ. *In general, in what follows we shall assume that* Σ *is some fixed, finite alphabet which is "sufficiently large" in the sense that* $|\Sigma| > c$ *for some constant* c. *As* c *need only be large enough to contain as many distinct letters as we use explicitly in our constructions, it can be considered "reasonably small" in the sense that the condition* $|\Sigma| > c$ *will typically*

be satisfied in practice (for example, in the case of string solvers which operate on a superset of the characters used in our proofs). In any specific cases where a "small" alphabet is required for a result to hold, we shall state this explicitly. Note also that it is often the case that results for larger alphabets Σ can be adapted also for cases when Σ is small, provided $|\Sigma| \geq 2$. However, for the sake of the exposition we do not focus on minimising the alphabet size needed for our results to hold.

We note the following result, based on well-known constructions (see e.g. also [38]).

Lemma 3 [32] *For any* WE*-formula ψ whose variables are $\{x_1, x_2, \ldots, x_k\}$, there exists a single word equation E containing the variables $x_1, x_2, \ldots, x_k$ and possible further additional variables $y_1, y_2, \ldots, y_\ell$ such that for any assignment $h : \{x_1, x_2, \ldots, x_k\} \to \Sigma^*$, h is a satisfying assignment for ψ if and only if there exists a solution $h' : \{x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_\ell\}^* \to \Sigma^*$ to E satisfying $h(x_i) = h'(x_i)$ for all $i, 1 \leq i \leq k$. Moreover, E can be computed from ψ.*

Corollary 1 [32] *A language is expressible by word equations if and only if it is expressible in* WE*. Moreover, it follows that languages expressible in* WE *are closed under concatenation, union and intersection.*

On the other hand, it was also shown in [32] that WE-expressible languages are not closed under complement. Specifically, in general the language expressed by x in E is not the complement of the language expressed by x in $\neg E$ due to the fact that if E contains other variables, then there could exist substitutions h_1, h_2 satisfying $h_1(x) = h_2(x) = w$ where h_1 is a solution to E and h_2 is not.

Lemma 3 is particularly useful as it allows us to switch between working with a single word equation or arbitrary WE-formulas depending on which form is more convenient.

We also define the following extensions of WE to allow two typical additional constraints occurring alongside word equations. The first adds regular language membership as atoms:

Definition 2 *Let* WE + REG *be the set of formulas adhering to the following syntax:*

- *A word equation is a* WE + REG*-formula.*
- *For a variable x, $x \in L$ is a* WE + REG*-formula where L is a regular language.*
- *For* WE + REG*-formulas ψ_1, ψ_2, the Boolean combinations $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$ and $\neg\psi_1$ are all* WE + REG*-formulas.*

The semantics of WE + REG extend those of WE by evaluating, for an assignment h, the subformula $x \in L$ as true if $h(x) \in L$ and false otherwise. We assume that the regular languages L are given by any of the typical representation methods: DFAs, NFAs or regular expressions. Since we do not concentrate on precise computational complexity in the current work, we can always assume that we can convert from one to the other where convenient.

The second extension to WE adds constraints on lengths of variables:

Definition 3 *For a variable x, we treat $|x|$ as a numerical variable taking values from $\mathbb{N}_0$ representing the length of the word x. Let* WE + LEN *be the set of formulas adhering to the following syntax:*

- *A word equation is a* WE + LEN*-formula.*
- *For any* $c_0, c_1, c_2, \ldots, c_k \in \mathbb{Z}$ *and variables* $x_1, x_2, \ldots, x_k$*, the linear equality* $c_0 + \sum_{1 \leq i \leq k} c_i |x_i| = 0$ *is a* WE + LEN *formula.*
- *For* WE + LEN*-formulas* ψ_1, ψ_2*, the Boolean combinations* $\psi_1 \wedge \psi_2$*,* $\psi_1 \vee \psi_2$ *and* $\neg\psi_1$ *are all* WE + LEN*-formulas.*

The semantics of WE + LEN extend those of WE by evaluating, for an assignment h, the subformula $c_0 + \sum_{1 \leq i \leq k} c_i |x_i| = 0$ as true if $c_0 + \sum_{1 \leq i \leq k} c_i |h(x_i)| = 0$ and false otherwise.

Remark 2 *In the definition above, only equality is included syntactically. However, we simulate a strict inequality* $c_0 + \sum_{1 \leq i \leq k} c_i |x_i| < 0$ *by introducing new string variables* y, z *and including the subformula* $1 - |y| = 0 \wedge c_0 + |y| + |z| + \sum_{1 \leq i \leq k} c_i |x_i| = 0$*. Since this subformula enforces* $|h(y)| = 1$ *and* $|h(z)| \geq 0$*, it is satisfied by an assignment* h *if and only if* $c_0 + \sum_{1 \leq i \leq k} c_i |h(x_i)| < 0$*. Similarly, we can simulate* $\ldots > 0$ *by inverting the constants and* $\ldots \neq 0$ *by a logical negation* $\neg(\ldots = 0)$*. Consequently,* WE + LEN *has the same expressive power as if arbitrary quantifier free Presburger arithmetic formulas are permitted over the length-variables* $|x|$*.*

We also extend both WE + LEN and WE + REG by combining them as follows:

Definition 4 *Let* WE + LEN + REG *be the set of formulas adhering to the following syntax:*

- *A* WE + REG*-formula is a* WE + LEN + REG *formula.*
- *A* WE + LEN*-formula is a* WE + LEN + REG *formula.*
- *For* WE + LEN + REG*-formulas* ψ_1, ψ_2*, the Boolean combinations* $\psi_1 \wedge \psi_2$*,* $\psi_1 \vee \psi_2$ *and* $\neg\psi_1$ *are all* WE + LEN + REG*-formulas.*

The semantics for WE + LEN + REG formulas follows directly from the semantics from WE + LEN and WE + REG. Generally, since our approach is primarily oriented around word equations, we shall use the terms length constraints and regular constraints to refer to subformulas involving the length-variables $|x|$ and regular language memberships $x \in L$ respectively.

We extend the notion of expressibility of languages to the extensions of WE in the natural way.

Definition 5 (Expressibility of languages) *Let* $\mathfrak{T}$ *be any of the logical theories* WE*,* WE + REG*,* WE + LEN *and* WE + LEN + REG *defined above. For a* $\mathfrak{T}$*-formula* ψ *and a variable* x *occurring in* ψ*, the language expressed by* x *in* ψ *is*

$$L = \{h(x) \mid h \text{ is a satisfying assignment to } \psi\}.$$

A language $L \subseteq \Sigma^*$ *is expressible in* $\mathfrak{T}$ *if there exists a* $\mathfrak{T}$*-formula* ψ *containing a variable* x *such that* L *is expressed by* x *in* ψ*. We shall use the notation* $\mathcal{L}(\mathfrak{T})$ *to denote the class of languages expressible in* $\mathfrak{T}$*.*

It is straightforward that

$$\mathcal{L}(\mathrm{WE}) \subseteq \mathcal{L}(\mathrm{WE}+\mathrm{LEN}), \mathcal{L}(\mathrm{WE}+\mathrm{REG}) \subseteq \mathcal{L}(\mathrm{WE}+\mathrm{LEN}+\mathrm{REG}).$$

In addition, we can infer the following directly from known results:

Theorem 2 [32] *$\mathcal{L}(\mathrm{WE})$ is incomparable to the classes of regular and context free languages. It is a strict subclass of the decidable (recursive) languages.*

From this we can also conclude that $\mathcal{L}(\mathrm{WE}+\mathrm{REG})$ and $\mathcal{L}(\mathrm{WE}+\mathrm{LEN}+\mathrm{REG})$ are strict superclasses of the regular languages. The following observation is easily obtained from the fact that intersection of languages can be modelled using conjunction, and that intersection of context free languages can be used to describe accepting computation histories of Turing machines (the idea being to use word equations to extract the initial part of the computation history corresponding to the input word).

Remark 3 *Let $\mathrm{WE}+\mathrm{CF}$ be the set of formulas obtained by extending regular language membership in $\mathrm{WE}+\mathrm{REG}$ to deterministic context free language membership. Then $\mathcal{L}(\mathrm{WE}+\mathrm{CF})$ is exactly the class of recursively enumerable languages.*

As a result of Lemma 3, Remark 2, and the well-known effective closure properties of regular languages, we can rewrite any $\mathrm{WE}+\mathrm{LEN}+\mathrm{REG}$-formula in the following normal form:

Lemma 4 *Let ψ be a $\mathrm{WE}+\mathrm{LEN}+\mathrm{REG}$-formula containing variables $X = \{x_1, x_2, \ldots, x_k\}$. Then we can compute from ψ a formula ψ' with variables $x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_\ell$ such that $h : \{x_1, x_2, \ldots, x_k\} \to \Sigma^*$ is a satisfying assignment to ψ if and only if there exists a satisfying assignment $h' : \{x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_\ell\} \to \Sigma^*$ to ψ' satisfying $h(x_i) = h'(x_i)$ for $1 \leq i \leq k$, and where ψ' has the form:*

$$\bigvee_{1 \leq i \leq N} \left(E_i \wedge \psi_{\mathrm{len},i} \wedge \bigwedge_{z \in X} z \in L(A_{i,z}) \right)$$

where E_i is a single (positive) word equation, $\psi_{\mathrm{len},i}$ is a quantifier-free Presburger arithmetic formula whose variables correspond to the lengths $|z|$ of variables $z \in \{x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_\ell\}$ and $A_{i,z}$ is a single DFA, such that for any $y, z \in X$ with $y \neq z$, $A_{i,y}$ and $A_{i,z}$ do not share any states.

Proof Firstly, rewrite ψ so that it is in disjunctive normal form (DNF). By commutativity of $\wedge$ we may then assume w.l.o.g. that we have a disjunction of N clauses of the form:

$$\bigwedge_{1 \leq i \leq k_1} \hat{E}_i \wedge \bigwedge_{1 \leq i \leq k_2} \hat{L}_i \wedge \bigwedge_{1 \leq i \leq k_3} \hat{R}_i$$

where:

- each $\hat{E}_i$ is either E or $\neg E$ for some word equation E,

- each $\hat{L}_i$ is either $c_0 + \sum_{1 \leq i \leq k} c_i |x_i| = 0$ or $\neg \left(c_0 + \sum_{1 \leq i \leq k} c_i |x_i| = 0 \right)$, and
- each $\hat{R}_i$ is either of the form $x \in L$ or $\neg (x \in L)$ for some variable x and regular language L.

By Lemma 3, we can replace $\bigwedge_{1 \leq i \leq k_1} \hat{E}_i$ with a single word equation E. Moreover, we can assume w.l.o.g. that each regular language is given as a DFA. For each $\hat{R}_i$ of the form $\neg (x \in L)$, we can compute the complement automaton accepting $\overline{L}$ and replace it with the subformula $x \in \overline{L}$. Finally, combine multiple subformulas involving regular language membership for the same variable into a single one. Specifically, we replace $x \in L_1 \wedge x \in L_2 \wedge \ldots \wedge x \in L_t$ by $x \in L$ where $L = \bigcap_{1 \leq i \leq t} L_i$. The corresponding DFA can be obtained via the product construction, and states renamed appropriately so that no two automata share any states. By taking $\psi_{\text{len},i} = \bigwedge_{1 \leq i \leq k_2} \hat{L}_i$ we obtain a formula ψ' of the desired form. □

2.2 Synchronising Factorisations and Inexpressibility for Word Equations

Both in Sections 3 and 4, we shall make use of a framework introduced in [32] for showing the inexpressibility of languages by word equations. Since we are adapting and extending this framework in a non-trivial way, it is convenient to recall, and in some cases rephrase, the technical details. Nevertheless, we encourage the interested reader to also consult [32] for full technical details and complete proofs where they are omitted here.

The general approach for showing inexpressibility in [32] is similar in nature to canonical pumping arguments e.g. for showing a language is not regular. We start with the assumption that the language L is expressible, and so, in the case of word equations or WE, that there is a word equation E and variable x expressing L. We then pick some word $w \in L$, implying the existence of a solution h to E satisfying $h(x) = w$. Next, we use some insights into the properties of solutions to word equations to modify w by changing some part(s) of it, yielding a new solution h' where $h'(x) = w'$. By taking care to ensure that this process leads to some $w' \notin L$, we arrive at a contradiction to the assumption that L is expressible.

A fundamental observation which facilitates this reasoning is that while some parts of a solution to a word equation are fixed (directly or indirectly) by the constants in the equation, under certain circumstances, parts of a solution might be entirely independent of any such constants. We shall make a distinction between "anchored" and "unanchored" parts of the solution. A very simple example can be derived from the solution $h(x) = a, h(y) = h(z) = b$ to the word equation $xz \doteq ay$. If we change the first (and only) letter of $h(x)$, we will get a mismatch to the constant a on the right hand side of the equation, so the first letter of $h(x)$ is fixed by, or anchored to that constant a. On the other hand the bs in $h(y)$ and $h(z)$ only need to match with each other, and not to any constant in the equation itself. We could replace the bs in $h(y)$ and $h(z)$ by any other word $w \in \Sigma^*$, and still have a solution to the equation. For example, replacing

b with abc yields a solution $h(x) = a, h(y) = h(z) = abc$. It is this replacement of an unanchored factor which will allow us to modify w to obtain a word w' outside the language in question. A more complex example involving anchored and unanchored parts of a solution to a word equation is given in Fig. 2.

Establishing which letters in a solution are fixed or anchored by which (if any) constants, and how they are fixed relative to each other, is known as the method of "filling the positions". However, considering individual letters only is not sufficiently powerful enough for this technique to be effective. Rather, a key insight in [32] is how this reasoning can be adapted to work for larger factors of the solution rather than just letters alone. This is non-trivial to do because factors can overlap and dependencies can propagate in a more complicated fashion. To keep track of the way in which factors can overlap, and to limit the resulting complexities, the authors introduce the notion of a synchronising factorisation, which we now recall.

A *factorisation scheme* is a mapping $\mathfrak{F} : \Sigma^* \to \bigcup_{k\geq 0} (\Sigma^+)^k$ of words w onto tuples of words $(w_1, w_2, \ldots, w_k)$ such that $w = w_1 w_2 \cdots w_k$. The tuple $(w_1, w_2, \ldots, w_k)$ is called the $\mathfrak{F}$-factorisation of w, and the w_i are called the $\mathfrak{F}$-factors of w. For example, one factorisation scheme, $\mathfrak{F}_{\text{runs}}$, might divide a word into "runs" or maximally long factors comprised of a single letter. In that case, the $\mathfrak{F}_{\text{runs}}$-factorisation of $aababaaabbaa$ would be $(aa, b, a, b, aaa, bb, aa)$. Naturally, we can extend the notion of an $\mathfrak{F}$-factorisation of a word to a substitution h. In this case, we get factorisations $(w_1, w_2, \ldots, w_k)$ for each variable-image $h(x)$. By the $\mathfrak{F}$-factors of h, we mean the union of the sets of factors occurring in the $\mathfrak{F}$-factorisation of each variable-image.

Synchronising factorisation schemes adhere to the following definition. Although technical when written formally, the general concept is natural and straightforward: if y occurs as a factor of x, then their corresponding $\mathfrak{F}$-factorisations must coincide, or "synchronise", for all but some constant number of factors on the left and right of y (see Fig. 1).

Definition 6 (Synchronising Factorisation Scheme [32]) *A factorisation scheme $\mathfrak{F}$ is synchronising if the following all hold:*

- *Every word possesses an $\mathfrak{F}$-factorisation (it is complete),*
- *Every word has at most one $\mathfrak{F}$-factorisation (it is uniquely deciphering),*
- *There exist parameters $l, r \in \mathbb{N}_0$, such that the following "synchronising condition" is satisfied: for all pairs of words x, y with $\mathfrak{F}$-factorisations $(x_1, ..., x_s)$ and*

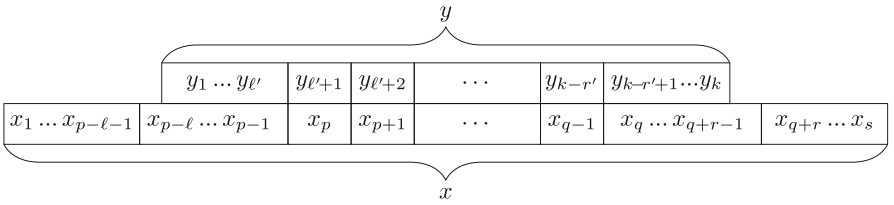

Fig. 1 Depiction of a synchronising factorisation adapted from a similar figure in [32]. If the factorisation scheme is synchronising, then the factorisations of a word x and a factor y of x should align exactly after a fixed number of factors to the left and right

$(y_1, ..., y_k)$, if $k > l + r$ and y occurs as a factor of x starting at the i^{th} letter, then there exist $l' \leq l$ and $r' \leq r$ such that for $U = y_1 \cdots y_{l'}$ and $V = y_{k-r'+1} \cdots y_k$:

- *The positions $i + |U|$ and $i + |y| - |V|$ in x are starting positions of $\mathfrak{F}$-factors, say x_p and x_q, respectively.*
- *The sequences of $\mathfrak{F}$-factors $x_p, ..., x_{q-1}$ and $y_{l'+1}, ..., y_{k-r'}$ are identical.*
- *The occurrence of U at position i in x covers at most $l - 1$ $\mathfrak{F}$-factors of x (i.e. $|x_1 x_2 \cdots x_{p-\ell-1}| < i \leq |x_1 x_2 \ldots x_p|$).*
- *The occurrence of V at position $i + |y| - |V|$ in x covers at most $r - 1$ $\mathfrak{F}$-factors of x (i.e. $|x_1 x_2 \cdots x_{q-1}| \leq i + |y| \leq |x|$).*

It is easily seen that the factorisation scheme $\mathfrak{F}_{\text{runs}}$ is synchronising with $l = r = 1$. Further examples can be found in [32] and in Section 3 and 4.

For our purposes, we do not actually rely on the precise partition of $\mathfrak{F}$-factors into "anchored" and "unanchored". Since the full formal definition requires several further lengthy technical definitions, for simplicity, we omit it and simply observe formally that $\mathfrak{F}$-factors may either be anchored or unanchored (Definition 7 below), and providing only an informal intuition instead.

Remark 4 *In [32], the terminology "anchored" and "unanchored" is not used, but the factors which can be freely swapped are called "proper" instead. We avoid using the term "proper" in order to limit confusion with its more common usage for factors not equal to the whole word.*

What we do need is a sufficient condition for unanchored $\mathfrak{F}$-factors to exist, along with the observation that we can swap them for other words to produce new solutions to an equation. These are given in Lemma 5, which is a rephrasing of Theorems 16 and 17 in [32].

Definition 7 ((Un)anchored Factors) *Let $\mathfrak{F}$ be a synchronising factorisation scheme. Let E be a word equation and let h be a solution to E. Let $u_1, u_2, \ldots, u_k$ be the $\mathfrak{F}$-factors of h. Then each u_i can either be anchored or unanchored.*

Informally, we can think of unanchored $\mathfrak{F}$-factors of a solution h to an equation $U \doteq V$ in terms of how they overlap. In particular, consider the two identical words $h(U)$ and $h(V)$, factorised by applying a factorisation scheme $\mathfrak{F}$ to the image of each symbol in U and then of each symbol in V. This gives us two factorisations of the solution-word $h(U)$, each uniquely determined by h and the corresponding side of the equation, U or V (see Fig. 2). We can then, for each factor in these factorisations, associate an interval $[i, j] \subseteq [1, |h(U)|]$ describing its occurrence in the solution word $h(U)$. We consider two occurrences of factors to overlap if their intervals have a non-empty intersection. They partially overlap if they overlap and their intervals are not identical. They match if the intervals are identical. Informally, we say that a $\mathfrak{F}$-factor u of h is unanchored if it satisfies the following (see also Fig. 2).

(i) It does not overlap any of the constants in the equation, and
(ii) it does not partially overlap any other occurrence of a $\mathfrak{F}$-factor of h, and
(iii) it does not overlap or match directly with any anchored $\mathfrak{F}$-factors.

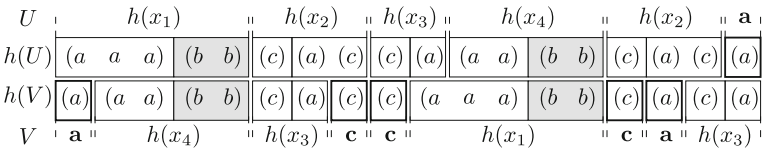

Fig. 2 Depiction of anchored and unanchored factors in a solution to a word equation. Specifically, the word equation E is given by $U \doteq V$ where U is $x_1x_2x_3x_4x_2a$ and V is $ax_4x_3ccx_1cax_3$. The variables are given by $X = \{x_1, x_2, x_3, x_4\}$ and the constants are given by $\Sigma = \{a, b, c\}$. The solution h is given by $h(x_1) = aaabb$, $h(x_2) = cac$, $h(x_3) = ca$, $h(x_4) = aabb$. The division of the variable-images and constants into their respective $\mathfrak{F}_{\text{runs}}$-factorisations is shown using brackets. Overall, there are 5 distinct $\mathfrak{F}_{run}$-factors occurring, namely a, aa, aaa, bb, c. Only one of these, bb satisfies the criteria for being unanchored. It is highlighted in grey. The others are all anchored because they have occurrences which do not align exactly, or which overlap with some constant. Note that the way in which the unanchored factor bb occurs means that its occurrences can be swapped for any other factor v, to obtain another solution to E: the "new" occurrences of v will line up exactly, and all other parts of the solution remain unaffected

All occurrences of unanchored factors occur entirely inside the variable-images, and are only dependent of the other unanchored parts of the solution, meaning that, like the letter b in the earlier example, they can be exchanged for any other word without disrupting the overall structure of the solution.

