Theory of Computing Systems
https://doi.org/10.1007/s00224-024-10198-4

Languages Generated by Conjunctive Query Fragments of FC[REG]

Sam M. Thompson[1] · **Dominik D. Freydenberger**[1]

Accepted: 19 September 2024

Abstract
FC is a finite model variant on the theory of concatenation, FC[REG] extends FC with regular constraints. This paper considers the languages generated by their conjunctive query fragments, FC-CQ and FC[REG]-CQ. We compare the expressive power of FC[REG]-CQ to that of various related language generators, such as regular expressions, patterns, and typed patterns. We then consider decision problems for FC-CQ and FC[REG]-CQ, and show that certain static analysis problems (such as equivalence and regularity) are undecidable. While this paper defines FC-CQ based on the logic FC, it can equally be understood as synchronized intersections of pattern languages, or as systems of restricted word equations.

Keywords Word equations · Conjunctive queries · Expressive power · Decision problems · Descriptional complexity

1 Introduction

This paper studies the languages that are generated by fragments of FC, a variant of the theory of concatenation, that has applications in information extraction.

Word Equations and the Theory of Concatenation We begin with *word equations*, which have the form $\alpha_L \doteq \alpha_R$, where both α_L and α_R are words over an alphabet Σ of terminal symbols and an alphabet Ξ of variables. Each variable represents some word from Σ^*. These equations can be used to define languages or relations (see Karhumäki et al. [1]), by choosing one or more variables. For example, consider the equation $(xy \doteq \mathtt{ab})$, where x and y are variables. Then, the variable x defines the language $\{\varepsilon, \mathtt{a}, \mathtt{ab}\}$, and the pair (x, y) defines the relation $\{(\varepsilon, \mathtt{ab}), (\mathtt{a}, \mathtt{b}), (\mathtt{ab}, \varepsilon)\}$.

✉ Sam M. Thompson
s.thompson4@lboro.ac.uk

Dominik D. Freydenberger
D.D.Freydenberger@lboro.ac.uk

1 Loughborough University, Loughborough, UK

The *theory of concatenation* uses word equations as *atomic formulas*, and combines them with the usual logical connectives and first-order quantifiers that range over words in Σ^*. For example, the variable x in the formula $\neg\exists y\colon (x \doteq yy)$ defines the language of all words that are not a square.

Usually, only its existential-positive fragment is studied, as *satisfiability* is decidable (see e.g. [2]); and even venturing only a little beyond that quickly leads to undecidability (see e.g. Durnev [3]). Also relevant for the current paper is an extension that allows the use of *regular constraints*, that is, atoms of the form $(x \mathrel{\dot{\in}} \alpha)$ for any regular expression α, which express that $x \in \mathcal{L}(\alpha)$.

The Logic FC Freydenberger and Peterfreund [4] introduced the logic FC as a finite model version of the theory of concatenation. FC fixes one word w as the *input word* (the word for which membership in the language is tested, or the word from which information is extracted), and considers factors of w for the variables. This is in contrast to the "standard" theory of concatenation, where variables range over the infinite set Σ^*.

In FC formulas, the input word w is represented by w. For example, the FC-formula $\exists x\colon (\mathsf{w} \doteq xx)$ accepts those words w that are squares, and

$$\forall x\colon \big(\neg(x \doteq \varepsilon) \rightarrow \neg\exists y\colon (x \doteq yy)\big)$$

accepts those words that do not contain a non-empty square.[1]

The logic FC[REG] extends FC by allowing regular constraints. For example,

$$\exists x, y\colon \big((\mathsf{w} \doteq xyxy) \land (x \mathrel{\dot{\in}} \mathtt{a}^+) \land (y \mathrel{\dot{\in}} \mathtt{b}^+)\big)$$

defines the language $\{\mathtt{a}^m\mathtt{b}^n\mathtt{a}^m\mathtt{b}^n \mid m, n \geq 1\}$.

FC[REG] is motivated by information extraction. Fagin et al. [5] introduced *core spanners* to formalize the core of the query language AQL that is used in IBM's SystemT. Core spanners use relational algebra on top of regular expressions with variables; this allows one to query text like one queries a relational database. But for various reasons, restrictions from database theory and algorithmic finite model theory that make relational queries tractable do not translate to spanners (for example, see [6]). In contrast to this, the canonical tractability restrictions from relational first-order logic can be adapted to FC[REG]. Moreover, its definition could be considered much simpler than that of core spanners (see [4]), and existential-positive FC[REG] has the same expressive power as core spanners (with polynomial-time transformations in both directions); see Theorem 5.9 of [4].

Conjunctive Queries One of the aforementioned tractability restrictions is based on *conjunctive queries (CQs)*, a model that has been studied extensively in database theory (see Abiteboul et al. [7] as a starting point). From the logic point of view, a CQ is a series of existential quantifiers applied to a conjunction of atoms.[2] While evaluating

[1] While w does not appear in this formula, all variables in an FC-formula range over the factors of the input word.

[2] In terms of SQL, it is `SELECT` applied to a `JOIN` of (potentially many) tables.

these is still NP-hard in general, there is a tractable subclass, the so-called *acyclic* CQs. Moreover, determining whether a CQ is acyclic can be decided in polynomial time.

Freydenberger and Thompson [8] adapted this to FC and FC[REG] by introducing FC-CQs and FC[REG]-CQs. These are defined analogously to CQs, and they have the same desirable algorithmic properties. Furthermore, they allow for additional optimization, as certain queries can be split into acyclic FC-CQs. As a step towards answering the question what can (and cannot) be expressed with acyclic FC-CQs and FC[REG]-CQs, the present paper looks into the properties of FC-CQ and FC[REG]-CQ in general.