The notation $n_{\mathfrak{F}}(w)$ is used to indicate the number of distinct $\mathfrak{F}$-factors of the word w. Since our aim is to replace certain $\mathfrak{F}$-factors, we also introduce a notation for that. In particular, for a substitution $h : (X \cup \Sigma)^* \to \Sigma^*$ and a synchronising factorisation scheme, $\mathfrak{F}$, denote by $h^{\mathfrak{F}}_{u\to v}$ the substitution obtained by replacing all occurrences of u with v in the $\mathfrak{F}$-factorisations of the variable-images $h(x)$.

Remark 5 *We derive $h^{\mathfrak{F}}_{u\to v}$ from h as follows. For each variable x, firstly divide $h(x)$ into its $\mathfrak{F}$-factorisation $(w_1, w_2, \ldots, u, \ldots, w_i, \ldots, u, \ldots, w_k)$, replace each occurrence of u with v to get a tuple (not necessarily a valid $\mathfrak{F}$-factorisation) $(w_1, w_2, \ldots, v, \ldots, w_i, \ldots, v, \ldots, w_k)$, and then concatenate the resulting factors to obtain $h^{\mathfrak{F}}_{u\to v}(x) = w_1w_2 \cdots v \cdots w_i \cdots v \cdots w_k$.*

Note that since the $\mathfrak{F}$-factorisation exists and is unique for any word by definition, the above procedure is always well-defined and deterministic. Consequently, $h^{\mathfrak{F}}_{u\to v}$ is always well-defined. In particular, there is no danger of trying to replace overlapping occurrences of a factor.

The following lemma is a crucial tool for our reasoning, and is adapted from [32] to allow us to use it as a "black box" in what follows. Most important to us, is that we can swap out unanchored factors in a solution h, and that we can guarantee their existence simply by ensuring that there are sufficiently many distinct $\mathfrak{F}$-factors in the $\mathfrak{F}$-factorisation of h.

Lemma 5 (Adapted from [32]) *Let $\mathfrak{F}$ be a synchronising factorisation scheme. Let E be a word equation with variables from X. Then the following hold:*

1. *There exists a constant c depending only on E and $\mathfrak{F}$ such that for any solution h to E, at most c distinct $\mathfrak{F}$-factors of h are anchored.*
2. *Let h be a solution to E. Let u be an unanchored factor in h. For any word v, $h^{\mathfrak{F}}_{u\to v}$ is well-defined and is also a solution to E.*

The general approach for showing that a language L is not expressible by word equations (that is, L cannot be expressed in WE) can now be described by the following steps:

- **Step 1.** Assume that L is expressible, and thus that there exists an equation E and variable x such that $\{w \mid \exists$ a solution h to E with $h(x) = w\} = L$.
- **Step 2.** Pick an appropriate *synchronising factorisation scheme* $\mathfrak{F}$.
- **Step 3.** For all sufficiently (with respect to the constant from Lemma 5) $k \in \mathbb{N}$, choose a word $w \in L$ such that w has more than k distinct $\mathfrak{F}$-factors. Note that $w \in L$ implies there exists at least one solution h to E such that $h(x) = w$, and by Lemma 5, at least one $\mathfrak{F}$-factor is unanchored.
- **Step 4.** Choose v such that swapping occurrences of u in w yields a word $w' = h^{\mathfrak{F}}_{u\to v}(x)$ not belonging to L. By Lemma 5, $h^{\mathfrak{F}}_{u\to v}$ is a solution to E, so $w' \in L$.
- **Step 5.** By the assumption that L is expressed by x in E, we have that $h'(x) \in L$, a contradiction. Thus L must be inexpressible.

For clarity, we include the following example, adapted from a similar one in [32], which demonstrates this approach by showing that the (regular) language $\{a, b\}^*c$ is not expressible by word equations.

Example 1 (Adapted from [32]) *Let $L = \{a, b\}^*c$. Suppose (step 1) that L is expressed by the variable x in some equation (or* WE*-formula) E. Then there exists a solution h to E with $h(x) = w$ if and only if $w \in L$. Now (step 2), we consider the factorisation scheme $\mathfrak{F}_{\text{runs}}$, which factorises a word into maximally long sequences of a single letter.*

Next (step 3), clearly for any k, the word $w = aba^2b^2 \ldots a^kb^kc$ is in L and has $2k + 1$ distinct $\mathfrak{F}_{\text{runs}}$-factors. Thus, by Lemma 5, for k large enough, there are at least two $\mathfrak{F}_{\text{runs}}$-factors u which are unanchored and can be swapped for any word v while still retaining membership in L. At least one of these factors will have the form $u = a^i$ or $u = b^i$ for some i, $1 \leq i \leq k$.

(Step 4) let $v = c$ and let $w' = h^{\mathfrak{F}}_{u\to v}(x)$ be the result of swapping all occurrences of u with v. Then (Step 5) by our assumptions we should have that $w' \in L$. However $w' \in \{a, b\}^*c\{a, b\}^*c$, so $w' \notin \{a, b\}^*c = L$. This is our contradiction and L cannot be expressed by word equations.

2.3 Visibly Pushdown Languages

In Section 3 we shall also consider a theory of word equations and visibly pushdown language membership constraints, which lies between word equations with regular constraints and word equations with context free constraints. Since visibly pushdown languages are not as widely known as regular and context free languages, we provide some further introduction here (see [3–5] for a thorough introduction). A pushdown alphabet $\widetilde{\Sigma}$ is a triple $(\Sigma_c, \Sigma_i, \Sigma_r)$ of pairwise-disjoint alphabets known as the call, internal and return alphabets respectively. A visibly pushdown automaton (VPA) is a pushdown automaton for which the stack operations (i.e. whether a push, pop or neither is performed) are determined by the input symbol currently being read. In particular, any transition for which the input symbol a belongs to the call alphabet Σ_c, must push a symbol to the stack while any transition for which $a \in \Sigma_r$ must pop a symbol from

the stack unless the stack is empty and any transition for which $a \in \Sigma_i$ must leave the stack unchanged. Acceptance of a word is determined by the state the automaton is in after reading the whole word. The stack does not need to be empty for a word to be accepted. A $\widetilde{\Sigma}$-visibly pushdown language is the set of words accepted by a visibly pushdown automaton with pushdown alphabet $\widetilde{\Sigma}$. A language L is a visibly pushdown language (and is part of the class VPLang) if there exists a pushdown alphabet $\widetilde{\Sigma}$ such that L is a $\widetilde{\Sigma}$-visibly pushdown language. The class VPLang is a strict superset of the class of regular languages and a strict subset of the class of deterministic context free languages, which retains many of the nice decidability and closure properties of regular languages. In particular, it has already been shown in [4] that VPLang is closed under union, intersection and complement and moreover that the emptiness, universality, inclusion and equivalence problems are all decidable for VPLang.

Similarly to the logics defined in Section 2.1, we define an extension of WE allowing VPL membership constraints. For this logic, we assume an underlying alphabet Σ which is a pushdown alphabet, and we treat the partition into Σ_c, Σ_i, Σ_r as constant --- that is we do not allow different subformulas to refer to visibly pushdown languages in which the roles of the letters are different.

Definition 8 *Let* WE + VPL *be the set of formulas adhering to the following syntax:*

- *A word equation is a* WE + VPL*-formula,*
- $x \in L$ *is a* WE + VPL*-formula where* L *is a visibly pushdown language specified by a visibly pushdown automaton,*
- *For* WE + VPL*-formulas* ψ_1, ψ_2*, the Boolean combinations* $\psi_1 \wedge \psi_2$*,* $\psi_1 \vee \psi_2$ *and* $\neg\psi_1$ *are all* WE + VPL*-formulas.*

3 Classes of Languages Expressible by Extended Word Equations

In this section, we consider the relative expressive power of the logics WE, WE + LEN, WE + REG and WE + LEN + REG in terms of the classes of languages they can express. It is easily seen that all languages expressible in the most general logic WE + LEN + REG are all recursively enumerable: given a formula ψ and variable x, a semi-decision procedure for membership of a word w in the language expressed by x in ψ can be obtained by simply enumerating through all possible assignments h satisfying $h(x) = w$ and checking whether each one is satisfying. If at some point a satisfying assignment is found, then the word belongs to the language. Since there are only a finite number of variables occurring in ψ, enumerating the substitutions is possible. The results in [32] are sufficient to directly establish a strict inclusion between $\mathcal{L}$(WE) and $\mathcal{L}$(WE + REG) while results from [11] can be used to establish a strict inclusion between $\mathcal{L}$(WE) and $\mathcal{L}$(WE + LEN). Thus, our starting point, prior to our results in this section, is the following hierarchy:

$$\mathcal{L}(\text{WE}) \subset \mathcal{L}(\text{WE} + \text{LEN}),\ \mathcal{L}(\text{WE} + \text{REG}) \subseteq \mathcal{L}(\text{WE} + \text{LEN} + \text{REG}) \subseteq \text{RE}.$$

In what follows, our primary aim is to establish strictness for the other inclusions and incomparability between $\mathcal{L}$(WE + REG) and $\mathcal{L}$(WE + LEN).

Arguably the most interesting inclusion, in terms of establishing separation is $\mathcal{L}(\mathrm{WE}+\mathrm{LEN}+\mathrm{REG}) \subseteq \mathrm{RE}$, since it is an open problem as to whether satisfiability (and therefore emptiness for the corresponding class of languages) is decidable for formulas in $\mathrm{WE}+\mathrm{LEN}+\mathrm{REG}$. The existence of examples of recursively enumerable languages which are not expressible is a necessary condition for having a decidable satisfiability problem, and if we wish to settle this open problem we must also settle the existence of such examples. Further, despite being a core class of constraints addressed by string solvers, the precise expressive power of this logic is not well understood, so finding examples and classes of languages which are/are not expressible is well motivated.

Our first main result does precisely this by establishing, with some involved argumentation, a sufficient criterion for languages to not be expressible in $\mathrm{WE}+\mathrm{LEN}+\mathrm{REG}$ and using it to identify a concrete example which is clearly recursively enumerable.

Our approach builds on the general approach from [32] described in Section 2.2. The difference we face in this context is that when swapping unanchored factors to derive the contradiction, we need to be able to guarantee that the substitution obtained after the swap still satisfies the length and regular language membership constraints.

For length constraints, the required adaptation is straightforward: if we simply swap a factor u for a word v satisfying $|u| = |v|$, then the lengths of the variable-images are guaranteed to stay the same, so the length constraints will remain satisfied. For regular constraints, the simplest approach involves guaranteeing that the factor u to be swapped is sufficiently long that it can be pumped in accordance with the pumping lemma for regular languages, and using this pumping to derive the word v. As we shall see, there are some details which need to be managed which lead to a slightly more intricate pumping argument, but it can nevertheless still be done.

Unfortunately, however these two adaptations, for length and regular constraints respectively, are mutually exclusive. We cannot pump factors of a word in a non-trivial way while maintaining its length. The obvious solution is to try to pump some factors positively and other factors negatively in order to achieve an overall balance in the length, however this does not appear always to be possible in this context.

We avoid this problem altogether by taking a somewhat different approach, based on the growth rate of a language. This leads to a more involved proof with a more complex contradiction, but nevertheless provides some insight into the types of languages which cannot be expressed in $\mathrm{WE}+\mathrm{LEN}+\mathrm{REG}$.

Theorem 3 *There exist recursively enumerable languages which are not expressible in* $\mathrm{WE}+\mathrm{LEN}+\mathrm{REG}$. *Thus*

$$\mathcal{L}(\mathrm{WE}+\mathrm{LEN}+\mathrm{REG}) \subset \mathrm{RE}.$$

Proof Given a language L, we define the counting function $\#_L : \mathbb{N}_0 \to \mathbb{N}_0$ such that $\#_L(n)$ is the number of words in L of length n. For example, if $L_1 = \{a, b\}^*$ and $L_2 = \{a\}^*$, then $\#_{L_1}(n) = 2^n$ while $\#_{L_2}(n) = 1$. We call $\#_L$ the growth function of L. Sometimes this function is also called "combinatorial complexity" in the literature (see e.g. [45]). In order to describe the asymptotic behaviour of potentially non-monotonic

functions $\#_L$, we shall use adaptations of the typical Ω, Θ notations, as in [45], as follows.

Given a function $f : \mathbb{N}_0 \to \mathbb{N}_0$, we say that the growth of L is at least f if there exists a positive constant C, such that there are infinitely many n such that $\#_L(n) \geq Cf(n)$, and we denote this growth as $\overline{\Omega}(f)$. Similarly, we say the growth of L is at most f if $\#_L(n)$ is $O(f)$. Finally, if the growth of L is both $\overline{\Omega}(f)$ and $O(f)$, then we say it is $\overline{\Theta}(f)$. For example, the growth of the language $\{w \in \{a, b\}^* || w| \text{ is even}\}$ is $\overline{\Theta}(2^n)$.

Let $\Sigma = \{a, b, c, d, @, \$\}$. Let $\pi_{ab} : \Sigma^* \to \{a, b\}^*$ be the projection onto $\{a, b\}^*$, (i.e. π_{ab} is the morphism such that $\pi_{ab}(a) = a$, $\pi_{ab}(b) = b$ and $\pi_{ab}(c) = \pi_{ab}(d) = \pi_{ab}(@) = \pi_{ab}(\$) = \varepsilon$). In what follows, we shall show that the following language L is not expressible by word equations with length constraints and regular constraints:

$$\begin{aligned} L = \{w = w_1 @^{2^{2^k}-1} \$ w_2 @^{2^{2^k}-1} \$ \ldots \$ w_k @^{2^{2^k}-1} \$ \mid \\ \forall i, j, 1 \leq i, j \leq k : w_i \in \{ac^{i-1}d^{k-i}, bc^{i-1}d^{k-i}\}^* \\ \wedge \pi_{ab}(w_i) = \pi_{ab}(w_j) \qquad \wedge |w_i| = k^2\}. \end{aligned}$$

Intuitively, words w in L are determined exactly by a numerical parameter k and a "base" word $w_{\text{base}} \in \{a, b\}^k$. The "full" word w then consists of k copies of w_{base} in succession, separated by the letter \$. Ideally, each copy of w_{base} in w would be over a distinct alphabet. However the number of copies grows with k, so to achieve this with only a finite alphabet, we encode the letters a and b in each copy by factors $ac^{i-1}d^{k-i}$ and $bc^{i-1}d^{k-i}$. In other words, for each i, $1 \leq i \leq k$, the set $\{ac^{i-1}d^{k-i}, bc^{i-1}d^{k-i}\}$ acts as a new copy of the alphabet $\{a, b\}$. Moreover, we want the number of possible base words w_{base} to be logarithmic with respect to the length of the word overall, so we pad each one with a large number of @ symbols.

We shall fix a factorisation scheme $\mathfrak{F}$ so that it divides any word in Σ^* into words of the form $(\Sigma \backslash \{\$\})^*\$$, and if necessary adds the final prefix from $(\Sigma \backslash \{\$\})^+$. For any word not containing \$, the factorisation is just the word itself. So, for example, $\mathfrak{F}(abababa) = (abababa)$ and $\mathfrak{F}(ab\$bbaba\$\$aba) = (ab\$, bbaba\$, \$, aba)$. Clearly, $\mathfrak{F}$ is synchronising.

Suppose that ψ is a WE + LEN + REG-formula with variables X where for some $x \in X$, L is expressed by x in ψ. In what follows, we shall use the growing number of copies of w_{base} in words $w \in L$ to force a sufficiently large number of unanchored factors with respect to $\mathfrak{F}$ so that we can apply a swapping argument in the style of [32] and as discussed in the preliminaries.

We shall then use the logarithmic growth of the number of choices for w_{base} to enforce, in a rough sense, a minimum of logarithmic growth in the regular constraints in ψ. Then, by properties of regular languages, we can extrapolate this logarithmic growth to linear growth which we can ultimately use to derive nearly-linear growth for the whole language L, a contradiction to the fact that L clearly has logarithmic growth:

Claim 1 *For each $n \in \mathbb{N}$, either:*

- *There exists k such that $n = k^3 + k2^{2^k}$ and there are exactly 2^k words of length n in L, or*

- *No such k exists and there are no words of length n in L.*

Thus the language L has growth $\overline{\Theta}(\log(n))$.

Proof Directly from the definitions. □

By Lemma 4 we can assume w.l.o.g. that ψ is given in the form:

$$\bigvee_{1 \leq i \leq N} \left(E_i \wedge \psi_{\text{len},i} \wedge \bigwedge_{z \in X} z \in L(A_{i,z}) \right)$$

where E_i is a single (positive) word equation, $\psi_{\text{len},i}$ is a Presburger arithmetic formula whose variables correspond to the lengths $|z|$ of variables $z \in X$ and $A_{i,z}$ is a single DFA, such that for any $y, z \in X$ with $y \neq z$, $A_{i,y}$ and $A_{i,z}$ do not share any states. Clearly $L = \bigcup_{1 \leq i \leq N} \tilde{L}_i$ where $\tilde{L}_i$ is the language expressed by x in $E_i \wedge \psi_{\text{len},i} \wedge \bigwedge_{z \in X} z \in L(A_{i,z})$.

Claim 2 *There is at least one i such that* $\tilde{L}_i$ *has growth* $\overline{\Theta}(\log(n))$.

Proof Since L has growth $\overline{\Theta}(\log(n))$, it has growth $O(\log(n))$. It follows directly that all the languages $\tilde{L}_i$ have growth $O(\log(n))$. Moreover, L has growth $\overline{\Omega}(\log(n))$ so there is a positive constant C and infinitely many $n \in \mathbb{N}_0$ such that L has at least $C \log(n)$ words of length n. Let $C' = \frac{C}{N}$. For each such n, at least one of the languages $\tilde{L}_i$ must have at least $C' \log(n)$ words. Since there are only finitely many choices of $\tilde{L}_i$, there must be at least one specific choice of $\tilde{L}_i$ which can be made for infinitely many of the n. Thus, there are at least $C' \log(n)$ words in $\tilde{L}_i$ for infinitely many n and $\tilde{L}_i$ has growth $\overline{\Omega}(\log(n))$. The claim then follows immediately by definition. □

For the next steps of the proof, we shall concentrate on a single subformula $\tilde{\psi} = E_i \wedge \psi_{\text{len},i} \wedge \bigwedge_{z \in X} z \in L(A_{i,z})$ whose corresponding language $\tilde{L}_i$ has growth $\overline{\Theta}(\log(n))$. For convenience, we drop the index i. Thus, let $\tilde{L} = \tilde{L}_i$, $E = E_i$, $A_{i,z} = A_z$ for each $z \in X$, and $\psi_{\text{len}} = \psi_{\text{len},i}$. We refer to ψ_{len} as the length constraints, and to $\bigwedge_{z \in X} z \in L(A_z)$ as the regular constraints. Denote by Q the set of all states occurring in one of the DFAs A_z. For each $z \in X$, denote by ι_z and $\hat{\delta}_z$ the initial state and extended transition function of A_z respectively. Let $\hat{\delta}_Q = \bigcup_{z \in X} \hat{\delta}_z$, and note that since the automata A_z do not share states, δ_Q is a well-defined total function. For each $w \in \tilde{L}$, denote by h_w some satisfying assignment to $\tilde{\psi}$ (note this is then also a satisfying assignment for ψ as a whole).

Since our aim is to swap some factor(s) of a word w from L while continuing to have a valid solution, we need a way of keeping track, for a given factor u of w, which choices of v we may substitute for u without "breaking" the regular constraints and length constraints. For length constraints, we restrict v so that $|v| = |u|$. For the regular constraints, if the corresponding automaton begins reading an occurrence of u in state p and finishes reading u in state q, then swapping u for a word v which also takes the

automaton from p to q will not disrupt that constraint. However, we need to account for the fact that there can be multiple occurrences of u which we swap for v, and in the images of any of the variables. Thus we need to consider the combinations of all states of all the automata in which an occurrence of u starts/ends in a given solution. For this reason, we define the following sets $\Phi(z, w, u)$ below (Definition 9). The set of words v which respect these combinations of states turns out to be a regular language, which we refer to as L_R (Definition 10). An example is given in Fig. 3.

Definition 9 *For each $z \in X$, $w, u \in \Sigma^*$, we define the following set:*

$$\Phi(z, w, u) = \{(p, q) \mid \exists w_1, w_2.\ w = w_1 u w_2 \wedge \hat{\delta}_z(\iota_z, w_1) = p \wedge \hat{\delta}_z(p, u) = q\}.$$

In other words, $(p, q) \in \Phi(z, w, u)$ if and only if there is an occurrence of u in w such that when reading $h_w(z)$, A_z is in state p just before reading the first letter of that occurrence of u and is in state q just after reading the last letter of that occurrence of u.