Two reasons led the authors to believe that approaching this based on languages is a natural choice. First, even though database theory usually sees inexpressibility more as the question which relations cannot be expressed, all relation inexpressibility results for fragments of FC[REG] and equivalent models (like spanners) rely on language inexpressibility (see [4, 5, 9–11]). Second, as we shall see, FC-CQs (and FC[REG]-CQs) are conceptually very close to (typed) pattern languages and system of word equations (with regular constraints). Hence, they define rich and natural classes of languages, which makes them interesting beyond their background and application in logic.

Contributions In Section 3, we first consider the closure properties of FC-CQ and FC[REG]-CQ, and show that FC-CQ and FC[REG]-CQ are both closed under intersection, but are not closed under complement. Regarding the closure under union, we show that FC-CQ is not closed under union, but whether FC[REG]-CQ is closed under union is left open. We then compare the expressive power of FC-CQ and FC[REG]-CQ to that of related language generators, such as patterns and regularly typed patterns. The results regarding expressive power are summarized in Fig. 1. To conclude this section, we show that a large and natural class of xregex can be represented by FC[REG]-CQ.

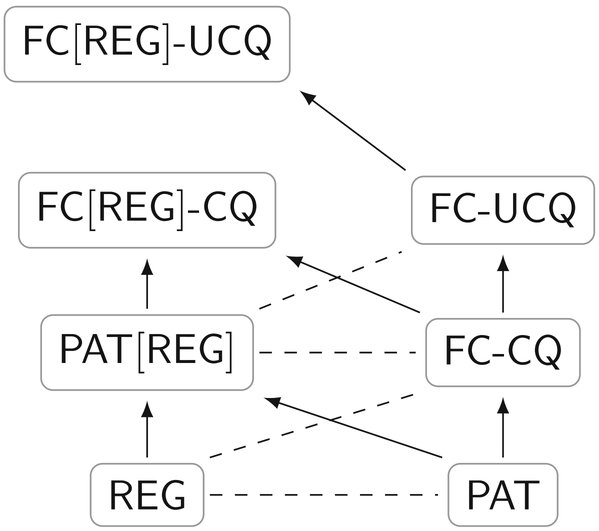

Fig. 1 A directed edge from A to B denotes that $\mathcal{L}(\mathsf{A}) \subset \mathcal{L}(\mathsf{B})$. A dashed, undirected edge between A and B denotes that $\mathcal{L}(\mathsf{A})$ and $\mathcal{L}(\mathsf{B})$ are incomparable

Section 4 considers the complexity bounds and decidability of various decision problems. The main result of this section is that the universality problem for FC[REG]-CQ is undecidable, and the regularity problem for FC-CQ is undecidable. In Section 5, we use Hartmanis' meta theorem to show non-recursive trade-offs from FC-CQs to regular expressions.

2 Preliminaries

For $n \geq 1$, let $[n] := \{1, 2, \ldots, n\}$. Let $\emptyset$ denote the *empty set*. We use $|S|$ for the *cardinality* of S. If S is a subset of T then we write $S \subseteq T$ and if $S \neq T$ also holds, then $S \subset T$. The difference of two sets S and T is denoted as $S \setminus T$. We use $\mathbb{N}$ to denote the set $\{0, 1, \ldots\}$ and $\mathbb{N}_+ := \mathbb{N} \setminus \{0\}$.

Let A be an alphabet. We use $|w|$ to denote the length of some word $w \in A^*$, and the word of length zero (the *empty word*) is denoted ε. The number of occurrences of some $\mathtt{a} \in A$ within w is $|w|_{\mathtt{a}}$. We write $u \cdot v$ or just uv for the concatenation of words $u, v \in A^*$. If we have words $w_1, w_2, \ldots, w_n \in A^*$, then we use $\Pi_{i=1}^{n} w_i$ as shorthand for $w_1 \cdot w_2 \cdots w_n$. If $u = p \cdot v \cdot s$ for $p, s \in A^*$, then v is a *factor* of u, denoted $v \sqsubseteq u$. If $u \neq v$ also holds, then $v \sqsubset u$. Let Σ be a fixed and finite alphabet of *terminal symbols* and let Ξ be a countably infinite alphabet of *variables*. We assume that $\Sigma \cap \Xi = \emptyset$ and $|\Sigma| \geq 2$. A *language* $L \subseteq \Sigma^*$ is a set of words. If A and B are alphabets, then a function $\sigma\colon A^* \to B^*$ is a *morphism* if $\sigma(w \cdot u) = \sigma(w) \cdot \sigma(u)$ holds for all $w, u \in A^*$.

2.1 Patterns

A *pattern* is a word $\alpha \in (\Sigma \cup \Xi)^*$. A *pattern substitution* (or just *substitution*) is a partial morphism $\sigma\colon (\Sigma \cup \Xi)^* \to \Sigma^*$ where $\sigma(\mathtt{a}) = \mathtt{a}$ must hold for all $\mathtt{a} \in \Sigma$. Let $\mathsf{vars}(\alpha)$ be the set of variables in α. We always assume that the domain of σ is a superset of $\mathsf{vars}(\alpha)$. The language α generates is defined as: $\mathcal{L}(\alpha) := \{\sigma(\alpha) \mid \sigma \text{ is a substitution}\}$. Note that some literature, such as Angluin [12], defines the language of a pattern based on *non-erasing substitutions*, whereas we always assume variables can be mapped to the empty word.

Example 1 Let $\Sigma := \{\mathtt{a}, \mathtt{b}\}$. Consider the pattern $\alpha := \mathtt{ab}x\mathtt{ba}xyx$ and the pattern substitution $\sigma\colon (\Sigma \cup \Xi)^* \to \Sigma^*$, where $\sigma(x) = \mathtt{aa}$ and $\sigma(y) = \varepsilon$, then it follows that $\mathtt{abaabaaaaa} \in \mathcal{L}(\alpha)$. By definition $\sigma(\mathtt{a}) = \mathtt{a}$ and $\sigma(\mathtt{b}) = \mathtt{b}$.

Let $\mathsf{PAT} := \{\alpha \mid \alpha \in (\Sigma \cup \Xi)^*\}$ be the set of all patterns and let $\mathcal{L}(\mathsf{PAT})$ be the class of languages definable by a pattern. Note that we always assume some fixed Σ before discussing patterns and the languages they generate.

Typed Patterns
Typed patterns extend patterns with a *typing function*. For this paper, we consider regularly typed patterns where the typing function T maps each variable x to a regular language $T(x)$. Then, each variable x must be mapped to a member of $T(x)$. More formally:

Definition 1 A *regularly typed pattern* is a pair $\alpha_T := (\alpha, T)$ consisting of a pattern $\alpha \in (\Sigma \cup \Xi)^*$, and a *typing function* T that maps each $x \in \mathsf{vars}(\alpha)$ to a regular language $T(x) \subseteq \Sigma^*$.

We denote the language α_T generates as $\mathcal{L}(\alpha_T)$ and this language is defined as $\{\sigma(\alpha) \mid \sigma$ is a substitution where $\sigma(x) \in T(x)$ for all $x \in \mathsf{vars}(\alpha)\}$.

The set of regular typed patterns is denoted by $\mathsf{PAT[REG]}$, and the class of languages definable by a regularly typed pattern is denoted by $\mathcal{L}(\mathsf{PAT[REG]})$.

Example 2 Consider $(x\mathtt{b}x, T)$, where $T(x) = \mathtt{a}^+$. Then, for any alphabet Σ where $\mathtt{a}, \mathtt{b} \in \Sigma$, we have that $\mathcal{L}(x\mathtt{b}x, T) = \{\mathtt{a}^n\mathtt{b}\mathtt{a}^n \mid n \in \mathbb{N}_+\}$, which is neither a regular language nor a pattern language.

Typed pattern languages have been considered in the context of *learning theory*, for example, see Geilke and Zilles [13] or Koshiba [14]. Schmid [15] compared the expressive power of regularly typed patterns to REGEX and *pattern expressions*.

2.2 FC Conjunctive Queries

A *word equation* is an equation $(\alpha_L \doteq \alpha_R)$, where $\alpha_L, \alpha_R \in (\Sigma \cup \Xi)^*$. For a substitution $\sigma\colon (\Sigma \cup \Xi)^* \to \Sigma^*$, we write $\sigma \models (\alpha_L \doteq \alpha_R)$, if $\sigma(\alpha_L) = \sigma(\alpha_R)$. Let $\mathsf{w} \in \Xi$ be a distinguished variable known as the *universe variable*. For a substitution $\sigma\colon (\Sigma \cup \Xi)^* \to \Sigma^*$, we say that σ is w-*safe* if $\sigma(x) \sqsubseteq \sigma(\mathsf{w})$ for all $x \in \Xi$. Now, we define the syntax and languages of FC-CQ.

Definition 2 An FC-CQ is denoted as $\varphi := \bigwedge_{i=1}^{n}(x_i \doteq \alpha_i)$ where $x_i \in \Xi$, and where $\alpha_i \in (\Sigma \cup \Xi)^*$ for each $i \in [n]$. For a w-safe substitution σ, we write $\sigma \models \varphi$ if $\sigma \models (x_i \doteq \alpha_i)$ for all $i \in [n]$. Then, $\mathcal{L}(\varphi) := \{\sigma(\mathsf{w}) \mid \sigma \models \varphi\}$.

We use $\mathcal{L}(\mathsf{FC\text{-}CQ})$, to define the class of languages definable in FC-CQ.

Example 3 In Example 2, we defined a regular typed pattern language that defines the language $\{\mathtt{a}^n\mathtt{b}\mathtt{a}^n \mid n \in \mathbb{N}_+\}$. Now consider:

$$\varphi := (\mathsf{w} \doteq x \cdot \mathtt{b} \cdot x) \wedge (x \doteq \mathtt{a} \cdot y) \wedge (x \doteq y \cdot \mathtt{a}).$$

If $vu = uv$ for $u, v \in \Sigma^*$, then there is some $z \in \Sigma^*$ and $k_1, k_2 \in \mathbb{N}$ such that $u = z^{k_1}$ and $v = z^{k_2}$ (for example, see Proposition 1.3.2 in Lothaire [16]). Thus, if $\sigma \models \varphi$, then $\sigma(\mathsf{w}) = \mathtt{a}^n \cdot \mathtt{b} \cdot \mathtt{a}^n$. Consequently, $\mathcal{L}(\varphi) = \{\mathtt{a}^n\mathtt{b}\mathtt{a}^n \mid n \in \mathbb{N}_+\}$.

In [4], FC was extended to FC[REG] by adding regular constraints. We extend FC-CQ to FC[REG]-CQ in the same way.

Definition 3 Let FC[REG]-CQ be the set of FC-CQ-formulas extended with regular constraints. That is, formulas of the form $\varphi := \psi \wedge \bigwedge_{j=1}^{m}(y_i \mathrel{\dot{\in}} \gamma_i)$ where ψ is an FC-CQ-formula, $y_j \in \Xi$ is a variable for all $j \in [m]$, and γ_j is a regular expression for all $j \in [m]$.

For a w-safe substitution σ, we write $\sigma \models \varphi$ if $\sigma \models \psi$, and $\sigma(y_j) \in \mathcal{L}(\gamma_j)$ for all $j \in [m]$. Let $\mathcal{L}(\varphi) := \{\sigma(\mathsf{w}) \mid \sigma \models \varphi\}$.