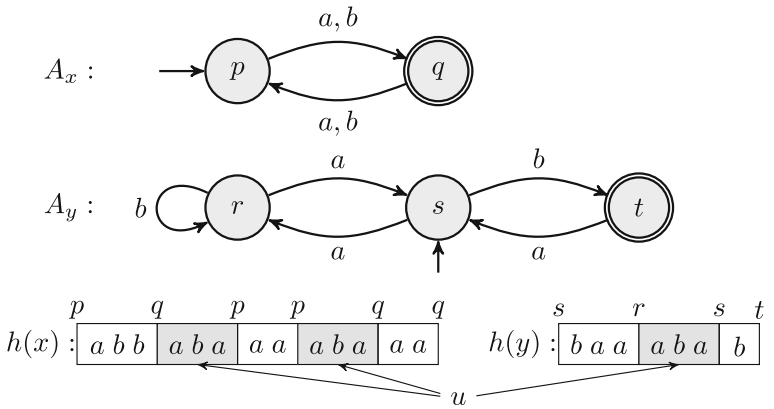

Fig. 3 An example illustrating Defintions 9, 10 and Claim 4. Suppose we have a solution h to a word equation E with variables x and y, which also satisfies some length constraints and some regular constraints given by DFAs A_x and A_y. Explicit values for A_x, A_y, $h(x)$, and $h(y)$ are given in the figure. Trap states and associated transitions are omitted for clarity. Let $u = aba$. Occurrences of u are highlighted, and some of the states visited in the runs of $h(x)$ on A_x and $h(y)$ on A_y respectively are also indicated. Occurrences of u start/end in states (p, q) and (q, p) in the run of $h(x)$ on A_x and start/end in states (r, s) in the run of $h(y)$ on A_y. Let $R = \bigcup_{z \in X} \Phi(z, h(z), u) = \{(p, q), (q, p), (r, s)\}$. Note that $\{v \mid \hat{\delta}_x(p, v) = q\} = \{v \mid \hat{\delta}_x(q, v) = p\} = \{v \in \Sigma^* \mid |v| \text{ is odd}\}$ and $\{v \mid \hat{\delta}_y(r, v) = s\} = b^*a(ba|ab^*a)^*$. Thus L_R consists of all odd-length words in $b^*a(ba|ab^*a)^*$. Suppose that u is unanchored. Then if we replace each occurrence of u in (the $\mathfrak{F}$-factorisations of) $h(x)$ and $h(y)$ with a word v, we obtain a new assignment h' which is still a solution to the word equation E. Moreover, if $v \in L_R$, then the sequences of states depicted in the figure remain the same (although some states "internal" to the u factor which are not depicted might change). Thus the resulting assignment h' will also satisfy the regular constraints. Finally, if we choose v such that $v \in L_R$ and $|v| = |u|$, then h' will also still satisfy the length constraints. For example, taking $v = aaa$ meets all these conditions and gives a new satisfying assignment $h'(x) = abbaaaaaabaaa$ and $h'(y) = baaaaab$

Definition 10 *Given $R \subseteq Q \times Q$, let L_R be the language*

$$\bigcap_{(p,q)\in R} \{v \mid \hat{\delta}_Q(p, v) = q\}.$$

Claim 3 *L_R is always a regular language, and it is accepted by an automaton with at most $|Q|^{|Q|^2}$ states.*

Proof For each pair $(p, q) \in R$ where p, q belong to an automaton A_z, we can construct an automaton $A_{(p,q)}$ accepting the language $\{v \mid \hat{\delta}_Q(p, v) = q\}$ of words v for which there is a path from p to q labelled v in A_z by simply making p the only initial state and q the only final state. Then, we can simply use the product automaton construction to construct an automaton accepting the intersection $\bigcap_{(p,q)\in R} L(A_{(p,q)})$.

Since there are at most $|Q|^2$ pairs in R, the product construction is applied to at most $|Q|^2$ automata. Moreover, there are at most $|Q|$ states in each of the individual automata. Thus, there are at most $|Q|^{|Q|^2}$ states overall. □

The following claim is the necessary generalisation of 2. from Lemma 5, and provides the conditions under which swapping an unanchored factor will preserve a satisfying assignment to our formula $\tilde{\psi}$. An example demonstrating how Claim 4 can be used is given in Fig. 3.

Claim 4 *Let h be a satisfying assignment to $\tilde{\psi}$. Let u be an unanchored factor in the $\mathfrak{F}$-factorisation of h. Let $R \supseteq \bigcup_{z\in X} \Phi(z, h(z), u)$. Let $v \in L_R$ such that $|u| = |v|$. Then $h^{\mathfrak{F}}_{u\to v}$ is well-defined and also a satisfying assignment for $\tilde{\psi}$.*

Proof By Lemma 5, $h^{\mathfrak{F}}_{u\to v}$ is well-defined and also a solution to E. In particular, recall that by definition of an $\mathfrak{F}$-factorisation, and since $h^{\mathfrak{F}}_{u\to v}$ is obtained by swapping $\mathfrak{F}$-factors in the variable-images $h(x), x \in X$, there is no danger of trying to simultaneously swap overlapping occurrences of u.

With this in mind, for each $z \in X$, we can write $h(z)$ as $w_1 u w_2 u \dots u w_\ell$ such that $h_{u\to v}(z) = w_1 v w_2 v \dots v w_\ell$. By definition, for each i, $1 \le i < \ell$, there is a pair $(p, q) \in R$ such that $\hat{\delta}_z(\iota_z, w_1 u w_2 \dots w_i) = p$ and $\hat{\delta}_z(p, u) = q$. Moreover, since $v \in L_R$, $\hat{\delta}_z(p, v) = q$ for each pair (p, q) of states from R. It follows that $\hat{\delta}_z(\iota_z, h(z)) = \hat{\delta}_z(\iota_z, h^{\mathfrak{F}}_{u\to v}(x))$. Thus, since $h(z) \in L(A_z)$, we may infer that $h^{\mathfrak{F}}_{u\to v}(z) \in L(A_z)$. This holds for all z, so $h^{\mathfrak{F}}_{u\to v}$ satisfies all the regular constraints in $\tilde{\psi}$.

Finally, since $|u| = |v|$ then the lengths of variable-images will not change. That is, $|h^{\mathfrak{F}}_{u\to v}(z)| = |h(z)|$ for all $z \in X$, so it follows from the fact that h is a satisfying assignment for ψ_{len} that $h^{\mathfrak{F}}_{u\to v}$ is also a satisfying assignment for ψ_{len} and the length constraints are also satisfied.

Thus $h^{\mathfrak{F}}_{u\to v}$ satisfies the word equation E, the regular constraints, and the length constraints and is therefore a satisfying assignment to $\tilde{\psi}$.

Next, we need to assert that for sufficiently long words in $\tilde{L}$, there will exist unanchored factors which provide candidates for being replaced. This is established by the following claim.

Claim 5 *For each $k \in \mathbb{N}$ large enough, for every word*

$$w = w_1 @^{2^{2^k}-1} \$ w_2 @^{2^{2^k}-1} \$ \dots w_k @^{2^{2^k}-1} \$ \in \tilde{L}$$

there is at least one i, $1 \leq i \leq k$ such that $w_i @^{2^{2^k}-1} \$$ is an unanchored $\mathfrak{F}$-factor of a satisfying assignment h_w witnessing $w \in \tilde{L}$.

Proof This follows directly from the fact that there is a constant c dependent only on E and $\mathfrak{F}$ such that there are at most c $\mathfrak{F}$-factors of w which are anchored (see Lemma 5). By definition of $\mathfrak{F}$ and $\tilde{L}$, for any two different values i, the factors $w_i @^{2^{2^k}-1} \$$ will be distinct $\mathfrak{F}$-factors of $h_w(x)$. Thus whenever $k > c$, there must be one choice of i such that $w_i @^{2^{2^k}-1} \$$ is unanchored. □

Claim 4 gives us a sufficient criterion for performing the previously described swapping for some factor u of a word w in the language. However, there is not yet any reason to assume that the set of possible choices v satisfying the conditions of the claim contains any word other than u itself. In what follows, we shall show firstly that by the construction of the language $\tilde{L}$, there are at least logarithmically many choices for v, and secondly by the properties of regular languages, that this leads to (nearly) linearly many. By choosing factors u that are sufficiently long, this gives the required contradictory lower bound on the growth of $\tilde{L}$.

The following claim establishes the logarithmic number of choices for v.

Claim 6 *There exists a constant $C_0 > 0$ and $R \subseteq Q \times Q$ such that for infinitely many $k \in \mathbb{N}$:*

- *there exist at least $C_0 \frac{2^k}{2|Q|^2}$ distinct (unanchored) factors u of length $k^2 + 2^{2^k}$ of words $w \in \tilde{L}$ such that $u \in L_R$, and*
- *there exists at least one word $w \in \tilde{L}$ of length $n = k^3 + k2^{2^k}$ and an unanchored factor u of length $k^2 + 2^{2^k}$ of w such that $R \supseteq \bigcup_{z \in X} \Phi(z, h(z), u)$, where h is a satisfying assignment to $\tilde{\psi}$ witnessing $w \in \tilde{L}$.*

Proof Since each $w \in \tilde{L}$ has length $k^3 + k2^{2^k}$ for some k, it follows from the fact that the growth of $\tilde{L}$ is $\overline{\Theta}(\log(n))$ that there is a constant C_0 and infinitely many k such that there are at least $C_0 2^k$ words of length $k^3 + k2^{2^k}$ in $\tilde{L}$. Moreover, each word $w \in \tilde{L}$ is fixed exactly by the initial prefix $w_1 @^{2^{2^k}-1} \$$, so this means there must be infinitely many k for which there are $C_0 2^k$ possible choices of the initial prefix $w_1 @^{2^{2^k}-1} \$$ of some word $w \in \tilde{L}$.

By Claim 5, for any k large enough, for any single word in $\tilde{L}$, at least one of its factors $w_i @^{2^{2^k}-1} \$$ is unanchored. Thus, for k large enough, across all words of length $k^3 + k2^{2^k}$ in $\tilde{L}$, we must have a combined total of at least $C_0 2^k$ distinct unanchored factors $u = w_i @^{2^{2^k}-1} \$$.

Each of these u must belong to at least one L_R for some $R \supseteq \bigcup_{z \in X} \Phi(z, h(z), u)$ where h is a satisfying assignment witnessing the corresponding word $w \in \tilde{L}$. Since

there are only $2^{|Q|^2}$ possibilities for R, there must be, for each k large enough, at least one R such that $\frac{C_0 2^k}{2^{|Q|^2}}$ of the factors u belong to the same language L_R. Moreover, since there are only finitely many R and infinitely many k, we must have that there is an infinite subset of the k's for which the choice of R satisfying the above is the same. This is enough to prove both statements of the claim. □

The following claim allows us to translate a logarithmic number of options v for swapping an unanchored $\mathfrak{F}$-factor u into a linear one simply by looking at the possible growth rates of regular languages. Note that while it is well-known that the growth rate of a regular language cannot be logarithmic generally, the context here means we need a more particular version of this statement concentrating just on subsets of possible lengths.

Claim 7 *Let L' be a regular language. Suppose there exists a constant C_1 and an infinite subset S of natural numbers such that for each $n \in S$, $|L' \cap \Sigma^n| \geq C_1 \log(n)$. Then there exists a constant C_2 and an infinite subset $S' \subseteq S$ such that for each $n \in S'$, $| L' \cap \Sigma^n | \geq C_2 n$.*

Proof Suppose L', S, C_1, C_2 satisfy the conditions imposed by the claim. Let A' be a DFA accepting L'. Let Q' be the set of states of A'. For convenience, we shall consider runs of words w' on A' as words over a combined alphabet $\Sigma \times Q'$ and ignoring the final state (which is uniquely determined by the last letter from $\Sigma \times Q'$ in the run since A' is a DFA). So e.g. $(a, q_0), (b, q_1), (a, q_2)$ would denote that the word aba, when read by A' starts in initial state q_0, then goes to state q_1 then goes to state q_2, before finishing in some unknown state. Since A' is a DFA, each word in L' has a unique run written in this form. Moreover, any run ending with a pair (a, q) defining a transition in A' to an accepting state necessarily defines a unique word in L'.

We shall call a word atomic if the corresponding run does not contain any state more than once. Clearly, any word/run in L' can be reduced to an atomic word by removing factors corresponding to cycles in A'. We shall call this process "depumping". Moreover, any word in L' can be obtained by taking an atomic word and "repumping" in factors corresponding to cycles. Note that there are only finitely many atomic words.

It might be that a state q does not belong to the run of some atomic word w'_{atom}, but is part of a run of the full word w' which was depumped. In this case, there is a cycle starting/ending at some state q' in the run of w'_{atom} visiting q. It is easy to see that if this is possible, then it is possible for at least one new state q with a cycle which does not itself have any sub-cycles (i.e. visits each state at most once). Thus, we can turn w'_{atom} into a word w'' of length at most $|Q|^2$ which contains every state reachable in some cycle from a state in the run of w'_{atom}. Let $\mathfrak{Q}$ be the function which maps an atomic word w'_{atom} to the length-lexicographically minimal choice of w''.

For each subset of states $Q'' \subseteq Q'$, denote by $C(Q'')$ the greatest common divisor of lengths of all cycles starting/ending at states in Q''.

Next we list several easily proven observations.

- For any $Q'' \subseteq Q'$, there is a finite subset of cycles starting/ending at states $q \in Q''$ whose greatest common divisor is equal to $C(Q'')$.

- For any word $w' \in L'$, if w'_{atom} is a corresponding atomic word obtained by depumping, then there exists $d \in \mathbb{N}$ such that $|w'| = |w'_{\text{atom}}| + dC(Q'')$ where Q'' is the set of states occurring in the run of w'_{atom} or occurring in a cycle starting/ending at one of those states. In particular, this is true for $w'' = \mathfrak{Q}(w'_{\text{atom}})$.
- Suppose $m_1, \ldots, m_r$ are (not necessarily distinct) numbers with $r > 1$ and $gcd(m_1, \ldots, m_r) = D$. Then any large enough $M \in \mathbb{N}$ which is divisible by D can be written as $M = a_1m_1 + a_2m_2 + \ldots a_rm_r$ with $a_i \in \mathbb{N}_0$. Moreover, if a_i, a_j are such that $i \neq j$ and a_j is the largest coefficient, if all other co-efficients are fixed, there are still at least $\lfloor \frac{a_j}{m_i} \rfloor$ choices for a_i, a_j such that the same equality still holds (we can take $a'_i = a_i + km_j$ and $a'_j = a_j - km_i$ for any k satisfying $km_i \leq a_j$). For each choice of $m_1, m_2, \ldots, m_r$, there are $\Omega(M)$ many choices of a_i, a_j, since we must have $a_j \geq \frac{M}{m_{max}r}$ where $m_{max} = \max\{m_1, m_2, \ldots, m_r\}$.

Now for each $n \in S$ we can assign a corresponding w'_{atom} which is the result of depumping some word w' of length n in L'. Note that w'_{atom} fixes the set Q'' of states occurring in the run of w'_{atom} or occurring in a cycle starting/ending at one of those states, as well as and $C(Q'')$ and $w'' = \mathfrak{Q}(w'_{\text{atom}})$. It also fixes $n' = n - |w''|$. Note that n' is divisible by $C(Q'')$ (since w'' is obtained from a word w' of length n by firstly removing a multiple of $C(Q'')$ letters to get w'_{atom} and then re-inserting a multiple of $C(Q'')$ letters).

Since there are finitely many possible choices for w'_{atom}, we can fix one choice for which there exists an infinite subset $S' \subseteq S$ containing only $n \in S$ to which that choice w'_{atom} is assigned such that for all $n \in S'$, there are at least $C_3 \log(n)$ words $w' \in L'$ of length n. Moreover, the $C_3 \log(n)$ words will all have runs only visiting states from Q''. Thus we may w.l.o.g. assume that A' only has the states Q'' and possibly some additional sink state.

We say that two cycles are similar if they (their runs written as words over $\Sigma \times Q$) have conjugate primitive roots. We try to fix a finite set $\mathfrak{C}$ of cycles starting at states occurring in the run of $\mathfrak{Q}(w'_{\text{atom}})$ such that the following hold:

- the greatest common divisor of the lengths of the cycles is equal to $C(Q'')$
- no two cycles in $\mathfrak{C}$ are similar
- there are at least two cycles in $\mathfrak{C}$

Suppose firstly that no such set $\mathfrak{C}$ exists. Then all cycles in A' starting and ending in states from Q'' are similar. Thus all cycles starting/ending at states in Q'' are repetitions of conjugates of a single cycle. This necessarily forces that the automaton A' is just a single cycle possibly with some further paths from the initial state which join the cycle and paths going out which reach a final state or trap state but such that none of the states on these paths have cycles.

It is straightforward that for an automaton with this structure, the number of words of length n is bounded by a constant and so cannot have growth $\overline{\Omega}(\log(n))$, a contradiction to the initial assumptions of the claim. Thus, $\mathfrak{C}$ does exist.

Let $m_1, m_2, \ldots, m_r$ be the lengths of the cycles in $\mathfrak{C}$. W.l.o.g. suppose there is an order on the states in Q'' and that the m_is are ordered w.r.t. the state the cycle starts/ends at. For all $n \in S'$ large enough, there exist $a_1, a_2, \ldots, a_r \in \mathbb{N}$ such that

$n' = n - |w''| = a_1 m_1 + a_2 m_2 + \ldots + a_r m_r$ and a pair a_i, a_j such that a_j has size $\Theta(n)$.

For each state in Q'', we pick some occurrence of a letter in w'' corresponding to that state in the run of w''. We re-pump w'' to obtain a word of length n by successively adding in a_ℓ copies of the cycle of length m_ℓ at the letter associated with the chosen occurrence of the ℓ^{th} state of Q'' in w'', starting at $\ell = 1$ and ending with $\ell = r$. Where two cycles are associated with the same state, we repump all occurrences of the first cycle followed by all occurrences of the second.

There is a constant C_2, independent of n, such that there are at least $C_2 n$ possible choices for a_i and a_j if we fix all the other a_ℓs. It remains to show that each resulting word of length n is different. Suppose to the contrary that two are the same. Then we get a word w''' whose run on A' can be simultaneously written as $w'_1 c_1^{a_i} w'_2 c_2^{a_j} w'_3$ where c_1 and c_2 are the cycles associated with a_i and a_j, for two different values of a_i and a_j (or it might be that a_i and a_j occur the other way around but the reasoning works in the same way). However, by cancelling identical prefixes and suffixes, this implies that $c_1^{s_1} w'_2 = w'_2 c_2^{s_2}$ for some $s_1, s_2 \in \mathbb{N}$. By Lemma 2 this is only possible if $c_1^{s_1}$ and $c_2^{s_2}$ are conjugate. Since repetitions of two words are conjugate if and only if their primitive roots are conjugate (see [38], Proposition 1.3.3), c_1 and c_2 are similar, This is a contradiction to the definition of $\mathfrak{C}$. Thus, we get at least $C_2 n$ distinct words of length n in L' for all $n \in S'$ large enough. By restricting S' to contain all n which are large enough, the conditions of the claim are satisfied. □

We are now finally ready to directly prove a contradiction about the growth of $\tilde{L}$ and thus that L is not expressed by ψ and therefore not expressible in WE + LEN + REG. In particular, we shall show that $\tilde{L}$, and thus L, has growth at least $\frac{n}{\log\log(n)}$, which is incompatible with Claim 1.

Let R be as defined in Claim 6. Then there is a constant C_1 such that for infinitely many values $n = k^3 + k2^{2^k}$:

- there are at least $C_1 2^k$ words in L_R of length $\frac{n}{k} = k^2 + 2^{2^k}$,
- there exists at least one word $w \in \tilde{L}$ of length n and an unanchored factor u of length $\frac{n}{k}$ of w such that $R \supseteq \bigcup_{z \in X} \Phi(z, h(z), u)$, where h is a satisfying assignment for $\tilde{\psi}$ witnessing $w \in \tilde{L}$.

Note that $C_1 2^k \in \Theta(\log(n))$ and $\frac{n}{k} \in \Theta\left(\frac{n}{\log\log(n)}\right)$. Since L_R is regular, and since $\log(n) > \log\left(\frac{n}{\log\log(n)}\right)$, by Claim 7, this implies that for some constant C_2, there are infinitely many n such that:

- there are at least $C_2 \frac{n}{k}$ words in L_R of length $\frac{n}{k}$,
- there exists at least one word $w \in \tilde{L}$ of length n and an unanchored factor u of length $\frac{n}{k}$ of w such that $R \supseteq \bigcup_{z \in X} \Phi(z, h(z), u)$, where h is a satisfying assignment for $\tilde{\psi}$ witnessing $w \in \tilde{L}$.

By Claim 4, for each $w \in \tilde{L}$ and corresponding satisfying assignment h, there is an unanchored factor u and at least $C_2 \frac{n}{k}$ words v such that $h^{\mathfrak{F}}_{u \to v}$ is a satisfying

assignment for $\tilde{\psi}$. By construction, each word $w'_v = h^{\mathfrak{F}}_{u\to v}(x)$ is unique and thus for infinitely many n, there are at least $C_2 \frac{n}{k}$ words in $\tilde{L}$ of length n. The growth rate of $\tilde{L}$ is therefore $\overline{\Omega}(\frac{n}{k}) = \overline{\Omega}\left(\frac{n}{\log\log n}\right)$. This implies that L also has growth rate at least $\overline{\Omega}\left(\frac{n}{\log\log n}\right)$. This contradicts Claim 1 so L cannot be expressible. □

We now turn our attention to weaker combinations of word equations and constraints. The case of word equations with length constraints is a straightforward adaptation of the existing approach from [32]. Since the language $L = \{vc \mid v \in \{a, b\}^*\}$ is regular, it is expressible in WE + REG. Thus the following lemma shows separation of $\mathcal{L}(\text{WE} + \text{LEN})$ and $\mathcal{L}(\text{WE} + \text{REG})$ and therefore a strict inclusion between $\mathcal{L}(\text{WE} + \text{REG})$ and $\mathcal{L}(\text{WE} + \text{LEN} + \text{REG})$.

Lemma 6 *Let a, b, c be distinct letters. Then the language $L = \{vc \mid v \in \{a, b\}^*\}$ is not expressible in* WE + LEN.

Proof Suppose to the contrary that L is expressible. Then there exists a formula ψ with variables X such that ψ consists of word equations and length constraints, and such that L is expressed by x in ψ. As in Lemma 4, due to constructions from [32] and using standard constructions regarding finite automata (for complement and intersection) and logical formulas (DNF) we can assume w.l.o.g. that ψ is given in the form:

$$\bigvee_{1\leq i\leq N} \left(E_i \wedge \psi_{\text{len},i}\right)$$

where E_i is a single (positive) word equation and $\psi_{\text{len},i}$ is a Presburger arithmetic formula whose variables correspond to the lengths $|z|$ of variables $z \in X$.