Also, we extend FC[REG]-CQ to FC[REG]-UCQ (unions of FC[REG]-CQ formulas) canonically. More formally, if $\varphi_i \in$ FC[REG]-CQ for all $i \in [m]$, then $\varphi := \bigvee_{i=1}^{m} \varphi_i$ is an FC[REG]-UCQ. We define $\mathcal{L}(\varphi) := \bigcup_{i=1}^{m} \mathcal{L}(\varphi_i)$. The class FC-UCQ is defined analogously.

Example 4 The language $L_< := \{\mathtt{a}^n\mathtt{b}\mathtt{a}^m \mid n, m \in \mathbb{N} \text{ and } n < m\}$ can be expressed in FC[REG]-CQ using the following formula:

$$\varphi_< := (\mathsf{w} \doteq x \cdot \mathtt{b} \cdot x \cdot y) \land (x \mathrel{\dot\in} \mathtt{a}^*) \land (y \mathrel{\dot\in} \mathtt{a}^+).$$

We can represent the language $L_{\neq} := \{\mathtt{a}^n\mathtt{b}\mathtt{a}^m \mid n, m \in \mathbb{N} \text{ and } n \neq m\}$ in FC[REG]-UCQ using a union of two queries analogous to $\varphi_<$. However, as we observe with the following formula, $L_{\neq}$ can also be expressed in FC[REG]-CQ:

$$\psi_{\neq} := (\mathsf{w} \doteq x \cdot y_{\mathtt{b},1} \cdot z \cdot y_{\mathtt{b},2} \cdot x) \land (\mathsf{w} \mathrel{\dot\in} \mathtt{a}^*\mathtt{b}\mathtt{a}^*) \land \\ (z \mathrel{\dot\in} \mathtt{a}^+) \land (x \mathrel{\dot\in} \mathtt{a}^*) \land (y_{\mathtt{b}} \doteq \mathtt{b}) \land (y_{\mathtt{b}} \doteq y_{\mathtt{b},1} \cdot y_{\mathtt{b},2}).$$

The use of $y_{\mathtt{b}}$ ensures that $y_{\mathtt{b},1} = \mathtt{b}$ and $y_{\mathtt{b},2} = \varepsilon$, or vice versa. Thus, the word $\sigma(z) \in \mathtt{a}^+$, for any substitution σ where $\sigma \models \psi_{\neq}$, appears one side of the $\mathtt{b}$ symbol. This ensures that $\sigma(\mathsf{w}) = \mathtt{a}^m\mathtt{b}\mathtt{a}^n$ and $m \neq n$.

From time to time, we shall make reference to the *existential positive fragment of* FC *and* FC[REG], denoted EP-FC and EP-FC[REG] respectively. This is simply an existential positive first-order logic, where each atom is of the form $x \doteq \alpha$, for some $x \in \Xi$ and $\alpha \in (\Sigma \cup \Xi)^*$. For details, see [4]. While we do not give the formal definitions of EP-FC and EP-FC[REG] in this paper, we note that $\mathcal{L}$(EP-FC) $=$ $\mathcal{L}$(FC-UCQ) and $\mathcal{L}$(EP-FC[REG]) $=$ $\mathcal{L}$(FC[REG]-UCQ). This follows from the fact that FC-UCQ (or FC[REG]-UCQ) is simply a disjunctive normal form of EP-FC (or EP-FC[REG]).

Immediately from the definitions, we know that $\mathcal{L}$(FC-CQ) $\subseteq$ $\mathcal{L}$(EP-FC), and $\mathcal{L}$(FC[REG]-CQ) $\subseteq$ $\mathcal{L}$(EP-FC[REG]). In Section 3, we shall look at the expressive power of FC-CQ and FC[REG]-CQ in more detail.

The Theory of Concatenation Before moving on, let us briefly discuss the theory of concatenation; in particular, the *existential positive fragment of the theory of concatenation*, which we shall denote as EP-C. Informally, EP-C can be seen as a logic that extends word equations with conjunction, disjunction, and existential quantification. While the syntax is similar to EP-FC, the semantics of EP-C allow variables to range over Σ^*. Thus, unlike with EP-FC, there is no separation between the membership problem (does a formula hold for a given word), and the satisfiability problem (does a formula hold for any word).

Regarding the class of languages $\mathcal{L}$(EP-C) generated by EP-C, it is known that this is exactly the class of languages generated by word equations. For more information, see Section 5.3 in Chapter 6 of [17]. We use EP-C[REG] to denote the existential theory of concatenation extended with regular constraints. The authors refer the reader to [4] for more information on the theory of concatenation, and its connection to FC.

3 Expressive Power and Closure Properties

This section considers the expressive power and closure properties of FC-CQ and FC[REG]-CQ. The main result is Theorem 2, which presents an inclusion diagram of the relative expressive power of FC[REG]-CQ and related models.

Lemma 1 *For $\varphi \in$ FC-CQ, we have that $\mathcal{L}(\varphi) = \Sigma^*$ if and only if $\varepsilon, \mathtt{a} \in \mathcal{L}(\varphi)$ for some $\mathtt{a} \in \Sigma$.*

Proof The only-if direction follows immediately, and therefore we focus on the if direction. First, we show that $\varphi := \bigwedge_{i=1}^{n}(\mathsf{w} \doteq \alpha_i)$ where $\alpha_i \in \big(\Sigma \cup (\Xi \setminus \{\mathsf{w}\})\big)^*$ for all $i \in [n]$ is a normal form for FC-CQ. This construction is performed in two steps:

Step One While there exists some atom of φ of the form $x \doteq \beta_1 \cdot \mathsf{w} \cdot \beta_2$, we replace this atom with $(\mathsf{w} \doteq x) \wedge (z \doteq \beta_1) \wedge (z \doteq \beta_2) \wedge (z \doteq \varepsilon)$, where $z \in \Xi$ is a new variable. We can show that the resulting formula is equivalent to φ using a length argument. It is clear that for any substitution σ such that $\sigma \models \varphi$, the equality $|\sigma(x)| = |\sigma(\beta_1)| + |\sigma(\mathsf{w})| + |\sigma(\beta_2)|$ holds. Since $|\sigma(x)| \leq |\sigma(\mathsf{w})|$, we know that $|\sigma(x)| = |\sigma(\mathsf{w})|$, and $|\sigma(\beta_1)| = |\sigma(\beta_2)| = 0$. This immediately implies that $\sigma(x) = \sigma(\mathsf{w})$ and $\sigma(\beta_1 \cdot \beta_2) = \varepsilon$.

Step Two While there exists some atom of φ of the form $(x \doteq \alpha)$ where $x \neq \mathsf{w}$, we replace this atom with $(\mathsf{w} \doteq p_x \cdot x \cdot s_x) \wedge (\mathsf{w} \doteq p_x \cdot \alpha \cdot s_x)$, where $p_x, s_x \in \Xi$ are new variables that are unique to x. It is clear that this re-writing step maintains equivalence between the given and resulting formulas using an analogous length argument to step one.

Actual Proof For the rest of the proof, we assume that $\varphi := \bigwedge_{i=1}^{n}(\mathsf{w} \doteq \alpha_i)$ where $\alpha_i \in \big(\Sigma \cup (\Xi \setminus \{\mathsf{w}\})\big)^*$ for all $i \in [n]$. Since $\varepsilon \in \mathcal{L}(\psi)$, it follows that for all $i \in [n]$, the pattern α_i is terminal-free (that is, $\alpha_i \in \Xi^*$).

We know that $\mathtt{a} \in \mathcal{L}(\varphi)$, therefore $\mathtt{a} \in \mathcal{L}(\alpha_i)$ for each $i \in [n]$. It follows that for any σ where $\sigma \models \varphi$ and $\sigma(\mathsf{w}) = \mathtt{a}$, each α_i has exactly one variable $x_i \in \mathsf{vars}(\alpha_i)$ such that $\sigma(x_i) = \mathtt{a}$ and for all $y \in \mathsf{vars}(\alpha_i) \setminus \{x_i\}$, we have that $\sigma(y) = \varepsilon$. Therefore, for any $w \in \Sigma^*$ we can define new substitution $\tau \models \varphi$, where $\tau(\mathsf{w}) = \tau(x_i) = w$ for all $i \in [n]$, and $\tau(y) = \varepsilon$ for all $y \in \mathsf{vars}(\varphi) \setminus \{x_i \mid i \in [n]\}$. Thus, $w \in \mathcal{L}(\varphi)$ for any $w \in \Sigma^*$. □

We shall also use Lemma 1 in Section 4 as a tool to help prove the complexity bounds for the *universality problem*.

Proposition 1 $\mathcal{L}$(FC-CQ) *and* $\mathcal{L}$(FC[REG]-CQ) *are closed under intersection and are not closed complement.* $\mathcal{L}$(FC-CQ) *is not closed under union.*

Proof We first show that $\mathcal{L}$(FC[REG]-CQ) and $\mathcal{L}$(FC-CQ) are closed under intersection. Let $\varphi_1 \in \mathcal{C}$ and $\varphi_2 \in \mathcal{C}$, for some $\mathcal{C} \in \{$FC-CQ, FC[REG]-CQ$\}$. Assume that $\mathsf{vars}(\varphi_1) \cap \mathsf{vars}(\varphi_2) = \emptyset$. We can make this assumption as if this does not hold, we simply rename variables for φ_2 uniformly. Then, we can define $\psi := \varphi_1 \wedge \varphi_2$. It follows from the definitions that $\mathcal{L}(\psi) = \mathcal{L}(\varphi_1) \cap \mathcal{L}(\varphi_2)$.

Next, we show that $\mathcal{L}(\mathsf{FC[REG]\text{-}CQ})$ is not closed under complement. To do so, we observe that $\mathcal{L}(\mathsf{FC[REG]\text{-}CQ})$ is a subclass of the language of *core spanners* (see [4, 5, 8]). It is not necessary for us to define core spanners, as we just use a language that cannot be expressed by core spanners.

Consider the *uniform-0-chunk language* which cannot be expressed by core spanners – see Theorem 6.1 from [5]. This language is defined as all words $w \in \{\mathtt{0}, \mathtt{1}\}^*$ such that $w = s_0 \cdot \prod_{i=1}^{n}(t \cdot s_i)$ for any $n \geq 1$, where $t \in \{\mathtt{0}\}^+$ and $s_i \in \{\mathtt{1}\}^+$ for all $i \in [n]$.

Let $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ and, working towards a contradiction, assume $\mathcal{L}(\mathsf{FC[REG]\text{-}CQ})$ is closed under complement. It follows immediately from De Morgan's law that $\mathcal{L}(\mathsf{FC[REG]\text{-}CQ})$ is also closed under union. We now construct $\varphi \in \mathsf{FC[REG]\text{-}CQ}$ such that $\mathcal{L}(\varphi)$ is the uniform-0-chunk language. First, let $\varphi_1 := (\mathsf{w} \,\dot{\in}\, \gamma_1)$ where $\mathcal{L}(\gamma_1) = \mathtt{1}^+ \cdot (\mathtt{0}^+ \cdot \mathtt{1}^+)^+$. Let

$$\varphi_2 := (\mathsf{w} \doteq z_1 \cdot \mathtt{0} \cdot y \cdot z_2 \cdot y \cdot z_3) \wedge \bigwedge_{x \in \{z_1, z_2, z_3, y\}} (x \,\dot{\in}\, \gamma_x),$$

where

- $\mathcal{L}(\gamma_{z_1}) = \Sigma^*$,
- $\mathcal{L}(\gamma_y) = \mathtt{0}^+$,
- $\mathcal{L}(\gamma_{z_2}) = (\mathtt{1}^+ \cdot \Sigma^* \cdot \mathtt{1}^+) \cup \mathtt{1}^+$, and
- $\mathcal{L}(\gamma_{z_3}) = \mathtt{1}^+ \cdot \Sigma^*$.