We proceed in a similar manner as for Example 1. For each k, let $w_k = aba^2ba^3b \dots a^kbc$. Let $\mathfrak{F}_{\text{runs}}$ be the synchronising factorisation scheme introduced in Section 2. Clearly $w_k \in L$ for all k. Moreover each w_k has exactly $k + 2$ distinct $\mathfrak{F}$-factors. For each k, there must exist i such that there is a satisfying assignment h for $\psi_i = E_i \wedge \psi_{\text{len},i}$ such that $h(x) = w_k$. By Lemma 5, if we take k large enough, then there is at least one $\mathfrak{F}$-factor a^j of w_k which is unanchored, and thus such that $h^{\mathfrak{F}}_{a^j\to c^j}$ is a satisfying assignment to E_i. Furthermore, since $|a^j| = |c^j|$, the lengths of the variable-images of $h^{\mathfrak{F}}_{a^j\to c^j}$ are the same as for h, so $h^{\mathfrak{F}}_{a^j\to c^j}$ satisfies $\psi_{\text{len},i}$. Thus $h^{\mathfrak{F}}_{a^j\to c^j}$ is a satisfying assignment for ψ_i and $w' = h^{\mathfrak{F}}_{a^j\to c^j}(x) \in L$. However, w' contains two distinct factors from c^+, so is not in L, a contradiction. It follows that L is not expressible in WE + LEN. □

Showing that a language is not expressible in WE + REG is slightly more involved than for WE + LEN, but we have already done most of the work in the proof of Theorem 3. We provide the following lemma providing a general necessary condition for a language to be expressible in WE + REG.

Lemma 7 *Let $\mathfrak{F}$ be a synchronising factorisation scheme. L be a language expressible in* WE + REG. *Then there exist constants c and d depending only on $\mathfrak{F}$ and L such that the following holds. For any word $w \in L$ with $\mathfrak{F}$-factorisation $(u_1, u_2, \dots, u_k)$,*

if there are at least c distinct factors u_i having length $u_i > d$, then there is at least one, u_j, and a word v with $|v| < |u_j|$ such that the word obtained by replacing each occurrence of u_j in $(u_1, u_2, \ldots, u_k)$ with v yields a word w' in L.

Proof Let ψ be a WE + REG formula and x a variable in ψ such that L is expressed by x in ψ. By Lemma 4, we can assume w.l.o.g. that ψ has the form:

$$\bigvee_{1 \leq i \leq N} \left(E_i \wedge \bigwedge_{z \in X} z \in L(A_{i,z}) \right)$$

where E_i is a single (positive) word equation and $A_{i,z}$ is a single DFA, such that for any $y, z \in X$ with $y \neq z$, $A_{i,y}$ and $A_{i,z}$ do not share any states. Clearly $L = \bigcup_{1 \leq i \leq N} \tilde{L}_i$ where $\tilde{L}_i$ is the language expressed by x in $E_i \wedge \bigwedge_{z \in X} z \in L(A_{i,z})$. Let Q be the set of all states occurring in any of the DFAs A_z.

Let h be a satisfying assignment to ψ witnessing that $w \in L$, so that $h(x) = w$. Let $d > |Q|^{|Q|}$. By Lemma 5, there is a constant c, depending on $\mathfrak{F}$ and L, such that if there are at least c distinct $\mathfrak{F}$-factors u_i with $|u_i| > d$, then at least one of them, u_j, will be unanchored. Set $u = u_j$ and let $\Phi(z, w, u)$ for $z \in X$, and L_R be defined as in the proof of Theorem 3, where $R = \bigcup_{x \in X} \Phi(x, h(x), u)$.

Now by Claim 3 in the proof of Theorem 3, the regular language L_R is described by a DFA A with at most d states. Moreover, as a corollary to Claim 4 in the same proof, for any $v \in L_R$, the assignment $h^{\mathfrak{F}}_{u_j \to v}$ is also satisfying for ψ (i.e. we can drop the condition $|u| = |v|$ because we do not have any length constraints), and therefore the word $w' = h^{\mathfrak{F}}_{u_j \to v}(x)$ obtained by replacing each occurrence of u_j in $(u_1, u_2, \ldots, u_k)$ with v is also in L. It remains to show that v exists such that $|v| \leq d$. However, by definition, $u \in L(A) = L_R$ so $L(A) \neq \emptyset$. Moreover, A has at most d states (Claim 3), so there must be at least one $v \in L(A)$ with $|v| \leq d < |u|$. This proves the lemma. □

We can apply Lemma 7 above to obtain the following separation between $\mathcal{L}$(WE + REG) and $\mathcal{L}$(WE + LEN + REG).

Lemma 8 *Let $L = \{ucv \mid u, v \in \Sigma^* \wedge |u| = |v|\}$. Then L is not expressible in* WE + REG.

Proof Recall that the factorisation scheme $\mathfrak{F}_{\text{runs}}$ introduced in Section 2 is synchronising. Suppose for contradiction that L is expressible in WE + REG. Let c, d be the constants from Lemma 7.

It is a straightforward observation that there exist pairwise distinct numbers $p_1, p_2, \ldots, p_c, q_1, q_2, \ldots, q_c > d$ such that $\sum p_i = \sum q_i$ and thus such that $w = a^{p_1} b^{p_1} a^{p_2} b^{p_2} \ldots a^{p_c} b^{p_c} c a^{q_1} b^{q_1} a^{q_2} b^{q_2} \ldots a^{q_c} b^{q_c}$ belongs to L.

The $\mathfrak{F}$-factorisation of w consists of $4c$ distinct factors having length greater than d (namely a^{p_i}, b^{p_i}, a^{q_i}, b^{q_i} for $1 \leq i \leq c$), so by Lemma 7 we may swap all occurrences of at least one block u of letters for a strictly shorter word v and obtain another word in L. However, since each block of letters occurs only once (by the fact that the p_is and q_is are all pairwise distinct), this would result in a word for which one side of

the central occurrence of c is shorter than the other, and thus not belonging to L, a contradiction. Thus L cannot be expressible, as required.

Summarising our results so far, we get the following (now strict) hierarchy:

Theorem 4 *For $\mathfrak{T} \in \{\text{WE} + \text{LEN}, \text{WE} + \text{REG}\}$, the following strict inclusions hold:*

$$\mathcal{L}(\text{WE}) \subset \mathcal{L}(\mathfrak{T}) \subset \mathcal{L}(\text{WE} + \text{LEN} + \text{REG}) \subset \text{RE}$$

Moreover, $\mathcal{L}(\text{WE} + \text{REG})$ and $\mathcal{L}(\text{WE} + \text{LEN})$ are incomparable.

Proof All languages expressible in any of the considered theories are recursively enumerable because a semi-decision procedure for membership can be obtained by trying all substitutions (e.g. in increasing length-lexicographic order). The other (non-strict) inclusions follow by definition. The strictness of the inclusions $\mathcal{L}(\text{WE}) \subset \mathcal{L}(\mathfrak{T})$ follow from [32] and [11]. Lemma 6 gives a regular language not expressible in WE + LEN, while Lemma 8 gives a language which is clearly expressible in WE + LEN but not in WE + REG. This establishes incomparability of $\mathcal{L}(\text{WE} + \text{REG})$ and $\mathcal{L}(WE + LEN)$, and moreover the strictness of the inclusions $\mathcal{L}(\mathfrak{T}) \subset \mathcal{L}(\text{WE} + \text{LEN} + \text{REG})$. Theorem 3 establishes the strictness of the inclusion $\mathcal{L}(\text{WE} + \text{LEN} + \text{REG}) \subset \text{RE}$. □

We now return to the fact, established in Theorem 3, that not all recursively enumerable languages are expressible in WE + LEN + REG. An obvious question remains: what kinds of constraints can we extend word equations with in order to be able to express all recursively enumerable languages? We have already noted in Remark 3 that (deterministic) context free language membership constraints are sufficient. As a result, we can consider the logic WE + CF in which atoms are word equations or deterministic context free language memberships to be a (strict) generalisation of WE + LEN + REG. However, this generalisation is both unsurprising and not particularly insightful, as the expressive power of intersections of deterministic context free languages (namely the fact that they can be used to encode computation histories of Turing machines) is well known.

In contrast, visibly pushdown languages are closed under intersection (as well as union and complement) and have decidable intersection-emptiness problem. In terms of both their closure and computability properties, they are much closer to regular languages than to (deterministic) context free languages. Nevertheless, we have the following observation that extending word equations with visibly pushdown language membership constraints generalises both regular language membership and length constraints.

Lemma 9 *Let ψ be a* WE + LEN + REG*-formula with variables $x_1, x_2, \ldots, x_k$ and underlying alphabet Σ. Then we construct a visibly pushdown alphabet $\widetilde{\Sigma}$ and a* WE + VPL*-formula ψ' with variables $x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_\ell$ such that for any assignment $h : \{x_1, x_2, \ldots, x_k\} \to \Sigma^*$, h satisfies ψ if and only if there exists a satisfying assignment $h' : \{x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_\ell\} \to \widetilde{\Sigma}^*$ for ψ' such that $h'(x_i) = h(x_i)$ for $1 \leq i \leq k$.*

Proof Let ψ be a WE + LEN + REG formula. Let $\widetilde{\Sigma} = (\Sigma_c, \Sigma_i, \Sigma_r)$ where $\Sigma_c = \Sigma$, $\Sigma_i = \emptyset$ and $\Sigma_r = \{\#\}$ with $\# \notin \Sigma$. We construct ψ' from ψ as follows. Firstly, add

to ψ atoms $x \in \Sigma^*$ for every variable x occurring in ψ. Since any regular language is also visibly-pushdown (for any partition of the alphabet), we can leave unchanged all atoms in ψ of the form $x \in L$ where L is a regular language. Likewise, since word equations can occur as atoms in both WE + LEN + REG and WE + VPL, we leave the word equations in ψ unchanged. What remains is to translate each length-atom having the form

$$c_0 + \sum_{1 \leq i \leq k} c_i |x_i| = 0$$

into a combination of word equations and visibly pushdown language membership atoms. Note that it is straightforward to construct a visibly pushdown automaton with alphabet $\widetilde{\Sigma}$ for the language $L_{len} = \{w\#^{|w|} \mid w \in \Sigma^*\}$. We convert the length-atom above as follows. Firstly rearrange the length-atom above by moving any terms with negative coefficients to the right hand side. As a result, we get an equality of the form

$$c_0 + \sum_{1 \leq i \leq k_1} c_i |x_i| = d_0 + \sum_{1 \leq i \leq k_2} d_i |z_i|$$

We then replace it with the following formula:

$$\begin{aligned}
&\hat{x} \doteq a^{c_0} x_1^{c_1} x_2^{c_2} \dots x_{k_1}^{c_{k_1}} \\
\wedge\ &\hat{z} \doteq a^{d_0} z_1^{d_1} z_2^{d_2} \dots z_{k_2}^{d_{k_2}} \\
\wedge\ &\hat{y_1} \in \{\#\}^+ \\
\wedge\ &\hat{y_2} \doteq \hat{x}\hat{y_1} \quad \wedge \quad \hat{y_3} \doteq \hat{z}\hat{y_1} \\
\wedge\ &\hat{y_2} \in L_{len} \quad \wedge \quad \hat{y_3} \in L_{len}
\end{aligned}$$

where $\hat{x}$, $\hat{y}$, $\hat{z_1}$, $\hat{z_2}$, $\hat{z_3}$ are all new variables and $a \in \Sigma$ is chosen arbitrarily. The first two lines ensure that any satisfying assignment h' adheres to $|h'(\hat{x})| = c_0 + \sum_{1 \leq i \leq k_1} c_i |x_i|$ and $|h'(\hat{z})| = d_0 + \sum_{1 \leq i \leq k_2} d_i |z_i|$. The final three lines then enforce that $|h'(\hat{x})| = |h'(\hat{z})|$, and thus that the original length-atom is satisfied. It follows therefore directly from our construction that ϕ' satisfies the conditions of the lemma. □

Unfortunately, despite their apparent similarity to regular languages, and the resulting hope that they might lead to an extension of WE + REG (and indeed WE + LEN + REG) while sharing the desirable property of having a decidable satisfiability problem, we are able to show that this is not the case: adding the VPL constraints is already as powerful as adding context free constraints, and satisfiability is therefore undecidable for WE + VPL.

Our proof follows similar ideas as classical proof that intersection emptiness is undecidable for context free languages, but with some additional adaptations, necessary as this problem is decidable for VPLs. In the classical proof, two languages L_1 and L_2 are constructed for a given Turing machine M such that words in L_1 are lists of any number of pairs of consecutive configurations of M stored in a palindromic

manner, and words in L_2 contain copies of configurations stored in the same palindromic manner, but offset by one lone configuration at either end. The combination of these two structures when taking the intersection leads to words which record the full recursive computations of M --- each pair adhering to the condition imposed by L_1 models a computation step, while each "offset pair" in L_2 copies the result over to the next pair for L_1.

For our purposes, simulating computation histories in this manner is ideal, as we can use the presence of word equations to extract the factor corresponding to the original input word, and thus express the language accepted by M. Doing this for all Turing machines then expresses the class of recursively enumerable languages. However, when moving from context free languages to VPLs, there are severe restrictions on the use of the stack memory which lead to problems in producing the "classical" versions of the languages L_1 and L_2. For example, roughly speaking, the pairs in L_1 and L_2 must be for configurations encoded as words of the same length, and the required offset in L_2 cannot be achieved in the same way and must be handled instead by the word equations.

As a result the proof becomes a little more technical. The main intuition nevertheless remains straightforward: the combination of the (very limited) unbounded memory and finite state control is still powerful enough to model small, local changes in copies of a word, so L_1-like languages can still be expressed as VPLs. Unlike context free languages, VPLs do not allow for the kind of information loss induced by copying only part of a word, which means that the "offset copy language" L_2 cannot be expressed as a VPL. Hence the classical proof of intersection emptiness being undecidable for context free languages fails for VPLs. Yet, combining VPLs with word equations reintroduces the possibility of this information loss and allows us, with some care, to again express languages like L_2, which we can then again combine with L_1-like languages to model recursive computations.

Theorem 5 *The class of languages* $\mathcal{L}(\mathrm{WE} + \mathrm{VPL})$ *expressible by word equations with* VPL *constraints is exactly the class* RE *of recursively enumerable languages.*

Proof Let $L \subseteq \Sigma^*$ be a recursively enumerable language. Then there exists a 1-Tape deterministic Turing machine M with input alphabet Σ and states Q accepting L such that the following all hold:

- M has a semi-infinite tape which is bounded to the left.
- Q contains a single halting (accepting) state q_f and a single initial state q_0.
- The tape alphabet Γ includes all symbols from Σ as well as a "blank" symbol B and a further symbol \$ which shall be used as a delimiter signalling the end of the tape on the left. Moreover, $|\Gamma| = O(|\Sigma|)$.
- In the initial configuration, M is in state q_0, the tape-head scans the leftmost tape cell containing the delimiter \$, and the input word is written immediately to the right of \$ and all remaining tape cells are blank.
- The delimiter \$ cannot be modified, and \$ cannot be written on any other cell of the tape, than the leftmost one (i.e., at any point in the computation, M writes \$ if and only if it has just read the cell that already contains \$).

- M accepts the input word or goes in an infinite loop. Moreover, M accepts only after making at least one step, and in the last (i.e., accepting) configuration the tape-head scans the leftmost blank cell of the tape, and the state of M is q_f.
- The transition function of M is $\delta : Q \times \Gamma \to Q \times (\Gamma \setminus \{B\}) \times \{R, L\}$ (where R and L are symbols denoting a left and, respectively, right movement of the tape-head of M) and δ respects the restrictions imposed above.

We shall write the configurations of M as words of the form $u(q, a)vB^t$, where $a \in \Gamma$, $uv \in \$(\Gamma \setminus \{B\})^*$ such that $uavB^\omega$ is the current content of the tape of the machine (where B^ω means a right-infinite string containing only blanks), q is the current state of the machine, the tape-head scans the cell containing a, and $t \geq 1$. Note that $uavB^t$ is a prefix of the content of the tape, read left to right, including one or more B symbols. We shall use two distinct letters q_1, q_2 to encode the states. In particular, we associate each $q \in Q$ with a unique number $i \in [1, |Q|]$ and encode q as $q_1^i q_2^{|Q|-i}$. The letters q_1, q_2 and $a \in \Gamma$ will be part of the call alphabet of our VPLs in the rest of the proof. For a state q represented as $q_1^i q_2^{|Q|-i}$, we shall encode the pair (q, a) as $q_1^i q_2^{|Q|-i} a$, and we shall use (q, a) as a shorthand for this encoding where convenient. Note that, importantly, there is more than one representation of the same configuration of the machine M.

Now, given two words C_1 and C_2 describing configurations of M, we say that there is a transition from C_1 to C_2, denoted $C_1 \vdash C_2$, if C_2 is a string describing the configuration in which M transitions from the configuration described by C_1 and, moreover, $|C_1| = |C_2|$.

If C describes a configuration of M reached after a finite number $k \geq 1$ of steps by M on the input w, then there exists a sequence of words $C_0, \ldots, C_k$ describing configurations of M such that C_0 describes the initial configuration of M for the input w and $C_0 \vdash C_1 \vdash \ldots \vdash C_k = C$ (meaning in particular that $|C_i| = |C_j|$ for all $i, j \in \{0, \ldots, k\}$, as well): we simply need to choose a representation C_0 of the initial configuration which already contains all the blanks which will be scanned by the tape-head during the first k computation steps of M on w. A direct consequence of this is that an accepting computation of M on a word w can always be described by a sequence $C_0 \vdash C_1 \vdash \ldots \vdash C_k$ of words of equal length encoding the successive configurations.

We now define a visibly pushdown alphabet $\widetilde{\Delta} = (\Delta_c, \Delta_i, \Delta_r)$ of pairwise-disjoint alphabets, which stand for the call, internal and return alphabets, respectively, for the VPAs which we will construct from now on. Let $\Delta_c = \Gamma \cup \{\#, @, q_1, q_2\} \cup \{a \mid a \in \Gamma\}$ and $\Delta_r = \{a' \mid a \in \Delta_c\}$; that is, Δ_r consists in copies of the letters of the alphabet Δ_c. Finally, $\Delta_i = \{\blacksquare\}$. Let $f : \Delta_c^* \to \Delta_r^*$ be the antimorphism defined by $f(a) = a'$ for all $a \in \Delta_c$.

Next, let $L_1 = \{@C_1\#C_2\# \cdots \#C_k\#\blacksquare\#'C_k'\#' \cdots \#'C_1'@'\}$ where:

- For $i \leq k-1$, C_i is a configuration of M and $C_i' = f(D_i)$, where $C_i \vdash D_i$. In other words, C_i' is the image under f of the string describing the configuration which follows the configuration described by C_i in a computation of M.
- C_k describes a final configuration of M, and we have that $C_k' = f(C_k)$.

We can show that L_1 is accepted by a nondeterministic VPA E. This VPA functions according to the following algorithm. We shall represent the contents of the stack of E as a word with the rightmost symbol corresponding to the topmost symbol in the stack.

1. In the first move, E attempts to read @ and writes $@'$ on the stack.
2. E then attempts to read a word of the form $C_1\#C_2\#\ldots C_{k-1}\#$, $k \geq 1$ where each C_i is a valid representation of a configuration, and while doing so computes the configurations D_i and writes $D'_1\#'D'_2\#'\ldots\#'D'_{k-1}\#'$ on the stack (further details on how this is achieved are given below).
3. At some point, after reading a #, E guesses that the next configuration is a final one and attempts to read a valid representation of a final configuration C_k followed by another #. While doing so, it also writes $C'_k\#'$ to the stack.
4. Then E attempts to read ■, while not altering the stack contents. It then enters a new phase where it checks whether $f(D_1\#\cdots D_k\#) = \#'C'_k\#'\cdots\#'C'_1$ by iteratively popping a symbol from the top of the stack if and only if it matches the current symbol read on the input tape. E accepts the input if, when the input tape was completely read, the stack is empty (this can be checked using the fact that the last symbol popped must be $@'$).

At any point if E does not successfully read symbols it expects, then it enters a rejecting state. Since the set of representations of configurations and the set of representations of final configurations are both easily described as regular expressions, this is easily handled by the finite state control.

Clearly E only pushes to the stack while reading symbols from Σ_c, only pops symbols from the stack while reading symbols from Σ_r, and leaves the stack unchanged while reading symbols from Σ_i, so it is a visibly pushdown automaton. What remains is to describe how E writes D'_i to the stack when reading C_i for $1 \leq i < k$. This is possible primarily due to the fact that C_i and D_i have the same length, and differ only by a "short" factor which can be recorded/predicted in the finite state control. Specifically, it can be achieved as follows:

- After reading a #, if E does not guess that the next configuration is final, it non-deterministically guesses a transition that M will makes while in this configuration, and keeps track of this choice in the state.
- If the respective transition is $\delta(q, a) = (q_1, b, L)$, then $C_i = u(q, a)vB^t$ where $u \neq \varepsilon$. In this case, E begins reading from left to right the symbols $d \in \Gamma \setminus \{B\}$ of C_i and pushes the corresponding primed version directly onto the stack, until it non-deterministically decides that it has reached the last symbol c of u. It then reads c and pushes $(q_1, c)'$ onto the stack. E then tries to read (q, a), pushes b' onto the stack, and continues reading the symbols d of C_i and pushing the primed versions onto the stack, until it reaches the next #.
- If instead the respective transition is $\delta(q, a) = (q_1, b, R)$, then E attempts to read the symbols d of C_i and push their primed versions onto the stack, until it reads the symbol (q, a). It then pushes b' onto the stack. It then attempts to read a symbol d from Γ, and push $(q_1, d)'$ onto the stack. Finally E continues reading the symbols of C_i and pushing the primed versions onto the stack, until the next #.