For intuition, we have that

$$\mathcal{L}(\varphi_2) = \Sigma^* \cdot \mathtt{0}^m \cdot \big((\mathtt{1}^+ \cdot \Sigma^* \cdot \mathtt{1}^+) \cup \mathtt{1}^+\big) \cdot \mathtt{0}^{m-1} \cdot \mathtt{1}^+ \cdot \Sigma^*.$$

Likewise, let

$$\varphi_3 := (\mathsf{w} \doteq \bar{z}_1 \cdot \bar{y} \cdot \bar{z}_3 \cdot \bar{y} \cdot \mathtt{0} \cdot \bar{z}_3) \wedge \bigwedge_{x \in \{\bar{z}_1, \bar{z}_2, \bar{z}_3, \bar{y}\}} (x \,\dot{\in}\, \gamma_x),$$

where

- $\mathcal{L}(\gamma_{\bar{z}_1}) = \Sigma^* \cdot \mathtt{1}^+$,
- $\mathcal{L}(\gamma_{\bar{y}}) = \mathtt{0}^+$,
- $\mathcal{L}(\gamma_{\bar{z}_2}) = (\mathtt{1}^+ \cdot \Sigma^* \cdot \mathtt{1}^+) \cup \mathtt{1}^+$, and
- $\mathcal{L}(\gamma_{\bar{z}_3}) = \Sigma^*$.

Analogously to $\mathcal{L}(\varphi_2)$, we have that

$$\mathcal{L}(\varphi_3) = \Sigma^* \cdot \mathtt{1}^+ \cdot \mathtt{0}^{m-1} \cdot \big((\mathtt{1}^+ \cdot \Sigma^* \cdot \mathtt{1}^+) \cup \mathtt{1}^+\big) \cdot \mathtt{0}^m \cdot \Sigma^*.$$

Thus, if follows that $\mathcal{L}(\varphi_1) \cap \big(\Sigma^* \setminus (\mathcal{L}(\varphi_2) \cup \mathcal{L}(\varphi_3)\big)$ is the uniform-0-chunk language which cannot be expressed by a $\mathsf{FC[REG]\text{-}CQ}$. Consequently, our assumption that $\mathcal{L}(\mathsf{FC[REG]\text{-}CQ})$ is closed under complement is incorrect.

Next, we show that $\mathcal{L}(\mathsf{FC\text{-}CQ})$ is not closed under union. Let $\varphi_1 := (\mathsf{w} \doteq \varepsilon)$ and $\varphi_2 := (\mathsf{w} \doteq \mathtt{a})$. It follows that $\mathcal{L}(\varphi_1) \cup \mathcal{L}(\varphi_2) = \{\varepsilon, \mathtt{a}\}$ which we know from Lemma 1 is a language that is not in $\mathcal{L}(\mathsf{FC\text{-}CQ})$. Since $\mathcal{L}(\mathsf{FC\text{-}CQ})$ is closed under intersection, and is not closed under union, it follows from De Morgan's law that $\mathcal{L}(\mathsf{FC\text{-}CQ})$ is not closed under complement. $\square$

3.1 Expressive Power

Our next focus is to look at the relative expressive power of FC[REG]-CQ and related models. We summarize our findings in Theorem 2, and the related Fig. 1. The proof of Theorem 2 is given as a series of lemmas.

For the language classes $\mathcal{L}_1$ and $\mathcal{L}_2$, we write $\mathcal{L}_1 \# \mathcal{L}_2$ to denote that $\mathcal{L}_1$ and $\mathcal{L}_2$ are *incomparable*. That is, there is some $L \in \mathcal{L}_1 \setminus \mathcal{L}_2$ and there is some $L' \in \mathcal{L}_2 \setminus \mathcal{L}_1$.

Since the pattern xx generates the non-regular language $\{ww \mid w \in \Sigma^*\}$, and the regular expression $\emptyset$ is not expressible by a pattern, $\mathcal{L}(\mathsf{PAT}) \# \mathcal{L}(\mathsf{REG})$. This further implies that $\mathcal{L}(\mathsf{REG}) \subset \mathcal{L}(\mathsf{PAT[REG]})$ and $\mathcal{L}(\mathsf{PAT}) \subset \mathcal{L}(\mathsf{PAT[REG]})$.

Lemma 2 *$\mathcal{L}(\mathsf{FC\text{-}CQ})$ and $\mathcal{L}(\mathsf{REG})$ are incomparable.*

Proof It is clear that $\{ww \mid w \in \Sigma^*\}$ is in $\mathcal{L}(\mathsf{FC\text{-}CQ})$ and is not regular. Furthermore, we know from [4] that the more general EP-FC cannot represent all regular languages. Thus, $\mathcal{L}(\mathsf{FC\text{-}CQ})$ and regular languages are incomparable. $\square$

Lemma 3 $\mathcal{L}(\mathsf{PAT}) \subset \mathcal{L}(\mathsf{FC\text{-}CQ})$.