As before, at any point where E reads a symbol which it does not expect according to the above process, it simply enters a rejecting state and rejects the input word. Clearly, by construction, E accepts exactly L_1. Further, the language $L_2 = \{z\blacksquare f(z) \mid z \in \Delta_c\}$ is a typical example of a VPL with alphabet $\widetilde{\Delta}$: we simply push the symbols before $\blacksquare$ to the stack and then pop them after reading $\blacksquare$ whilst checking that for each one the corresponding copy is read in the input. Furthermore, it is immediate that all regular languages can be accepted by VPAs.

So, we define the following formula φ with variables x, y, z, u, v, s_1, s_2:

$$\begin{aligned}
&x \in \Sigma^* \\
&\wedge s_1 \doteq y\blacksquare\#'u \quad \wedge s_1 \in L_2 \\
&\wedge v \in \{B\}^+ \\
&\wedge z \in (\Delta_r \setminus \{\#'\})^* \\
&\wedge s_2 \doteq @(q_0, \$)xv\#y\blacksquare\#'z\#'u@' \quad \wedge s_2 \in L_1.
\end{aligned}$$

It remains to show that the language expressed by x is L. From $s_2 \doteq @(q_0, \$)xv\#y\blacksquare\#'z\#'u@' \wedge s_2 \in L_1$ we get that

$$@(q_0, \$)xv\#y = @C_1\#C_2\#...\#C_k\#$$

and

$$\#'z\#'u@' = \#'C'_k\#'....\#'C'_1@'$$

for some configurations $C_1, \ldots, C_k$ where C_k is final, $C'_k = f(C_k)$ and $C'_i = f(D_i)$ with $C_i \vdash D_i$ for $1 \leq i < k$. Moreover, from $s_1 \in L_2$, $x \in \Sigma^*$ and $v \in \{B\}^+$, we can infer that $C_1 = (q_0, \$)xB^t$ for some $t \geq 1$, so C_1 must be an initial configuration of M where x corresponds precisely to the input word. We can also infer that $y = C_2\#\cdots C_k\#$. where $C_2, \ldots, C_k$ are valid representations of configurations. From $z \in (\Delta_r \setminus \{\#'\})^*$ we can also infer that $z = C'_k$ and $u = C'_{k-1}\#'\cdots\#'C'_1$.

Now, from $y\blacksquare\#'u \in L_2$, we get that $C'_i (= f(D_i)) = f(C_{i+1})$, for $1 \leq i < k$. Since f is clearly injective, this implies $D_i = C_{i+1}$ and thus that $C_i \vdash C_{i+1}$ for $1 \leq i < k$. Therefore, $C_1 \vdash \ldots \vdash C_k$ is an accepting computation of M, so $w \in L$. Altogether, we see that for any satisfying assignment to φ, the value of x is in L. Moreover, it is straightforward to see that the converse also holds: if $w \in L$, then there exists a sequence $C_1 \vdash C_2 \vdash \ldots \vdash C_k$ of words representing configurations of M such that C_0 describes the initial configuration of M for the input w and C_k describes a final configuration. With this in mind, a satisfying assignment can easily be derived from $x = w$, $v = B^{|C_1|-|w|-1}$, $y = C_2\#\cdots C_k\#$, $z = f(C_k)$, and $u = f(C_k)\#'f(C_{k-1})\cdots\#'f(C_2)$. Our claim now follows, and we have shown that any recursively enumerable language can be expressed in WE + VPL. □

Corollary 6 *Satisfiability for* WE + VPL *is undecidable. Moreover, given a single word equation E, and for each variable x occurring in E a single visibly pushdown language L_x, deciding whether E has a solution h satisfying $h(x) \in L_x$ for all variables x is also undecidable.*

Remark 6 *It is worth pointing out that the formula φ in the proof of Theorem 5 only uses word equations in a very restricted way: to introduce new variables expressing concatenations of previous variables and constants. Thus, expressibility of* RE *languages and all the consequences regarding negative decidability properties can be inferred for a much weaker set of formulas which only allow conjunctions of atoms of the form $T \in L$ where T is a concatenation of variables and constants and L is a VPL. Such a set of formulas can be seen as a generalisation of intersection.*

We conclude this section by considering the decision problem of whether a language expressed in one logic is expressible in another. In cases where the first logic expresses a subclass of languages expressed by the second, this problem is clearly trivial. However there are clear practical advantages to being able to solve the problem in the other direction: when asking whether a language (or more generally, a property on words) expressed by a formula from a more general logic can be rewritten in simpler form in a more restrictive logic without altering the language or property itself. Unfortunately, we are able to show that in several cases this is undecidable. We define the decision problem formally as follows.

Definition 11 (Simplification Problem) *Given logical theories $\mathfrak{T}_1, \mathfrak{T}_2$, such that $\mathcal{L}(\mathfrak{T}_1) \supset \mathcal{L}(\mathfrak{T}_2)$, and a language L expressible in $\mathfrak{T}_1$ (given by a formula and variable), is L expressible in $\mathfrak{T}_2$?*

In the case that $\mathfrak{T}_1 = \mathrm{WE} + \mathrm{VPL}$ (or any more general logic), Theorem 3 allows us to directly apply Rice's theorem.

Corollary 7 *Let $\mathfrak{T}_2 \in \{\mathrm{WE} + \mathrm{LEN} + \mathrm{REG}, \mathrm{WE} + \mathrm{LEN}, \mathrm{WE} + \mathrm{REG}, \mathrm{WE}\}$ (or any other logic for which the expressible languages are a strict subset of* RE*). Then the Simplification Problem for* $\mathrm{WE} + \mathrm{VPL}$ *and $\mathfrak{T}_2$ is undecidable.*

Using Greibach's theorem, below, we are also able to show undecidability in the case of $\mathrm{WE} + \mathrm{LEN} + \mathrm{REG}$ and $\mathrm{WE} + \mathrm{REG}$. Note that due to Theorem 3, we cannot apply Rice's theorem in this case.

Theorem 8 (Greibach's Theorem [29]) *Let $\mathcal{C}$ be a class of formal languages over an alphabet $\Sigma \cup \{\#\}$ such that each language in $\mathcal{C}$ has some associated finite description. Suppose $\mathcal{P} \subset \mathcal{C}$ such that all the following hold:*

1. *$\mathcal{C}$ and $\mathcal{P}$ both contain all regular languages over $\Sigma \cup \{\#\}$,*
2. *$\mathcal{P}$ is closed under quotient by a single letter,*
3. *Given (descriptions of) $L_1, L_2 \in \mathcal{C}$ descriptions of $L_1 \cup L_2$, $L_1 R$ and $R L_1$ can be computed for any regular language $R \in \mathcal{C}$,*
4. *It is undecidable whether, given $L \in \mathcal{C}$, $L = \Sigma^*$.*

Then the problem of determining, for a language $L \in \mathcal{C}$, whether $L \in \mathcal{P}$ is undecidable.

Theorem 9 *There is a constant c such that for $|\Sigma| > c$, the Simplification Problem for* $\mathrm{WE} + \mathrm{LEN} + \mathrm{REG}$ *and* $\mathrm{WE} + \mathrm{REG}$ *is undecidable.*

Proof Note that in order to apply Greibach's theorem, we need a variant of the universality problem to be undecidable which refers to a sub-alphabet, rather than the whole alphabet. Thus we define the subset-universality problem as follows:

Definition 12 *Let* $|\Sigma| \geq 2$. *Let* $S \subset \Sigma$. *We define the S-universality problem for a logical theory* $\mathfrak{T}$ *as follows: given a formula* $\psi \in \mathfrak{T}$, *and a variable* x *occurring in* ψ, *is the language expressed by* x *in* ψ *exactly* S^**?*

We have the following claim.

Claim 8 *There is a constant* c *such that for* $S \subset \Sigma$ *with* $|S| \geq c$, *the S-universality problem is undecidable for* WE + REG.

Proof In [25], the authors show that the "standard" universality problem is undecidable for WE for sufficiently large alphabets Σ by constructing, for any 2-counter automaton M, a WE formula ψ_M with variable x such that the language expressed by x contains words over Σ which are not valid computation histories for M (under some appropriate encoding). In [25], the construction actually uses an unbounded alphabet in general, including symbols for each state in a given two-counter automaton. However, the encoding is easily altered to rely only on a finite alphabet, e.g. by numbering the states and encoding each state in unary instead. We expect that a two letter alphabet is in general sufficient, but we do not include a full proof of that in the present work, thus we instead use an abstract constant c for the number of letters required for this encoding.

Now, if $S \subset \Sigma$ has enough letters for the encoding of computation histories, defining $\psi'_M = \psi_M \wedge x \in S^*$, we are able to derive an equivalent proof of undecidability for the S-universality problem. Clearly ψ'_M is a WE + REG-formula, so the claim holds. □

It remains to show that the classes $\mathcal{C} = \mathcal{L}(\text{WE} + \text{LEN} + \text{REG})$ and $\mathcal{P} = \mathcal{L}(\text{WE} + \text{REG})$ satisfy the conditions for Greibach's theorem to apply. Note that $\mathcal{P} \subset \mathcal{C}$ follows from Theorem 4. It is trivial that both $\mathcal{C}$ and $\mathcal{P}$ contain all regular languages. For a regular language R, it is expressed by x in the formula $x \in R$ which belongs to both logics. Hence Condition 1 holds. Let $L \in \mathcal{P}$ be expressed by x in a WE + REG formula ψ. Let $a \in \Sigma$ and let $\psi' = \psi \wedge ay \doteq x$ where y is a new variable not already in ψ. Then clearly y expresses the quotient of L by a in ψ', so Condition 2 holds. Let $L_1, L_2 \in \mathcal{C}$ be languages expressed by variables x_1, x_2 in WE + LEN + REG-formulas ψ_1, ψ_2 respectively, and R be a regular language. W.l.o.g. we may assume that ψ_1, ψ_2 do not share variables. Let x, y be variables not already occurring in ψ_1 or ψ_2. Then $L_1 \cup L_2$ is expressed by the variable x in the formula $\psi_1 \wedge \psi_2 \wedge (x \doteq x_1 \vee x \doteq x_2)$. $L_1 R$ is expressed by x in the formula $\psi_1 \wedge x \doteq x_1 y \wedge y \in R$ and $R L_1$ is expressed by x in the formula $\psi_1 \wedge x \doteq y x_1 \wedge y \in R$. Since all the above formulas belong to WE + LEN + REG, Condition 3 holds. Finally, Condition 4 holds directly due to Claim 8, since if $|\Sigma| = S \cup \{\#\}$ with $|\Sigma| > c$, then $S \subset \Sigma$ with $|S| \geq c$. Thus, the theorem holds as a consequence of Greibach's theorem. □

We conclude this section by noting some interesting open cases for the Simplification Problem.

Open Problem 1 *Is the Simplification Problem decidable for any of the following pairs of logics?*

$\mathfrak{T}_1$	$\mathfrak{T}_2$
WE + LEN	WE
WE + REG	WE
WE + REG	WE + LEN
WE + LEN + REG	WE
WE + LEN + REG	WE + LEN

4 Regular Languages (In)Expressible by Word Equations

In this section we turn our attention to WE, and consider the relationship between $\mathcal{L}$(WE) and the class of regular languages. Our first result is rather surprising, and can be seen as evidence of the complexity of WE.

Theorem 10 *Given a* WE*-formula* ψ *and variable* x*, it is undecidable whether the language expressed by* x *in* ψ *is regular.*

Proof We shall prove the statement by giving a reduction from the problem of determining whether or not the set of words belonging to 0^+ accepted by a 2-Counter Machine (2CM) is finite. Since we shall use word equations to model computations of 2CMs, our proof has a similar flavour to the one in [25], but since our aims and setting are different, the details and our construction are also necessarily different.

A 2CM M is a deterministic finite state machine with 3 semi-infinite storage tapes, each with a leftmost cell but no rightmost cell. One is the input tape, on which the input is initially placed. There is a read-only head which can move along the input tape in both directions but cannot move beyond the input and cannot overwrite the input. The other two tapes represent counters. They each store a non-negative integer represented by the position of a head which can move to the left or right. If the head is in the leftmost position, the number represented is 0, and increments of one are achieved by moving the head one position to the right. We assume the ends of the tapes (on the left) are marked by $\triangleleft$. We assume the end of the input is indicated by a blank symbol $\square$.

M can test if each counter is empty but cannot compare directly the stored numbers for equality. It accepts a word if the computation with that word as input terminates in an accepting state and such that all tape heads (input and both counters) are at the leftmost position. Formally, a 2CM is a tuple $(Q, \Delta, \delta, q_0, F)$ where:

1. Q is a finite set of states, $q_0 \in Q$ is an initial state and $F \subseteq Q$ is a set of final or accepting states.
2. Δ is a finite input tape alphabet.
3. $\delta : Q \times \Delta \times \{T, F\} \times \{T, F\} \to Q \times \{1, 2, 3\} \times \{L, R\}$ is a transition function.

The interpretation of the transition function is as follows: $\delta(q, a, Z_1, Z_2) = (q', i, D)$ if before the transition M is in state q and currently reads letter a on the input tape, and Z_1 and Z_2 are T if the first and second counters are 0 respectively and F otherwise, and after the transition M is in state q', D indicates the direction in which one of the tape heads moves (L for left and R for right), and i determines which tape head moves

(1 for input head and 2 and 3 for the first and second counters respectively). Since it is possible to test if the counters are 0, we can assume there are not transitions which try to decrease the counters when they would become negative as a result. We assume that if the symbol $\triangleleft$ is read by the input tapehead (so, if the tapehead tries to move beyond the limit of the input tape), that it moves to the right in the next transition and does not change the two counters. Likewise if the input tapehead reads $\square$, it moves immediately to the left and does not change the counters.

We represent states by encoding them as consecutive non-negative integers $\{1, 2, \ldots, |Q|\}$ over a unary alphabet $\{\mathfrak{q}\}$. W.l.o.g. we may assume that the initial state q_0 is represented by $\mathfrak{q}$. To keep the notation concise, we shall not distinguish between a state and its representation, so we shall assume $Q = \{\mathfrak{q}^i \mid i \in \{1, 2, \ldots, |Q|\}$. We may also assume w.l.o.g. that the initial state is not accepting, and thus that a 2CM will always perform at least one computation step.

We can then represent a configuration of a 2CM at any point in a computation as a word belonging to $Q\Delta^*\square a^+b^+c^+$ (assuming b, c are new letters such that $\{\mathfrak{q}\}, \Delta, \{b, c\}$ are pairwise disjoint). We take $a \in \Delta$ for a technical reason which becomes clear later in the proof. The prefix from Q records the current state, the factor from Δ^* records the contents of the input tape (so, the input), and the a's b's and c's denote in unary notation the position of the input tape head and the values of the two counters. For convenience, we add one to all these values so that the sequences of a's, b's and c's are all non-empty. An initial configuration on input $w \in \Delta^*$ has the form $\mathfrak{q}w\square abc$, and a final configuration belongs to $Fw\square abc$.

A valid computation history of a 2CM M on input word w is a finite word $C = C_1C_2C_3 \ldots C_n$ such that each C_i is a configuration, C_1 is the initial configuration for the input w, C_n is a final configuration, and such that each successive pair of configurations C_i, C_{i+1} respects the transition function δ of M.

It is well-known that 2CMs can simulate the computations of Turing Machines, and therefore that they accept the class of recursively enumerable languages. Hence, a straightforward application of Rice's theorem yields that it is undecidable whether the language accepted by a 2CM contains infinitely many words from $\{0\}^+$ or not, where $0 \in \Delta$. Moreover, since 2CMs are deterministic, each word accepted by a given 2CM has exactly one valid computation history. So, it follows that the set of words from $\{0\}^+$ accepted by a 2CM M is finite if and only if the set $S_M = \{C \mid C$ is a valid computation history for M on some input word $w \in \{0\}^+\}$ is finite.

Moreover, the set S_M is finite if and only if it is regular. Indeed, if it is finite, it is trivially regular. For the converse, suppose for contradiction that it is both infinite and regular. By our assumption that each 2CM performs at least one computation step on any input, all elements of S_M will have a prefix C_0C_1 consisting of at least two configurations. Moreover, since S_M is infinite, there are infinitely many words in 0^+ accepted, and thus the configurations C_0, C_1, which both contain the input word, can be arbitrarily long. However, since S_M is regular, it is recognised by some DFA with n states. Now, for an input word from 0^+ of length greater than n, we must be able to "pump" a non-empty factor of the occurrence of the input word in C_0, without changing the occurrence in C_1. This yields a word not adhering to the syntax of S_M. Thus S_M cannot be regular.

Next, we note that S_M is regular if and only if its complement is regular. In what remains, we shall construct, for any given 2CM M, a WE-formula ψ containing a variable x such that the language expressed by x in ψ is exactly the complement of S_M. This construction thus facilitates a reduction from the finiteness problem described at the beginning of the proof to the problem of whether or not the language expressed by a variable in a WE-formula is regular.

Let $\Sigma = \{\mathfrak{q}, b, c, 0\}$. For technical reasons which shall become clear later, we shall take $a = 0$. That is, we shall use the same letter for the input words we are interested in and for the counter for the position of the input tapehead. We shall use a and 0 interchangeably, in order to highlight the role the letter is intended to play. Let us fix a 2CM M. We construct the formula ψ as the disjunction of 4 subformulas, each of which accounts for a particular way in which a word substituted for x could violate the definition of a valid computation history of M on an input from 0^+. Let $x, y_1, y_2, y_3, y_4, z_1, z_2, z_3, z_4, u, u', v, v', v'', w, w'$ be variables.

Throughout the construction we shall repeatedly use the well-known fact, established in Lemma 1, that for two words w_1, w_2, we have $w_1w_2 = w_2w_1$ if and only if they are repetitions of the same word, that is there exists a word w_3 and $p_1, p_2 \in \mathbb{N}_0$ such that $w_1 = w_3^{p_1}$ and $w_2 = w_3^{p_2}$.

Now, ψ is the formula

$$\psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4$$

where $\psi_1, \psi_2, \psi_3, \psi_4$ are defined below. The subformula ψ_1 will be satisfiable for a given value of x if x does not belong to $\mathfrak{q}0^+\Box a^+b^+c^+(Q0^+\Box a^+b^+c^+)^*$, and thus that it is not a sequence of configurations of M starting in an initial state. The subformulas ψ_2 and ψ_3 will cover the cases when x does not start with an initial configuration or end with a final configuration respectively. Finally ψ_4 will cover the case that two consecutive configurations in x do not respect the transition relation δ.

Let P be the set of pairs of letters which may not occur consecutively in $(Q0^+\Box a^+b^+c^+)^+$. That is, $P = \Sigma^2\backslash\{00, 0\Box, \Box 0, 0b, bb, bc, cc, \mathfrak{q}0, \mathfrak{q}\mathfrak{q}, c\mathfrak{q}\}$. Note that P is finite. Note also that x is not in the language $\mathfrak{q}0^+\Box a^+b^+c^+(Q0^+a^+b^+c^+)^*$ if and only if one of the following hold:

- it is empty, or
- it contains consecutive letters included in P, or
- it contains a factor of the form $0^+\Box 0^+A$ for $A \neq b$, or
- it starts with a letter other than $\mathfrak{q}$ or
- it starts with more than one $\mathfrak{q}$, or
- it ends with a letter other than c, or
- it contains more than $|Q|$ consecutive occurrences of $\mathfrak{q}$.

Thus the subformula ψ_1 is given by:

$$x \doteq \varepsilon \vee \bigvee_{AB \in P} x \doteq uABv$$

$$
\begin{aligned}
&\vee \left(\bigvee_{A \in \Sigma \setminus \{\mathfrak{b}\}} x \doteq u0u' \square 0v' A v \wedge u'0 \doteq 0u' \wedge v'0 \doteq 0v' \right) \\
&\vee \bigvee_{A \in \Sigma \setminus \{\mathfrak{q}\}} x \doteq Au \\
&\vee \; x \doteq \mathfrak{q}\mathfrak{q}u \\
&\vee \bigvee_{A \in \Sigma \setminus c} x \doteq uA \\
&\vee \; x \doteq u\mathfrak{q}^{|Q|+1}v.
\end{aligned}
$$

With ψ_2, we want to enforce that it is true only if x has a prefix other than the initial configuration, namely $\mathfrak{q}w\square abc$ for some $w \in 0^+$. We only need to cover cases when ψ_1 is not satisfied (so we may assume that x belongs to $\mathfrak{q}0^+\square a^+b^+c^+(Q0^+a^+b^+c^+)^*$. Thus ψ_2 is given as:

$$
u0 \doteq 0u \wedge \left(\bigvee_{A_1A_2A_3A_4 \neq \square abc,\ A_1 \neq 0} x \doteq \mathfrak{q}uA_1A_2A_3A_4v \right).
$$

In the above formula, u must be the complete sequence of 0s occurring after $\mathfrak{q}$. The cases when the next four letters after u are not $\square abc$ are then covered by the disjunction. The subformula ψ_3 can be constructed similarly as follows:

$$
u0 \doteq 0u \wedge \left(\bigvee_{A_0A_1A_2A_3A_4 \neq 0\square abc} x \doteq vA_0A_1A_2A_3A_4 \vee \bigvee_{q \in Q \setminus F} x \doteq vcqu\square abc \right).
$$

The first of the two disjuncts inside the brackets covers all cases when x does not end with a configuration of the form $\mathfrak{q}^i0^*\square abc$, or in other words, when all tape heads have not returned to their leftmost positions. The second disjunct covers the cases when tape heads are in their leftmost positions but the state is not final.

Finally we construct ψ_4 as

$$
\begin{aligned}
&\bigvee_{q,q' \in Q} (x \doteq uqy_1\square y_2y_3y_4q'z_1\square z_2z_3z_4v \\
&\quad \wedge (v \doteq \varepsilon \vee v \doteq \mathfrak{q}w) \wedge (u \doteq \varepsilon \vee u \doteq w'c) \\
&\quad \wedge \; y_10 \doteq 0y_1 \\
&\quad \wedge \; y_2a \doteq ay_2 \\
&\quad \wedge \; y_3b \doteq by_3 \\
&\quad \wedge \; y_4c \doteq cy_4 \\
&\quad \wedge \; z_10 \doteq 0z_1
\end{aligned}
$$

$$
\begin{aligned}
&\wedge\ z_2 a \doteq a z_2 \\
&\wedge\ z_3 b \doteq b z_3 \\
&\wedge\ z_4 c \doteq c z_4 \\
&\wedge\ (y_1 0 u' \doteq z_1 \vee y_1 \doteq z_1 0 u' \ \vee \\
&\quad\ y_2 a a u' \doteq z_2 \vee y_2 \doteq z_2 a a u' \ \vee \\
&\quad\ y_3 b b u' \doteq z_3 \vee y_3 \doteq z_3 b b u' \ \vee \\
&\quad\ y_4 c c u' \doteq z_4 \vee y_4 \doteq z_4 c c u' \ \vee \\
&\quad\ \bigvee_{\psi' \in D} \psi'))
\end{aligned}
$$

where D is a set of formulas describing transitions which are not possible in M, which is again given below. The first 10 lines of ψ_4 enforce that $q y_1 y_2 y_3 y_4$ and $q' z_1 z_2 z_3 z_4$ are consecutive configurations in x and that q, q' represent states, while y_1, z_1 are the parts containing 0's, y_2, z_2 contain the a's y_3, z_3 contain the b's and $y_4 z_4$ contain the c's. Specifically, assuming that ψ_1, ψ_2, ψ_3 are not satisfied (and so x represents a sequence of syntactically correct configurations), then for each consecutive pair of configurations $C_i C_{i+1}$ in x, there is an assignment for $u, v, y_1, y_2, y_3, y_4, z_1, z_2, z_3, z_4$ satisfying lines 1-10 such that each variable represents the appropriate part of one of the configurations as described above. Moreover, any assignments not satisfying ψ_1, ψ_2, ψ_3 which satisfy lines 1-10 will adhere to this condition for some consecutive configurations in x.

The 11th line accounts for when the input word is not correctly copied from the previous configuration to the next. Specifically, assuming the previously described parts of the formula are satisfied by some assignment, y_1 and z_1 must represent the input word for two consecutive configurations C_i and C_{i+1}. We have already covered any cases where the input word for some configuration does not belong to 0^+ (namely through ψ_1), so we only need to cover the case that they have different lengths. Assuming $y_1, z_1 \in 0^+$, the subformula $y_1 0 u' \doteq z_1$ is satisfiable for some u' if and only if z_1 is longer than y_1. The case when z_1 is shorter than y_1 is handled symmetrically by $y_1 \doteq z_1 0 u'$.

Line 12 works similarly for the first "counter", which represents the position of the head on the input tape. This value (represented by the lengths of y_2 and z_2) can change by zero or one between consecutive configurations in a valid computation history. Thus we need to cover the cases when one of the counters changes length by at least two. Under the assumption that $y_2, z_2 \in a^+$, $y_2 a a u' \doteq z_2$ is satisfiable if and only if z_2 has length at least two more than y_2. The case that y_2 has length at least two more than z_2 is handled symmetrically by $y_2 \doteq z_2 a a u'$. The same condition, that the length cannot change by more than one, is imposed on the second two counters by lines 13 and 14, which work identically to line 12.

We conclude with the subformulas from D, which must be satisfiable when the input is copied correctly, tapehead positions/counters do not move by more than one, but the transition is still not valid. The first formula in D is the following, where $B_2 = a$, $B_3 = b$, $B_4 = c$, which covers the case that two tapeheads/counters move at

the same time.

$$\bigvee_{i,j\in\{2,3,4\}\wedge i\neq j} ((y_iB_i \doteq z_i \vee y_i \doteq z_iB_i) \wedge (y_jB_j \doteq z_j \vee y_j \doteq z_jB_j))$$

Next, D contains the following formulas which cover the cases when the input tapehead reads $\square$ or $\triangleleft$. Note that $\triangleleft$ is read when the input tapehead position is 0, so when $y_2 = a$, and $\square$ is read when the input tapehead position is $|w|$ where w is the input word, so when $|y_1| = |y_2|$. Since we can't directly compare the lengths of two variables, this is why we choose $a = 0$, so comparing the lengths of y_1 and y_2 becomes the same as checking their direct equality.

$$y_2 \doteq a \wedge (z_2 \doteq a \vee y_3 \doteq z_3b \vee y_3b \doteq z_3 \vee y_4 \doteq z_4c \vee y_4c \doteq z_4) \ \vee$$
$$y_2 \doteq y_1 \wedge (z_2 \doteq y_2a \vee z_2 \doteq y_2 \vee y_3 \doteq z_3b \vee y_3b \doteq z_3 \vee y_4 \doteq z_4c \vee y_4c \doteq z_4)$$

The first line above accounts for when the tapehead reads $\triangleleft$ (so is at the leftmost part of the input tape) and either one of the two counters changes in the next configuration, or the tapehead position does not increase in the next configuration. The case where the tapehead position reduces further correspond to when $y_2 = \varepsilon$, and are already handled by ψ_1. The second line accounts for when the tapehead reads $\square$ (so is at the rightmost part of the input) and either the tapehead position remains the same, increases, or one of the two counters changes in the next configuration.

Finally, for every syntactically correct transition not adhering to δ (so every transition having the correct form but not allowed in the specific 2CM M), D contains a formula accounting for the case that this transition takes place in the computation history. Due to previous subformulas we only need to cover transitions where the input symbol is a 0 and where the states are q, q' respectively. Suppose $Z_1, Z_2 \in \{T, F\}$, $D \in \{L, R\}$, $i \in \{1, 2, 3\}$ and such that $\delta(q, 0, Z_1, Z_2) \neq (q', i, D)$. Then D will include a formula ψ' which is a conjunction of the following:

- if $Z_1 = T$, then $y_3 \doteq b$ and if $Z_2 = F$ then $y_3 \doteq bbv'$,
- if $Z_2 = T$, then $y_4 \doteq c$ and if $Z_3 = F$ then $y_4 \doteq ccv''$,
- if $D = R$, then $z_{i+1} \doteq y_{i+1}A_i$, and if $D = L$ then $z_{i+1}A_i \doteq y_{i+1}$,

where $A_1 = a, A_2 = b, A_3 = c$. For example, if $\delta(q_1, 0, T, F) \neq (q_2, 1, R)$, D would include the subformula

$$y_3 \doteq b \wedge y_4 \doteq ccv'' \wedge z_2 \doteq y_2a.$$

By similar arguments as above ψ' is satisfiable whenever there are consecutive configurations in x which are related according to the (illegal) transition $\delta(q_1, 0, Z_1, Z_2) = (q_2, i, D)$.

All together, we have shown a construction for a formula ψ which can be satisfied for a particular value of x if and only if x is not a valid computation history for M on input word of the form 0^+. In other words, the language expressed by x in ψ is exactly the complement of S_M. Thus it is regular if and only if M accepts only finitely many words from 0^+. This completes the reduction and we conclude that the problem

of deciding whether a formula and variable from WE express a regular language is undecidable as claimed. □

One way of explaining Theorem 10 is that although we are checking for a "simple" property, the (representations of the) objects we are looking at are complicated. An analogy can be drawn e.g. to checking simple or trivial properties of languages accepted by Turing machines.

Just as interesting, if not more so, is the converse problem. Unfortunately, we leave this problem open, however the rest of this section is devoted to providing some further insights.

Open Problem 2 *Is it decidable whether a regular language is expressible by word equations?*

In this case, we are rather asking whether relatively simple objects possess a particular property. Answering this question in either the positive or negative would provide some insight as to whether it is the class of languages $\mathcal{L}(\text{WE})$ which itself contains inherent computational complexity, or whether that complexity rather arises more from the way in which the languages are represented.

Some intuition for Open Problem 2 can be gained by looking at some examples which are or are not expressible. It is trivial that $\emptyset$ and $\{a\}$ for each $a \in \Sigma$ are expressible in WE. All regular languages can be obtained from these languages by taking closure under union, concatenation and Kleene star. It is also straightforward to see that $\mathcal{L}(\text{WE})$ is closed under both concatenation and union. However since there are regular languages which are not expressible (see e.g. Lemma 6), we may directly infer that $\mathcal{L}(\text{WE})$ is not closed under Kleene star. Nevertheless there are some examples of languages L for which $L^* \in \mathcal{L}(\text{WE})$.

Proposition 11 *Let $w \in \Sigma^+$, and $\mathcal{E} \subseteq \mathbb{N}$. Let $Y = \{w^i : i \in \mathcal{E}\}$. Then the language Y^* is expressible in* WE.

Proof If $|\mathcal{E}| = 0$, then $Y^* = \emptyset^* = \{\varepsilon\}$, which is finite and thus expressible. So let us assume hereafter that $|\mathcal{E}| \geq 1$. We shall make use of the following well-known result:

Theorem 12 (Weakened Schur's Theorem) *Let $\{m_1, ..., m_t\} \subset \mathbb{N}$ be relatively prime. Then for every sufficiently large $n \in \mathbb{N}$, n can be written as a linear combination of the numbers $m_1, ..., m_t$. with non-negative integer coefficients.*

Let $K = \gcd(\mathcal{E})$. Note that K cannot exceed any $i \in \mathcal{E}$. Then for each $i \in \mathcal{E}$, we can write $i = \tilde{\imath} K$ for some $\tilde{\imath} \in \mathbb{N}$. It is clear that the set $\tilde{\mathcal{E}} = \{\tilde{\imath} : i \in \mathcal{E}\}$ must then be relatively prime. Let $u = w^K$. Then for each $i \in \mathcal{E}$, we have $w^i = w^{\tilde{\imath} K} = u^{\tilde{\imath}}$. So $Y = \{u^{\tilde{\imath}} : \tilde{\imath} \in \tilde{\mathcal{E}}\}$.

Fix now some $\tilde{\jmath} \in \tilde{\mathcal{E}}$. Let D be the set of divisors of $\tilde{\jmath}$ which are greater than 1. D must be finite. For every $d \in D$, we must be able to find some $\tilde{\jmath}_d \in \tilde{\mathcal{E}}$ which does not have d as a divisor. (If we couldn't find some $\tilde{\jmath}_d$, it would contradict the fact that $\tilde{\mathcal{E}}$ is relatively prime). Then introduce the set $\mathcal{J} = \{\tilde{\jmath}\} \cup \{\tilde{\jmath}_d : d \in D\}$. $\mathcal{J}$ is then a finite subset of $\tilde{\mathcal{E}}$ which is relatively prime. (Indeed, its gcd must be a divisor of $\tilde{\jmath}$, but our construction rules out every divisor of $\tilde{\jmath}$ which is greater than 1).

We now apply Theorem 12 to the set $\mathcal{J}$. It tells us that there exists $N \in \mathbb{N}$, such that every integer $n > N$ can be written as a (finite) linear combination $n = \sum_{\tilde{\imath} \in \mathcal{J}} \alpha_{\tilde{\imath}} \tilde{\imath}$,

with each coefficient $\alpha_{\tilde{\imath}} \in \mathbb{N}_0$. Then for every $n > N$, we can write the word u^n in the (finite) form

$$u^n = u^{(\sum_{\tilde{\imath} \in \mathcal{J}} \alpha_{\tilde{\imath}} \tilde{\imath})} = \prod_{\tilde{\imath} \in \mathcal{J}} (u^{\tilde{\imath}})^{\alpha_{\tilde{\imath}}}.$$

Each of the (finitely many) $u^{\tilde{\imath}}$ in the product here is a member of Y. Therefore, for every $n > N$, $u^n \in Y^*$.

Thus Y^* can be obtained from $\{u\}^*$ by the removal of only finitely many words $v_1, ...v_k$. Therefore, by Lemma 1, we can express Y^* via the variable y in the WE-formula:

$$uy \doteq yu \wedge y \doteq z^m \wedge \bigwedge_{r \in \{1,...,k\}} \neg(y \doteq v_r)$$

where q is the primitive root of u, and $u = q^m$. Specifically, it follows from Lemma 1 that $uy \doteq yu$ implies that u, y share a primitive root, which must be q, so $y \in \{q\}^*$. The equation $y \doteq z^m$ similarly implies z has primitive root q, and thus that $y \in \{q^m\}^* = \{u\}^*$. It is clear that any $y \in \{u\}^*$ can be part of a satisfying assignment, and the final conjunct handles the "removal" of the words $v_1, ..., v_k$. □

On the other hand, it is shown in [32] that if $\Sigma \supset \{a, b\}$, then $\{a, b\}^*$ is not expressible in WE. As we shall later see, similar arguments can be made to show that S^* is often not expressible in WE when S contains two words with distinct primitive roots. We can partition the regular languages into two subclasses based on whether or not they can be generated by a regular expression which only applies Kleene star to subexpressions matching the form given in Proposition 11. It follows from closure of $\mathcal{L}$(WE) under union and concatenation that those which can are expressible in WE. In what follows, we focus on showing inexpressibility for those which cannot.

As in the previous section, we shall make use of the framework from [32] to show inexpressibility. This generally involves two distinct challenges: firstly setting up the conditions to allow a swap of factors to take place, and secondly choosing an appropriate factor to swap. The second step is trivial in the case of thin languages, for which there is a factor not occurring in any word in the language. We therefore handle thin languages first, where we are able to characterise those regular languages which are/are not expressible in WE, and look at languages which are dense (i.e. not thin) in a subsequent section.

Definition 13 (Thin and Dense Languages) *A forbidden factor of $X \subseteq \Sigma^*$ is a word $z \in \Sigma^*$ which is not a factor of any $w \in X$. X is called thin if it has a forbidden factor. X is called dense if it has no forbidden factor (i.e., if it is not thin).*

4.1 (In)Expressibility of Thin Regular Languages

In this section, we provide in Theorem 14 a characterisation of thin regular languages expressible in WE. Since thin languages have a forbidden factor, Step 4 in the general approach to showing inexpressibility outlined in Section 2.2 becomes trivial, and we therefore focus more on steps 2 and 3. We begin by introducing a family of factorisation schemes designed for following step 2 in a general setting. The main aim with these factorisation schemes is to be able to guarantee a large number of distinct $\mathfrak{F}$-factors

flexibly, across a large class of languages. To achieve this, the factorisation schemes are parametrised by a word u, and informally can be thought of as "splitting" a word at every point where an occurrence of u occurs as a factor. For instance, if $u = aa$, then the corresponding factorisations of the words $abaaabbaab$, a^5 and $bbbbbba$ are $(ab, a, aabb, aab)$, (a, a, a, aa), and $(bbbbbba)$ respectively. Formally, we define these factorisation schemes as follows.

Definition 14 *For $u \in \Sigma^+$ and $w \in \Sigma^*$, let $\mathcal{I}_{u,w} = \{i \mid 1 < i \leq |w| - |u| + 1 \mid w[i : i + |u|] = u\}$. Let $\mathfrak{F}_u$ be the factorisation scheme which maps a word w to the factorisation $(w[1 : i_1], w[i_1 : i_2], \ldots, w[i_k : |w| + 1])$ where $\mathcal{I}_{u,w} = \{i_1, i_2, \ldots, i_k\}$ with $i_1 < i_2 < \ldots < i_k$.*

In other words, the start of w and the start of each occurrence of u in w, are exactly the positions in w from which $\mathfrak{F}_u$-factors of w begin. Note that if w does not contain the word u, or if u only occurs as a prefix of w, then the $\mathfrak{F}_u$-factorisation of w is simply (w). Otherwise, every i_j denotes a position in w at which an occurrence of u begins.

Lemma 10 *For any $u \in \Sigma^+$, $\mathfrak{F}_u$ is synchronising.*

Proof Let $u \in \Sigma^+$. It is immediate from the definition that $\mathfrak{F}_u$ is complete and uniquely deciphering. We shall now show that $\mathfrak{F}_u$ satisfies the synchronising condition for $l = 2$ and $r = |u| + 1$. Let us take $l' = 1 \leq l$ and $r' = 1 \leq r$. Let $x_1, ..., x_s$ and $y_1, ..., y_k$ be the $\mathfrak{F}_u$-factorisations of some arbitrary words $x, y \in \Sigma^*$, respectively, and let $U = y_1 \ldots y_{l'}$ and $V = y_{k-r'+1} \ldots y_k$. Suppose that $k > l + r = |u| + 3$, and that y occurs in x at position i. Note that this implies that y (and therefore also x) both contain at least one occurrence of u. We now show that this choice of l', r' makes all four criteria of the "synchronising condition" hold. Indeed,

- At the position $|y_1|$ in y, the $\mathfrak{F}_u$-factor y_2 begins. Hence the factor u occurs in y in its position $|y_1|$. Hence the factor u occurs in x in its position $i + |y_1| = i + |u|$. Hence an $\mathfrak{F}_u$-factor x_p of x begins at position $i + |U|$ of x. Similarly, at the position $|y| - |y_k|$ in y, the $\mathfrak{F}_u$-factor y_k begins. Hence the factor u occurs in y at the position $|y| - |y_k|$. Hence the factor u occurs in x at the position $i + |y| - |y_k| = i + |y| - |V|$. Hence an $\mathfrak{F}_u$-factor x_q of x begins at the position $i + |y| - |V|$ of x.
- Since $x_p \cdots x_{q-1}$ and $y_2 \cdots y_{k-1}$ are factors starting and ending in the same positions in x, we necessarily have that $x_p \cdots x_{q-1} = y_2 \cdots y_{k-1}$. Moreover, since u occurs in y at position $|y| - |y_k|$, and since y_k is a suffix of y, we necessarily have $|y_k| \geq |u|$. Thus, (since y is a factor of x) both x and y extend for at least $|u|$ letters after the ends of the factors $x_p \cdots x_{q-1}$ and $y_2 \cdots y_{k-1}$, respectively. Hence the positions within $x_p \cdots x_{q-1}$ and $y_2 \cdots y_{k-1}$ corresponding to positions in x and y where the factor u occurs will be identical. It follows that the sequences of $\mathfrak{F}_u$-factors $x_p, ..., x_{q-1}$ and $y_2, ..., y_{k-1} = (y_{l'+1}, ..., y_{k-r'})$ are also identical.
- Suppose for contradiction that the occurrence of U at position i in x covers more than one $\mathfrak{F}_u$-factor of x. Then there is some "splitting point" of x properly within the occurrence of U. Then the factor u begins in x somewhere properly within the occurrence of U. But then the factor u must begin in y somewhere properly between its positions 0 and $|y_1|$. (y cannot end before this factor u completes, since y contains a full occurrence of the factor u at its later position $|y_1|$). So a "splitting

point" exists in y somewhere properly between its positions 0 and $|y_1|$. This is a contradiction, as y did not get split at such a position. So by contradiction, the occurrence of U at position i in x covers at most $1 = l - 1$ $\mathfrak{F}_u$-factor of x.

- Suppose for contradiction that the occurrence of V at position $i + |y| - |V|$ in x covers more than $|u|$ $\mathfrak{F}_u$-factors of x. Then there are more than $|u| - 1$ "splitting points" of x which are properly within the occurrence of V. Now let ι be the position in x of the leftmost "splitting point" in x which is properly within the occurrence of V. Because of how the "splitting points" are chosen, the factor u must begin in x at its position ι. But because there are so many "splitting points" to the right of ι within the occurrence of V, this instance of the factor u must also end before V does. Then there is an occurrence of the factor u properly contained within the $\mathfrak{F}_u$-factor y_k of y. This is a contradiction, and so our assumption must have been incorrect: the occurrence of V at position $i + |y| - |V|$ in x can cover at most $|u| = r - 1$ $\mathfrak{F}_u$-factors of x.

The entire definition is satisfied, and so $\mathfrak{F}_u$-factorisation is synchronising. □

An intuitive reason why Lemma 10 holds is that the "splitting points" in the $\mathfrak{F}_u$-factorisation procedure are defined locally. Hence the "splitting points" --- and thus the $\mathfrak{F}_u$-factors of a factor y of a word x will coincide after only a small (bounded by $|u|$) number of letters (and therefore factors). The following example demonstrates how we can use this family of synchronising $\mathfrak{F}$-factorisations in line with the general approach from [32].

Example 2 *Let us show that* $L = \{ab, ba\}^*$ *is inexpressible in* WE. *Suppose for contradiction that L is expressed by a variable x in some word equation E. By Lemma 10, the factorisation scheme* $\mathfrak{F}_{aa}$ *is synchronising. We shall apply Lemma 5. Let c be the constant referenced by the lemma. Then consider the word*

$$w = \prod_{j \in (1,\ldots,c)} \left(ba \cdot (ab)^j \right) \in L.$$

Let h be a solution to E such that $h(x) = w$. *Clearly,* $n_{\mathfrak{F}_{aa}}(w) = c + 1 > c$, *and so by Lemma 5, there is at least one* $\mathfrak{F}_{aa}$*-factor u of w which is unanchored in h. Thus, for* $v = aaa$, *we also have* $w' = h^{\mathfrak{F}_{aa}}_{u \to v}(x) \in L$. *However aaa is a factor of* w', *so* $w' \notin L$, *a contradiction. The assumption that L was expressible in* WE *must have been incorrect, so L is inexpressible in* WE.

Notice in the previous example, that by carefully choosing the word u on which the factorisation scheme $\mathfrak{F}_u$ is based, we are able to produce a family of words $w \in L$ with arbitrarily many $\mathfrak{F}_u$ factors. Since the language $\{ab, ba\}^*$ is thin, having e.g. aaa as a forbidden factor, the rest of the proof then becomes straightforward.