Proof The inclusion $\mathcal{L}(\mathsf{PAT}) \subseteq \mathcal{L}(\mathsf{FC\text{-}CQ})$ follows trivially from the definitions. We now show that $\mathcal{L}(\mathsf{PAT}) \neq \mathcal{L}(\mathsf{FC\text{-}CQ})$.

Let $\varphi := (x \doteq \mathtt{a}) \land (y \doteq \mathtt{b})$. Thus, we have that

$$\mathcal{L}(\varphi) = \{w \mid w \in \Sigma^* \text{ and } |w|_{\mathtt{a}} \geq 1 \text{ and } |w|_{\mathtt{b}} \geq 1\}.$$

Working towards a contradiction, assume that there exists $\alpha \in (\Sigma \cup \Xi)^*$ such that $\mathcal{L}(\alpha) = \mathcal{L}(\varphi)$. We first prove that $|\alpha|_{\mathtt{a}} = 1$ and $|\alpha|_{\mathtt{b}} = 1$. To show this, assume that $|\alpha|_{\mathtt{a}} = 0$. Then, consider the pattern substitution σ where $\sigma(x) = \mathtt{b}$ for all $x \in \mathsf{vars}(\alpha)$. Hence, there exists $w \in \mathcal{L}(\alpha)$ such that $|w|_{\mathtt{a}} = 0$, which is a contradiction. The same reasoning shows that $|\alpha|_{\mathtt{b}} \geq 1$. Now, if $|\alpha|_{\mathtt{a}} > 1$ or $|\alpha|_{\mathtt{b}} > 1$, then $\mathtt{ab} \notin \mathcal{L}(\alpha)$ which is a contradiction.

We now have two possibilities for α. If the symbol $\mathtt{a}$ is left of $\mathtt{b}$ in α, then $\mathtt{ba} \notin \mathcal{L}(\alpha)$. If the symbol $\mathtt{b}$ is left of $\mathtt{a}$ in α, then $\mathtt{ab} \notin \mathcal{L}(\alpha)$. Since $\mathtt{ab}, \mathtt{ba} \in \mathcal{L}(\varphi)$, we have a contradiction and therefore $\mathcal{L}(\varphi)$ is not a pattern language. $\square$

Lemma 4 $\mathcal{L}(\mathsf{FC\text{-}CQ}) \subset \mathcal{L}(\mathsf{FC[REG]\text{-}CQ})$, *and* $\mathcal{L}(\mathsf{FC\text{-}UCQ}) \subset \mathcal{L}(\mathsf{FC[REG]\text{-}UCQ})$.

Proof From the definitions, we immediately can determine that $\mathcal{L}(\mathsf{FC\text{-}CQ}) \subseteq \mathcal{L}(\mathsf{FC[REG]\text{-}CQ})$ and $\mathcal{L}(\mathsf{FC\text{-}UCQ}) \subseteq \mathcal{L}(\mathsf{FC[REG]\text{-}UCQ})$. The fact that $\mathcal{L}(\mathsf{FC\text{-}CQ}) \neq \mathcal{L}(\mathsf{FC[REG]\text{-}CQ})$ and $\mathcal{L}(\mathsf{FC\text{-}UCQ}) \neq \mathcal{L}(\mathsf{FC[REG]\text{-}UCQ})$ follows from the fact that EP-FC cannot represent all regular languages (see [4]). $\square$

Lemma 5 $\mathcal{L}(\mathsf{PAT[REG]}) \# \mathcal{L}(\mathsf{FC\text{-}CQ})$ *and* $\mathcal{L}(\mathsf{PAT[REG]}) \subset \mathcal{L}(\mathsf{FC[REG]\text{-}CQ})$.

Proof Notice that for every $\alpha_T \in \mathsf{PAT[REG]}$, there is an equivalent $\mathsf{FC[REG]\text{-}CQ}$. More formally, let $\alpha_T := (\alpha, T)$ and let

$$\varphi_{\alpha_T} := (\mathsf{w} \doteq \alpha) \wedge \bigwedge_{x \in \mathsf{vars}(\alpha)} \big(x \,\dot{\in}\, T(x)\big).$$

It is clear from the definitions that $\mathcal{L}(\alpha_T) = \mathcal{L}(\varphi_{\alpha_T})$. It therefore follows that $\mathcal{L}(\mathsf{PAT[REG]}) \subseteq \mathcal{L}(\mathsf{FC[REG]\text{-}CQ})$. Therefore, if $\mathcal{L}(\mathsf{PAT[REG]}) \# \mathcal{L}(\mathsf{FC\text{-}CQ})$, then the inclusion $\mathcal{L}(\mathsf{PAT[REG]}) \subset \mathcal{L}(\mathsf{FC[REG]\text{-}CQ})$ immediately follows. The rest of this proof focuses on showing $\mathcal{L}(\mathsf{PAT[REG]}) \# \mathcal{L}(\mathsf{FC\text{-}CQ})$.

We know that there are regular languages that cannot be expressed in $\mathsf{FC\text{-}CQ}$. Consequently, $\mathcal{L}(\mathsf{PAT[REG]}) \setminus \mathcal{L}(\mathsf{FC\text{-}CQ}) \neq \emptyset$: For example, consider Lemma 1 and $(\mathtt{a} \vee \varepsilon)$. Next, we show that $\mathcal{L}(\mathsf{FC\text{-}CQ}) \setminus \mathcal{L}(\mathsf{PAT[REG]}) \neq \emptyset$. Consider

$$\varphi := (x_1 \doteq \mathtt{a}) \wedge (x_2 \doteq \mathtt{b} \cdot y \cdot \mathtt{b} \cdot y \cdot \mathtt{b}).$$