In our next main result, Theorem 13, we generalise this reasoning to work for all thin sets of the form Y^*, and thus provide greater insight into when languages obtained by an application of Kleene star are expressible in WE. Cases where Y contains only repetitions of a single word w are covered already by Proposition 11. Thus we focus

on cases where Y contains words w_1, w_2 which do not share a primitive root. For the more general case, we shall consider the $\mathfrak{F}_{w_1^i}$-factorisation of words

$$w = \prod_{j \in J} w_1^i (w_2 w_1)^j w_2 w_1^i \in L.$$

for some suitably chosen tuple $J \subset \mathbb{N}$.

As we show in Lemma 11, $(w_1 w_2 w_1)$ is never a factor of w_1^i when w_1, w_2 don't share a primitive root. This allows us to keep track precisely of where the splitting points occur, and thus see that as j grows, there must exist longer and longer $\mathfrak{F}_{w_1^i}$-factors of w. It is this reasoning that will permit us to conclude that $n_{\mathfrak{F}_{w_1^i}}(w) > c$ for whatever value of c arises from Lemma 5.

Lemma 11 *Given words $w_1, w_2 \in \Sigma^+$, either*

(1) $\exists u \in \Sigma^+$, $\exists p_1, p_2 \in \mathbb{N}$ such that $w_1 = u^{p_1}$ and $w_2 = u^{p_2}$, (so w_1, w_2 share a primitive root and thus commute), or
(2) $w_1 w_2 w_1$ is not a factor of u^i for all $i \in \mathbb{N}$

Proof Let u be the primitive root of w_1. We shall assume (2) does not hold. In particular, we let $i \in \mathbb{N}$ and assume that $w_1 w_2 w_1$ is a factor of u^i. Then there exists a prefix U and a suffix V of u^i such that

$$U u w_2 u V = u^i.$$

It follows that $U = u^r u'$ where u' is a proper prefix of u and $r \in \mathbb{N}_0$. Let u'' be the corresponding suffix of u, so $u = u'u''$. Cancelling u^r from both sides yields that $u'u$ is a prefix of u^{i-r}. Clearly, uu' is also a prefix of u^{i-r} and both have the same length, we must have $u'u = uu'$. However since u is primitive, by Lemma 1, $u' = \varepsilon$. Hence $U \in \{u\}^*$. A symmetrical argument means that $V \in \{u\}^*$ also holds. By cancelling the prefix Uu and suffix uV, we get that $w_2 \in \{u\}^*$. Thus (1) holds. □

Our next lemma provides the analysis of the $\mathfrak{F}_{w_1^i}$-factors occurring in the word $w = \prod_{j \in J} w_1^i (w_2 w_1)^j w_2 w_1^i$ which will form the basis of what follows.

Lemma 12 *Let $w_1, w_2 \in \Sigma^+$ be words which do not commute. Let $i \in \mathbb{N}$ such that $|w_1^i| > |w_1 w_2 w_1|$. Let J be a finite tuple of integers greater than 2, and let $w = \prod_{j \in J} w_1^i (w_2 w_1)^j w_2 w_1^i$. Then there exist $v, v' \in \Sigma^*$ such that:*

- *v is a prefix of $w_1 w_2$, and*
- *v' is a suffix of $w_2 w_1$, and*
- *for each $j \in J$, $j \geq 2$, $w_1^i v' (w_2 w_1)^{j-2} v$ is an $\mathfrak{F}_{w_1^i}$-factor of w, and*
- *all other $\mathfrak{F}_{w_1^i}$-factors of w have length at most $|w_1^i|$.*

Proof Let w_1, w_2, i, J, w be defined according to the lemma. Let $\mathfrak{F} = \mathfrak{F}_{w_1^i}$. For each $j \in J$, consider the factors $W_j = w_1^i (w_2 w_1)^j w_2 w_1^i$ of w.

Let v be the shortest prefix of w_1w_2 such that there is an occurrence of w_1^i starting at position $|W_j| - |w_1w_2w_1^i| + |v| + 1$ in W_j. Let v' be the shortest suffix of w_2w_1 such that there is an occurrence of w_1^i starting at position $|w_1w_2| - |v'| + 1$ in W_j. Notice that v, v' are well-defined because there are occurrences of w_1^i starting at positions 1 and $|W_j| - |w_1^i| + 1$, and notice also that v, v' will be the same for all j.

By Lemma 12, no occurrence of w_1^i can have $w_1w_2w_1$ as a factor, so every occurrence of w_1^i in W_j must start in a position no greater than $|w_2w_1|$ of W_j, or no smaller than $|W| - |w_1w_2w_i| + 1$ if W_j. It follows that $w_1^iv'(w_2w_1)^{j-2}v$ is a $\mathfrak{F}$-factor of w for all j. It also follows that all other $\mathfrak{F}$-factors are contained entirely within an occurrence of w_1^i in w. Thus, all four conditions of the lemma are satisfied.

Armed with Lemma 12, we are now ready to prove our theorem regarding expressibility of thin languages Y^*.

Theorem 13 *Let $Y \subset \Sigma^+$ such that Y^* is thin. Then Y^* is expressible in* WE *if and only the words in Y are pairwise commutative.*

Proof The "if" direction follows directly from Proposition 11. Thus we shall concentrate on the "only if" direction, for which we prove the contrapositive. To that end, suppose that Y contains a pair of words w_1, w_2 which do not commute. Note that $w = \prod_{j \in J} w_1^i(w_2w_1)^j w_2w_1^i$ belongs to Y^*. Suppose for contradiction that Y^* is expressed by a variable x in some word equation E.

Let i be the smallest integer such that $|w_1^i| \geq |w_1w_2w_1|$. Let us take $\mathfrak{F}$ to be the $\mathfrak{F}_{w_1^i}$-factorisation of words from Σ^*. We have shown this $\mathfrak{F}$ to be synchronising in Lemma 10. Hence we can apply Lemma 5. Let c be the constant provided by this lemma. Let $J = (3, 4, \ldots, c+3)$. Lemma 12 tells us that there exist w', w'' such that for each $j \in J$, the word $u_j = w_1^iw''(w_2w_1)^{j-2}w'$ is an $\mathfrak{F}$-factor of w. Since the u_j all have different lengths, they are all distinct. There are $c+1$ such factors u_j, and hence we must have $n_{\mathfrak{F}_{w_1^i}}(W) > c$.

Recall that since $W \in Y^*$ and Y^* is expressed by x in E, there is a solution h to E satisfying $h(x) = W$. By Lemma 5, there is at least one u_j which is unanchored w.r.t. h. Since Y^* is thin, by definition, there exists $v \in \Sigma^*$, such that no word from Y^* has v as a factor. However, by Lemma 5, $w' = h^{\mathfrak{F}}_{u_j \to v}(x)$ is also in the language expressed by x in E, and so is in Y^*. However w' has v as a factor. This is a contradiction, and hence our assumption that Y^* was expressible must have been wrong, and Y^* is indeed inexpressible in WE.

Notice that, in the above theorem, there is no requirement on Y being finite. On the other hand, Y^* being thin precludes the possibility that $Y = \Sigma$. In particular Σ^* is an example of a WE-expressible language of the form Y^* where Y does contain a pair of non-commuting words. Theorem 13 provides a full characterisation of when the Kleene star of a set Y is expressible in WE in the case that Y^* is thin. We now turn our attention to extending this result to all thin regular languages. To do so, it is convenient to consider regular expressions. For the sake of clarity and completeness, we recall the definition here.

Definition 15 (Regular Expressions) *$\emptyset$, ε and every $a \in \Sigma$ is a regular expression. Moreover, for regular expressions e_1, e_2, each of the following is also a regular expression:*

- $e_1 e_2$,
- $e_1 | e_2$,
- (e_1),
- e_1^*.

For a regular expression e, we denote by $L(e)$ the (regular) language generated by e.

Since regular expressions capture the regular languages directly as a result of their closure properties, and since we also investigate the WE-expressible languages from this perspective, it is convenient to use regular expressions in our later characterisation of when a thin regular language is expressible in WE (Theorem 14). To do so, we restrict slightly the form of the regular expressions that we consider. We therefore provide the notion of well-formed regular expressions below.

Definition 16 (Well-formed regular expressions) *Let e be a regular expression. We say that e is well-formed if one of the following two conditions holds:*

- *$e = \emptyset$ (and therefore $L(e) = \emptyset$), or*
- *e does not contain the symbol $\emptyset$.*

Remark 7 *Since we allow ε as a regular expression, it is easily shown that well-formed regular expressions still generate the full class of regular languages.*

Before proving Theorem 14, we need the following lemma.

Lemma 13 *Let e be a well-formed regular expression. Then $L(e)$ is thin if and only if, for every subexpression Y^* of e, $L(Y^*)$ is thin.*

Proof For the "if" direction, the contrapositive follows straightforwardly from the definitions: if e is a well-formed regular expression with some subexpression Y^* where $L(Y)^*$ is not thin (and therefore dense), then every possible word can occur as a factor of the subexpression Y^*, and thus of a word in $L(e)$. It follows that $L(e)$ is also dense.

Consider now the "only if" direction. In the degenerate case $e = \emptyset$, the statement is trivial. For cases when $e \neq \emptyset$, we proceed by induction. Suppose that, for every subexpression Y^* $L(Y)^*$ is thin, and that for all subexpressions of e, the statement of the lemma holds, and thus that the language of each subexpression is also thin. The base cases are $e = \varepsilon$ and $e = a \in \Sigma$. Clearly in these cases, $L(e)$ is finite, and thus thin. Otherwise, we have three cases as follows.

- Suppose that $e = (e')^*$, for a well-formed regular expression e'. By our assumptions, we know that $L(e')$ is thin and moreover that $L(e')^*$ is thin. Thus $L(e) = L((e')^*) = L(e')^*$ is thin and the statement of the lemma holds.
- Suppose that $e = e_1 | e_2$, for well-formed regular expressions e_1, e_2. By our assumptions, $L(e_1)$ and $L(e_2)$ are both thin. Thus, there exists $u_1 \in \Sigma^*$ which is not a factor of any word $w \in L(e_1)$, and $u_2 \in \Sigma^*$ which is not a factor of any word $w \in L(e_2)$. It follows that $u_1 u_2$ is not a factor of any $w \in L(e)$, so $L(e)$ is thin and the statement of the lemma holds.

- Suppose that that $e = e_1 e_2$, for regular expressions e_1, e_2. By our assumptions, $L(e_1)$ and $L(e_2)$ are both thin, so there exists $u_1 \in \Sigma^*$ which is not a factor of any $w \in L(e_1)$, and $u_2 \in \Sigma^*$ which is not a factor of any $w \in L(e_2)$. It follows that $u_1 u_2$ is not a factor of any $w \in L(e)$, so $L(e)$ is thin and the statement of the lemma holds.

In all cases, if the statement of the lemma holds for subexpressions, it holds for the whole expression too and by induction, it holds in general. □

We are now able to give the characterisation of when a thin regular language is expressible in WE. As one might expect, it involves incorporating our characterisation for languages of the form Y^* into the structure of well-formed regular expressions. Formally, the statement is given as follows.

Theorem 14 *Let $L \subseteq \Sigma^*$ be regular and thin. Let e be a well-formed regular expression for L. Then L is expressible in* WE *if and only if, for every subexpression Y^* of e, the words of $L(Y)$ are pairwise commutative.*

Proof We begin with the "if" direction. Let e be a well formed regular expression and suppose that, for every subexpression Y^* of e, the words in $L(Y)$ are pairwise commutative. By Lemma 13, for every subexpression Y^* of e, $L(Y)^*$ is also thin. Consequently, by Theorem 13, for every subexpression Y^* of e, $(L(Y))^* = L(Y^*)$ is expressible in WE. Now $L(e)$ can be produced by applying the operations of concatenation and union to:

- the languages $L(Y^*)$ for subexpressions Y^* of e
- the languages $\emptyset$, $\{\varepsilon\}$, and $\{a\} \subseteq \Sigma$,

It is already shown in [32] that every finite language is expressible in WE, and that the WE-expressible languages are closed under union and concatenation. We have already concluded that $L(Y^*)$ for each subexpression Y^* is also expressible, therefore $L(e) = L$ is also expressible.

For the "only if" direction, we prove the contrapositive. The proof follows similar lines to that of Theorem 13. Suppose that e is a well-formed regular expression generating a thin regular language $L = L(e)$, and that there exists a subexpression Y^* of e, and two words $w_1, w_2 \in L(Y)$ which do not commute. Clearly, L must be nonempty, since e is well-formed and contains symbols other than $\emptyset$. Let us now assume for contradiction that L is expressible in WE. In particular, by Lemma 3, we may assume that L is expressed by a variable x in a single word equation E.

Let i be the smallest integer such that $|w_1^i| \geq |w_1 w_2 w_1|$. Let $\mathfrak{F} = \mathfrak{F}_{w_1^i}$ and recall from Lemma 10 that $\mathfrak{F}$ is synchronising for some constants r, ℓ. Let c be the constant provided by Lemma 5, and let $J = (3, 4, \ldots, c + r + \ell + 3)$. Let $W = \prod_{j \in J} w_1^i (w_2 w_1)^j w_2 w_1^i$. Then $W \in L(Y^*)$, and thus W is a factor of some word in L. That is to say, there exists $U, V \in \Sigma^*$ such that $\mathcal{W} = UWV \in L$.

By Lemma 12, there exist w', w'' such that for each $j \in J$, the factor $u_j = w_1^i w'' (w_2 w_1)^j w'$ is an $\mathfrak{F}$-factor of W. Consequently, since each u_j has a different length, there are at least $|J| = c + r + \ell + 1$ distinct factors u_j. By definition of a synchronising factorisation scheme, since W is a factor of $\mathcal{W}$, there are at most $r + \ell$

$\mathfrak{F}$-factors u_j of W which are not $\mathfrak{F}$-factors of $\mathcal{W}$. Thus there are at least $c+1$ distinct $\mathfrak{F}$-factors of $\mathcal{W}$.

Consequently, we can follow the same steps as usual to obtain a contradiction: firstly, we conclude by the above described analysis that $n_{\mathfrak{F}}(\mathcal{W}) > c$. Moreover, since $\mathcal{W} \in L$, there exists a solution h to the equation E such that $h(x) = \mathcal{W}$. Thus, by Lemma 5, there is at least one $\mathfrak{F}$-factor which is unanchored in h. Since L is thin, there exists $v \in \Sigma^*$ such that no $z \in L$ has v as a factor. On the other hand, by Lemma 5, $W' = h^{\mathfrak{F}}_{u \to v}(x)$ has v as a factor but should also belong to L.

This is a contradiction, and hence our assumption that L was expressible must be wrong and L is indeed inexpressible as required. □

It is not difficult to see that it is decidable whether or not a regular language L contains two words which do not commute: it is enough to first check emptiness, and if the language is not empty, to find at least one word w in the language, compute the primitive root u of w, and check whether $L \subseteq \{u\}^*$. All of these things can be done using standard techniques for finite automata.

As a consequence, the condition given in Theorem 14 is decidable, allowing us to provide the opposite result to Theorem 10 for the dual problem of deciding whether a regular language is expressible in WE in the thin case.

Corollary 15 *Given a thin regular language L (as a regular expression or finite automaton), the property "L is expressible in* WE*" is decidable.*

This provides a partial answer to Open Problem 2. Unfortunately, we leave the remaining case of dense regular languages open in general. However, in next section we provide some further results covering specific cases.

4.2 (In)Expressibility of Dense Regular Languages

One challenge we face when considering dense languages is that while for thin regular languages L, WE-expressibility can be checked via a purely syntactic condition on (nearly) any regular expression generating L (due to Theorem 14), for dense regular languages this seems unlikely to still be the case e.g. due to the following example.

Example 3 *Let $\Sigma = \{a, b\}$. In this example it is important that we restrict the underlying alphabet of the logic* WE *to be $\{a, b\}$. If we consider* WE *with respect to a larger alphabet, e.g. $\{a, b, c\}$, then in fact the language L is thin, and Theorem 14 does apply to the regular expression e, and tells us that the language is not expressible.*

Let e be the regular expression

$$(aa \mid ab \mid aba \mid ba \mid bb \mid bba \mid baa \mid bab)^*$$

and let $L = L(e)$. Clearly L does not satisfy the conditions from Theorem 14 to be expressible in WE*. However since $e = Y^*$ such that Y contains every word over $\{a, b\}$ of length two, L is dense, so Theorem 14 does not apply in this case and we cannot use it to determine* WE*-expressibility one way or the other.*

In fact, the denseness of L allows us to represent L in an entirely different way, using a seemingly unrelated regular expression e'. To see why, note firstly that all words of even length belong to $L(e)$. Moreover, if a word $w \in \Sigma^*$ has odd length and contains ba as a factor, then $w = u_1cbau_2$ or $w = u_1bacu_2$ where $c \in \{a, b\}$ and u_1, u_2 have even length. Thus all such words also belong to $L(e)$. Finally, note that if w has odd length and does not contain ba as a factor, then $w \notin L(e)$. Thus, $L(e)$ contains all words which either contain ba as a factor, or have even length and belong to a^*b^*. Consequently, $L = L(e')$ where e' is given by

$$(a \mid b)^* \, ba \, (a \mid b)^* \mid (aa)^* \, (\varepsilon \mid ab) \, (bb)^*.$$

The subexpression $e'_1 = (aa)^* \, (\varepsilon \mid ab) \, (bb)^*$ on the right does satisfy the conditions of Theorem 14, and so $L(e'_1)$ is expressible in WE. *The subexpression $e'_2 = (a \mid b)^* \, ba \, (a \mid b)^*$ on the left is a special case: although it contains two subexpressions of the form Z^* where Z contains two non-commuting words, in both cases, we have $Z = \Sigma$, and it is easily seen that Σ^* is expressible in* WE *(e.g. by x in $x \doteq x$). It therefore follows by the fact that* WE*-expressible languages are closed under concatenation that $L(e'_2)$ is expressible in* WE*, and since* WE*-expressible languages are also closed under union, that L is also expressible in* WE.

Generalising the previous example, we can derive the following sufficient condition for a dense regular language to be expressible in WE. Due to the example above, this condition is rather a condition on the language itself, and is not seemingly related to the syntactic properties of any given regular expression generating it.

Proposition 16 *Let $L \subseteq \Sigma^*$ be a language which can be obtained as $L(e)$ for at least one well-formed regular expression e such that for every subexpression Y^* of e, either $L(Y) = \Sigma$ or the words in $L(Y)$ are pairwise commutative. Then L is expressible in* WE.

Proof Suppose the conditions of proposition are met. Then L can be obtained by applying the operations of union and concatenation successively to languages of the form:

- $\{\varepsilon\}$ and $\{a\}$ for $a \in \Sigma$
- Σ^*
- X^* where the words in X are pairwise commutative.

It is straightforward that $\{\varepsilon\}$, $\{a\}$ and Σ^* are all expressible in WE. Moreover, if the elements of X pairwise commute and $X \neq \emptyset$, there exists $\mathcal{E} \subseteq \mathbb{N}$ and $w \in \Sigma^+$ such that $X = \{w^i \mid i \in \mathcal{E}\}$. It follows by Proposition 11 that X^* is also expressible in WE. Finally, we recall that it was shown in [32] that languages expressible in WE are closed under union and concatenation. Thus, due to these closure properties, we may conclude that L is expressible in WE. □

While we are unable to extend the previous sufficient condition into a characteristic one, we shall focus for the rest of this section on examples of inexpressible dense regular languages (i.e. on necessary conditions). Dense languages also pose more of

a challenge when showing WE-inexpressibility, in part due to the inherent role in the main technique(s) of relying on swapping factors to arrive at a contradiction (Steps 4 and 5 in the general approach outlined in Section 2.2). Nevertheless, we have already in the proof of Theorem 3 shown, in a sense, how this reasoning can be modified and extended to produce other types of contradiction. In what follows we investigate some classes of dense regular languages which are not expressible in WE. By focusing on codes, we have some additional "control" in the absence of forbidden factors. We briefly recall some definitions from the theory of codes (see e.g. [9]).

Definition 17 *A code is a non-empty set Y of words which does not satisfy a non-trivial relation. In other words, every word in Y^* has a unique decomposition into words from Y. Furthermore, Y is uniform if all words in it have the same length. A code Y is bifix if for every pair y_1, y_2 of words in Y, y_1 is neither a prefix nor suffix of y_2. Two codes $Z \subseteq \Sigma_1^+$ and $Y \subseteq \Sigma_2^+$ are composable if there is a bijection $\hat{b} : \Sigma_1 \to Y$. The composition $Z \circ Y$ is given by $\{\hat{b}(z) \mid z \in Z\}$ where $\hat{b} : {\Sigma_1}^* \to Y^*$ is obtained by extending $\hat{b}$ to be a morphism. It follows that $(Z \circ Y)^* \subseteq Y^*$ and $Z \circ Y$ is also a code.*

Given our interest beyond expressibility in WE to e.g. WE + LEN and WE + LEN + REG, it makes sense to consider dense languages with properties associated with the lengths of words. For example, we might ask whether the set of even length words is expressible in WE, or more generally the set of words of length $0 \pmod n$ for some arbitrary $n \in \mathbb{N}, n > 1$. More generally, we might ask about the expressibility of sets Y^* where Y is a uniform code (setting $Y = \Sigma^n$ yields the set of words of length $0 \pmod n$). More generally still, we consider languages X^* where X is a code obtained via composition with a uniform code. In this case, we obtain the following characterisation. Note that Proposition 17 is the only time we consider an infinite alphabet. The set Z and only Z may be taken to have a countably infinite alphabet in this setting. This allows us to provide a more general class of languages X^*. An example of how an infinite alphabet may be used in the context of Proposition 17 is given in Example 4.

Proposition 17 *Let $Y \subset \Sigma^+$ be a code, and let Z be a uniform code over a finite or countably infinite alphabet, whose words each have length $p > 1$. Suppose that X may be obtained as a composition $Z \circ Y$. Then $X^* \subseteq \Sigma^*$ is expressible in* WE *if and only if at least one of $|Y| = 1$ or $|Z| = 1$ holds.*

Proof For the "if" direction, note firstly that $|X| = |Z|$, so if $|Z| = 1$ then $|X| = 1$. Moreover, by definition $X \subseteq Y^+$ so if $|Y| = 1$ then the elements of X pairwise commute. In both cases, X^* is expressible by Proposition 11.

The rest of the proof concerns the "only if" direction. We shall prove the contrapositive. To that end, let Y and Z both contain at least two elements. By the definition of a code, we know that these two elements do not commute. It follows that $|X| = |Z| > 1$ and X is also a code. Thus, there exist distinct words $w_1, w_2 \in X$ which do not commute. Let i be the smallest integer such that $|w_1^i| \geq |w_1 w_2 w_1|$ and let $\mathfrak{F} = \mathfrak{F}_{w_1^i}$. Recall from Lemma 10 that $\mathfrak{F}$ is synchronising.

Let us now suppose for contradiction that X^* is expressed by a variable x in some word equation E. Let c be the constant provided by Lemma 5. Note that since Y contains a pair of non-commuting elements, we must have $|\Sigma| > 1$. Let

$J = (3, 4, \ldots, 3 + c + |\Sigma|^{i|w_1|} + 1)$. Let $W = \prod_{j \in J} w_1^i (w_2 w_1)^j w_2 w_1^i$. Note that $W \in X^*$. By Lemma 12, since there are at most $|\Sigma|^{i|w_1|+1}$ $\mathfrak{F}$-factors of W of length at most $|w_1^i|$, there will be at least $c + 1$ $\mathfrak{F}$-factors of W having length greater than $|w_1^i|$ which contain w_i^i as a prefix and which occur only once in W. By Lemma 5, at least one of these "long" $\mathfrak{F}$-factors, u, of W is unanchored in h, where h is some solution to E such that $h(x) = W$.

Let $y \in Y$ and let v be the result of inserting an occurrence of y between two consecutive occurrences of w_1 in u. Let $W' = h^{\mathfrak{F}}_{u \to v}(x)$.

Then by Lemma 5, W' belongs to the language expressed by x in E, and thus $W' \in X^*$. Moreover, since $X^* \subseteq Y^*$, we get $W, W' \in Y^*$. Consequently, W and W' both admit unique decompositions into elements of Y. However, since W is constructed explicitly as a sequence of w_1s and w_2s, and since $w_1, w_2 \in X$, and since $X \subseteq Y^+$, it follows that the decomposition of W into elements of Y can be obtained by first decomposing W into the factors w_1 and w_2 and then further decomposing each w_1 or w_2 into elements of Y. Consequently, by construction, the decomposition of W' has precisely one more element than the decomposition of W does.

However, for every word in X^*, the number of elements from Y in the decomposition must be divisible by p. Since $p > 1$, it is impossible that both $W \in X^*$ and $W' \in X^*$. Thus we have a contradiction, and may conclude that X^* is inexpressible. □

Example 4 (See Example 3.3.9 in [9]) *Consider the code* $Y = \{wba^{|w|} : w \in \Sigma^*\}$ *over the alphabet* $\Sigma = \{a, b\}$. *Let* Δ *be an infinite alphabet, comprising a letter* ℓ_w *for each* $w \in \Sigma^*$. *Then let* $Z = \Delta^2 \backslash \{\ell_a \ell_{aa}\}$. Z *is a uniform code over the alphabet* Δ. *It comprises most, but not all words of length two, and so* Z^* *contains most, but not all words of even length. Define a bijection* $\hat{b} : \Delta \to Y$ *in the natural way: so that* $\hat{b}(\ell_w) = wba^{|w|}$ *for all* $w \in \Sigma^*$. *Using* $\hat{b}$, *we can define the composition* $X = Z \circ Y$, *given by*

$$X = \{w_1 ba^{|w_1|} w_2 ba^{|w_2|} : w_1, w_2 \in \Sigma^* \wedge (w_1, w_2) \neq (a, aa)\}.$$

X *is clearly dense, meaning that* X^* *is also dense. Consequently, the results of Section 4.1 cannot be used to determine the expressibility of* X^*. *Nevertheless, from Proposition 17, it follows that* X^* *is inexpressible in* WE.

Corollary 18 *Let* $Y \subset \Sigma^+$ *be a code, and let* $X = Y^p$ *for some* $p > 1$. *Then* X^* *is expressible in* WE *if and only if* $|Y| = 1$.

Our next proposition has a similar flavour to Proposition 17, again considering languages X^* where the lengths of words are restricted, but with a less technical condition. The reasoning is very similar but note that the classes of languages addressed are incomparable. In particular, we are able to drop the condition that the underlying set X is a code.

Proposition 19 *Let* $X \subset \Sigma^+$ *satisfy* $\gcd(\{|w| : w \in X\}) = p > 1$. *Then* X^* *is expressible in* WE *if and only if the elements of* X *are pairwise commutative.*

Proof The "if" direction is covered by Proposition 11. The rest of the proof focuses on the "only if" direction. Specifically, we prove the contrapositive. To that end, let

X contain non-commuting words w_1, w_2, and let us assume for contradiction that the resulting language X^* is expressed by a variable x in some word equation E. Let i be the smallest integer such that $|w_1^i| \geq |w_1 w_2 w_1|$ and let $\mathfrak{F} = \mathfrak{F}_{w_1^i}$. Recall from Lemma 10 that $\mathfrak{F}$ is synchronising. Let c be the constant provided by Lemma 5. Let $J = (3, 4, \ldots, 3 + c + |\Sigma|^{i|w_1|+1})$. Let $W = \prod_{j \in J} w_1^i (w_2 w_1)^j w_2 w_1^i$. Note that $W \in X^*$. By Lemma 12, since there are at most $|\Sigma|^{i|w_1|+1}$ $\mathfrak{F}$-factors of W of length at most $|w_1^i|$, there will be at least $c + 1$ $\mathfrak{F}$-factors of W having length greater than $|w_1^i|$ and which occur only once in W. By Lemma 5, at least one of these "long" $\mathfrak{F}$-factors, u, of W is unanchored in h, where h is some solution to E such that $h(x) = W$. Let $W' = h^{\mathfrak{F}}_{u \to au}(x)$. Then by Lemma 5, W' belongs to the language expressed by x in E, and thus $W' \in X^*$. However, this implies that both $|W|$ and $|W'|$ should be divisible by 1. By construction, $|W'| = |W| + 1$. Since $p > 1$, this is a contradiction, and hence we can conclude that X^* is inexpressible.

In Proposition 19, we did not require the set X to be a code. It would be nice to remove the analogous requirement for Y to be a code from Corollary 18. However, it is worth pointing out that this cannot be done without further modification to the statement. Indeed, setting $\Sigma = \{a, b\}$, $Y = \{a, b, ba\}$ and $X = Y^2$, we obtain the language $X^* = \{aa, ab, aba, ba, bb, bba, baa, bab, baba\}^*$ considered in Example 3. Then the elements of $Y \subset \Sigma^+$ are not pairwise commutative. However, we have already shown in Example 3 that X^* is indeed expressible in WE.

Our final result in this section provides a third class of potentially dense languages X^* for which we can characterise expressibility in WE.

Theorem 20 *Let $X \subset \Sigma^+$ contain two distinct words w, each of which satisfies the condition:*

$$\forall x \in X \backslash \{w\}, \{w, x\} \text{ is a bifix code.}$$

Then X^* is expressible in WE *if and only if $\Sigma \subseteq X$.*

Proof Suppose first that $\Sigma \subseteq X$. Then clearly $X^* = \Sigma^*$, so X^* is expressible. Suppose instead that there is some $a \in \Sigma \backslash X$. Let w_1, w_2 be two distinct words satisfying the criterion from the lemma. Let us assume for contradiction that the resulting language X^* is expressible. Note that w_1, w_2 cannot commute. If they did, then they would be two powers of the same primitive root, and $\{w_1, w_2\}$ would not be a bifix code.

Let us assume for contradiction that the resulting language X^* is expressed by a variable x in some word equation E. Let i be the smallest integer such that $|w_1^i| \geq |w_1 w_2 w_1|$ and let $\mathfrak{F} = \mathfrak{F}_{w_1^i}$. Recall from Lemma 10 that $\mathfrak{F}$ is synchronising. Let c be the constant provided by Lemma 5. Let $J = (3, 4, \ldots, 3 + c + |\Sigma|^{i|w_1|+1})$. Let $W = \prod_{j \in J} w_1^i (w_2 w_1)^j w_2 w_1^i$. Note that $W \in X^*$. By Lemma 12, since there are at most $|\Sigma|^{i|w_1|+1}$ $\mathfrak{F}$-factors of W of length at most $|w_1^i|$, there will be at least $c + 1$ $\mathfrak{F}$-factors of W having length greater than $|w_1^i|$ which contain w_i^i as a prefix and which occur only once in W. By Lemma 5, at least one of these "long" $\mathfrak{F}$-factors, u, of W is unanchored in h, where h is some solution to E such that $h(x) = W$.

Let W' be obtained from W by inserting a into the single occurrence of u in W, between two factors w_1. That is, let $v = u_1 w_1 a w_1 u_2$ where $u = u_1 w_1 w_1 u_2$ and

let $W' = h^{\mathfrak{F}}_{u\to v}(x)$. Then by Lemma 5, $W' \in X^*$. Moreover, it follows from the conditions of the theorem that

- no proper prefix of w_1 or w_2 is in X,
- no proper suffix of w_1 or w_2 is in X,
- no words in X have w_1 or w_2 as a proper prefix,
- no words in X have w_1 or w_2 as a proper suffix.

Thus, we can "strip" the factors w_1 and w_2 from each end of W', whilst retaining the resulting word's membership of X^*. The result of iterating this process is that $a \in X^*$. However this implies $a \in X$, which is a contradiction. Hence we can conclude that X^* is inexpressible. □

We conclude this section, and the main technical content of the paper with the following immediate consequence of Theorem 20.

Corollary 21 *Let $X \subset \Sigma^+$ be bifix. Then X^* is expressible in* WE *if and only if $X = \Sigma$ or $|X| \leq 1$.*

Acknowledgements We would like to express our gratitude to the anonymous referees of this paper for their careful reviews many helpful comments and suggestions which have undoubtedly improved the final version. In particular, we thank them for the suggestions simplifying the technical analysis in Lemma 12 and as a result several proofs in Sections 4.1 and 4.2. The work of Florin Manea was supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG), by the project with number 466789228.

Author Contributions All authors contributed to the main content/ideas. J.D. prepared the final manuscript, with Section 4 adapted from an earlier draft prepared by M.K and other sections based on prior versions to which all authors contributed. All authors reviewed the manuscript.

Declarations

Competing Interests The authors declare no competing interests.

References

1. Abdulla, P.A., Atig, M.F., Chen, Y.F., Diep, B.P., Dolby, J., Janku, P., Lin, H.H., Holík, L., Wu, W.-C., : Efficient handling of string-number conversion. In: Donaldson A.F., Torlak, E. (eds.) Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020. pp. 943–957. ACM, (2020)
2. Abdulla, P.A., Atig, M.F., Chen, Y.-F., Holík, L., Rezine, A., Rümmer, P., Stenman, J. : Norn: An SMT solver for string constraints. In: Kroening, D., Pasareanu, C.S. (eds.) Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I, volume 9206 of Lecture Notes in Computer Science. pp. 462–469. Springer, (2015)

3. Alur, R., Kumar, V., Madhusudan, P., Viswanathan, M. : Congruences for visibly pushdown languages. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings, volume 3580 of Lecture Notes in Computer Science. pp. 1102–1114. Springer, (2005)
4. Alur, R., Madhusudan, P. : Visibly pushdown languages. In: Babai, L. (ed.) Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004. pp. 202–211. ACM, (2004)
5. Alur, R., Madhusudan, P. : Adding nesting structure to words. J. ACM, 56(3):16:1–16:43, 2009
6. R. Amadini: A survey on string constraint solving. ACM Comput. Surv., 55(1), 2021
7. P. Barceló, P. Muñoz : Graph logics with rational relations: The role of word combinatorics. ACM Trans. Comput. Log., 18(2):10:1–10:41, 2017
8. Barrett, C.W. , Conway, C.L., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C. : CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings, volume 6806 of Lecture Notes in Computer Science. p. 171–177. Springer, (2011)
9. Berstel, J., Perrin, D., Reutenauer, C. : Codes and automata, volume 129. Cambridge University Press, (2010)
10. Berzish, M., Kulczynski, M., Mora, F., Manea, F., Day, J.D., Nowotka, D., Ganesh, V. : An SMT solver for regular expressions and linear arithmetic over string length. In: Silva, A., Rustan, K., Leino, M. (eds) Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II, volume 12760 of Lecture Notes in Computer Science. pp. 289–312. Springer, (2021)
11. Büchi, J.R., Senger, S.: Definability in the existential theory of concatenation and undecidable extensions of this theory. Math. Log. Q. **34**(4), 337–342 (1988)
12. Chen, T., Chen, Y., Hague, M., Lin, A.W., Wu, Z. : What is decidable about string constraints with the replaceall function. Proc. ACM Program. Lang., 2(POPL):3:1–3:29, (2018)
13. Chen, T., Flores-Lamas, A., Hague, M., Han, Z., Hu, D., Kan, S., Lin, A.W., Rümmer, P., Wu, Z. : Solving string constraints with regex-dependent functions through transducers with priorities and variables. Proc. ACM Program. Lang., 6(POPL):1–31, (2022)
14. Chen, T. , Hague, M., Lin, A.W., Rümmer, P., Wu, Z. : Decision procedures for path feasibility of string-manipulating programs with complex operations. Proc. ACM Program. Lang., 3(POPL):49:1–49:30, (2019)
15. Day, J.D. , Ganesh, V., Grewal, N., Manea, F. : On the expressive power of string constraints. In: Proc. ACM Program. Lang. ACM, (2023)
16. Day, J.D., Ganesh, V., He, P., Manea, F., Nowotka, D. : The satisfiability of word equations: Decidable and undecidable theories. In: Potapov, I., Reynier, P.-A. (eds) Reachability Problems - 12th International Conference, RP 2018, Marseille, France, September 24-26, 2018, Proceedings, volume 11123 of Lecture Notes in Computer Science. pp. 15–29. Springer, (2018)
17. Diekert, V., Gutiérrez, C., Hagenah, C.: The existential theory of equations with rational constraints in free groups is pspace-complete. Inf. Comput. **202**(2), 105–140 (2005)
18. Durnev, V.G.: On equations in free semigroups and groups. Matematicheskie Zametki **16**, 717–724 (1974). (**In Russian; English translation: Math. Notes of the Acad. of Sci. of the USSR 16 (1975) 1024–1028**)
19. Durnev, V.G.: 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**)
20. Durnev, V.G. : Studying algorithmic problems for free semi-groups and groups. In: Adian, S., Nerode, A. (eds) Proceedings of the 4th International Symposium on Logical Foundations of Computer Science (LFCS'97), Yaroslavl, Russia, July 6–12, 1997, volume 1234. pp. 88–101, (1997)
21. Figueira, D., Jeż, A., Lin, A.W. : Data path queries over embedded graph databases. In: Libkin, L., Barceló, P. (eds) PODS '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022. pp. 189–201. ACM, (2022)
22. Freydenberger, D.D.: A logic for document spanners. Theory Comput. Syst. **63**(7), 1679–1754 (2019)
23. Freydenberger, D.D., Holldack, M.: Document spanners: From expressive power to decision problems. Theory Comput. Syst. **62**(4), 854–898 (2018)
24. Freydenberger, D.D., Peterfreund, L. : The theory of concatenation over finite models. In: Bansal, N., Merelli, E., Worrell, J. (eds) 48th International Colloquium on Automata, Languages, and Program-

ming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference), volume 198 of LIPIcs. pp. 130:1–130:17. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, (2021)
25. Ganesh, V., Minnes, M., Solar-Lezama, A., Rinard, M.C. : Word equations with length constraints: What's decidable? In: Biere, A., Nahir, A., Vos, T.E.J. (eds) Hardware and Software: Verification and Testing - 8th International Haifa Verification Conference, HVC 2012, Haifa, Israel, November 6-8, 2012. Revised Selected Papers, volume 7857 of Lecture Notes in Computer Science. pp. 209–226. Springer, (2012)
26. Hague, M.: Strings at MOSCA. ACM SIGLOG News **6**(4), 4–22 (2019)
27. Halfon, S., Schnoebelen, P., Zetzsche, G. : Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In: 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017. pp. 1–12. IEEE Computer Society, (2017)
28. Holík, L., Janku, P., Lin, A.W., Rümmer, P., Vojnar, T.: String constraints with concatenation and transducers solved efficiently. Proc. ACM Program. Lang. **2**(POPL), 4:1-4:32 (2018)
29. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory. Addison-Wesley, Languages and Computation (1979)
30. Jeż, A.: Word equations in non-deterministic linear space. J. Comput. Syst. Sci. **123**, 122–142 (2022)
31. Kan, S., Lin, A.W., Rümmer, P., Schrader, M.: Certistr: a certified string solver. In: Popescu, A.,Zdancewic, S. (eds) CPP '22: 11th ACM SIGPLAN International Conference on Certified Programs and Proofs, Philadelphia, PA, USA, January 17 - 18, 2022. pp. 210–224. ACM, (2022)
32. Karhumäki, J., Mignosi, F., Plandowski, W.: The expressibility of languages and relations by word equations. J. ACM **47**(3), 483–505 (2000)
33. Kiezun, A., Ganesh, V., Guo, P.J., Hooimeijer, P., Ernst, M.D. : HAMPI: a solver for string constraints. In: Rothermel, G., Dillon, L.K. (eds) Proceedings of the Eighteenth International Symposium on Software Testing and Analysis, ISSTA 2009, Chicago, IL, USA, July 19-23, 2009. pp. 105–116. ACM, (2009)
34. Le, Q.L., He, M. : A decision procedure for string logic with quadratic equations, regular expressions and length constraints. In: Ryu, S. (ed) Programming Languages and Systems-16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings, volume 11275 of Lecture Notes in Computer Science. pp. 350–372. Springer, (2018)
35. Liang, T., Tsiskaridze, N., Reynolds, A., Tinelli, C., Barrett, C.W. : A decision procedure for regular membership and length constraints over unbounded strings. In: Lutz, C., Ranise, S. (eds) Frontiers of Combining Systems-10th International Symposium, FroCoS 2015, Wroclaw, Poland, September 21-24, 2015. Proceedings, volume 9322 of Lecture Notes in Computer Science. pp. 135–150. Springer, (2015)
36. Lin A.W., Majumdar, R. : Quadratic word equations with length constraints, counter systems, and presburger arithmetic with divisibility. In: Lahiri, S.K., Wang, C. (eds) Automated Technology for Verification and Analysis-16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings, volume 11138 of Lecture Notes in Computer Science. pp. 352–369. Springer, (2018)
37. Lin, A.W., Barceló, P. : String solving with word equations and transducers: towards a logic for analysing mutation XSS. In: Bodík, R., Majumdar, R. (eds) Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20-22, 2016. pp. 123–136. ACM, (2016)
38. Lothaire, M.: Combinatorics on words, 2nd edn. Cambridge University Press, Cambridge Mathematical Library (1997)
39. Lothaire, M.: Algebraic combinatorics on words. Cambridge University Press (2002)
40. Makanin, G.S.: The problem of solvability of equations in a free semigroup. Mathematics of the USSR-Sbornik **32**(2), 129 (1977)
41. Mora, F., Berzish, M., Kulczynski, M., Nowotka, D., Ganesh, V. : Z3str4: A multi-armed string solver. In: Huisman, M., Pasareanu, C.S., Zhan, N. (eds.) Formal Methods-24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings, volume 13047 of Lecture Notes in Computer Science. pp. 389–406. Springer, (2021)
42. Plandowski, W. : Satisfiability of word equations with constants is in PSPACE. In: 40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA. pp. 495–500. IEEE Computer Society, (1999)
43. Quine, W.V.: Concatenation as a basis for arithmetic. The Journal of Symbolic Logic **11**(4), 105–114 (1946)

44. Schulz, K.U. : Makanin's algorithm for word equations - two improvements and a generalization. In: Schulz, K.U. (ed.) Word Equations and Related Topics, First International Workshop, IWWERT '90, Tübingen, Germany, October 1-3, 1990, Proceedings, volume 572 of Lecture Notes in Computer Science. pp. 85–150. Springer, (1990)
45. Shur, A.M. : Combinatorial complexity of regular languages. International Computer Science Symposium in Russia. pp. 289–301. Springer (2008)
46. Trinh, M.-T., Chu, D.-H., Jaffar, J. : Progressive reasoning over recursively-defined strings. In: Chaudhuri, S., Farzan, A. (eds) Computer Aided Verification-28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I, volume 9779 of Lecture Notes in Computer Science. pp. 218–240. Springer, (2016)

Authors and Affiliations

Joel Day[1] · Vijay Ganesh[2] · Nathan Grewal[2] · Matthew Konefal[1] · Florin Manea[3]

Vijay Ganesh
vijay.ganesh@uwaterloo.ca

Nathan Grewal
negrewal@uwaterloo.ca

Matthew Konefal
M.Konefal-20@student.lboro.ac.uk

Florin Manea
florin.manea@informatik.uni-goettingen.de

1 Loughborough University, Loughborough, UK

2 University of Waterloo, Waterloo, ON N2L, Canada

3 Universität Göttingen, Göttingen, Germany