We can represent $\mathcal{L}(\varphi)$ as

$$\mathcal{L}(\varphi) := (\Sigma^* \cdot \mathtt{a} \cdot \Sigma^*) \cap (\Sigma^* \cdot \{\mathtt{b} \cdot u \cdot \mathtt{b} \cdot u \cdot \mathtt{b} \mid u \in \Sigma^*\} \cdot \Sigma^*).$$

First, we show that $\mathcal{L}(\varphi)$ is not regular. To the contrary, assume that $\mathcal{L}(\varphi)$ is regular. Since regular languages are closed under intersection we have that

$$L_1 := \mathcal{L}(\varphi) \cap (\mathtt{b} \cdot \mathtt{a}^* \cdot \mathtt{b} \cdot \mathtt{a}^* \cdot \mathtt{b})$$

is regular. Where, from the definition of $\mathcal{L}(\varphi)$, we have

$$L_1 = \{\mathtt{b} \cdot \mathtt{a}^n \cdot \mathtt{b} \cdot \mathtt{a}^n \cdot \mathtt{b} \mid n \in \mathbb{N}\}.$$

Proving L_1 is non-regular is straightforward exercise in the pumping lemma for regular languages (for example, see [18]). Therefore, we can continue with the proof with the knowledge that $\mathcal{L}(\varphi)$ is not regular.

Assume there exists a regularly typed pattern language $\alpha_T := (\alpha, T)$ such that $\mathcal{L}(\alpha_T) = \mathcal{L}(\varphi)$. Further assume that $\alpha := \alpha_1 \cdot \alpha_2 \cdots \alpha_n$ where $\alpha_i \in \Xi$ for all $i \in [n]$. We can make the assumption that α is terminal free because every terminal symbol $\mathtt{a}$ can be represented as a new variable $x_\mathtt{a}$ with the regular type $T(x_\mathtt{a}) := \{\mathtt{a}\}$.

Since for all $w \in \mathcal{L}(\varphi)$, we have that $|w|_\mathtt{a} \geq 1$, it follows that there is some $i \in [n]$ such that for all $u \in T(\alpha_i)$, we have that $|u|_\mathtt{a} \geq 1$. Otherwise, there is a word $w \in \mathcal{L}(\alpha_T)$ such that $|w|_\mathtt{a} = 0$. This can be seen by picking a substitution τ such that $\tau(\alpha) \in \mathcal{L}(\alpha_T)$ and $|\tau(x)|_\mathtt{a} = 0$ for all $x \in \mathsf{vars}(\alpha)$. Note that $|\alpha|_{\alpha_i} = 1$ and there cannot exist $i' \in [n] \setminus \{i\}$ where for all $u \in T(\alpha_{i'})$ we have that $|u|_\mathtt{a} \geq 1$. Otherwise, $\mathtt{abbb} \in \mathcal{L}(\varphi)$ would not be expressible. We call α_i the $\mathtt{a}$-*keeper*.

Likewise, there must be some $x \in \mathsf{vars}(\alpha)$, where for all $v \in T(x)$ we have that $|v|_\mathtt{b} \geq 1$. But, x may not be unique, and $|\alpha|_x = 1$ does not necessarily hold. We call such a variable $x \in \mathsf{vars}(\alpha)$ a $\mathtt{b}$-*keeper*. Let $j \in [n]$ be the smallest (or left-most) position such that α_j is a $\mathtt{b}$-keeper, and let $k \in [n]$ be the largest (or right-most) position such that α_k is a $\mathtt{b}$-keeper.

For any $i' \in [n]$ where $\alpha_{i'}$ is not the $\mathtt{a}$-keeper, nor a $\mathtt{b}$-keeper, we have that $\varepsilon \in T(\alpha_{i'})$. This is because we have shown there is exactly one $\mathtt{a}$-keeper, and there exists $w \in \mathcal{L}(\varphi)$ such that $|w|_{\mathtt{c}} = 0$ for all $\mathtt{c} \in \Sigma \setminus \{\mathtt{a}, \mathtt{b}\}$.

We now look at some cases for the relative positions of i, j, and k, and prove a contradiction for each:

Case 1. $i < j$: For this case, we consider those $w \in \mathcal{L}(\alpha_T)$ where $|w|_{\mathtt{a}} = 1$, $|w|_{\mathtt{b}} = 3$ and $|w|_{\mathtt{c}} = 0$ for all $\mathtt{c} \in \Sigma \setminus \{\mathtt{a}, \mathtt{b}\}$. Due to the fact that $i < j$, we have that the $\mathtt{a}$-keeper appears to the left of the left-most $\mathtt{b}$-keeper. Therefore, for any word in $\mathcal{L}(\alpha_T)$, the $\mathtt{a}$-symbol must appear to the left of some $\mathtt{b}$-symbol. Thus, $\mathtt{bbba} \in \mathcal{L}(\varphi) \setminus \mathcal{L}(\alpha_T)$.

Case 2. $i > k$: For this case, we again consider those $w \in \mathcal{L}(\alpha_T)$ where $|w|_{\mathtt{a}} = 1$, $|w|_{\mathtt{b}} = 3$ and $|w|_{\mathtt{c}} = 0$ for all $\mathtt{c} \in \Sigma \setminus \{\mathtt{a}, \mathtt{b}\}$. Analogously to Case 1, the $\mathtt{a}$-symbol must come to the right of some $\mathtt{b}$-symbol. It therefore follows that $\mathtt{abbb} \in \mathcal{L}(\varphi) \setminus \mathcal{L}(\alpha_T)$.

Case 3. $j < i < k$: Again, we consider those $w \in \mathcal{L}(\alpha_T)$ where $|w|_{\mathtt{a}} = 1$, $|w|_{\mathtt{b}} = 3$, and $|w|_{\mathtt{c}} = 0$ for all $\mathtt{c} \in \Sigma \setminus \{\mathtt{a}, \mathtt{b}\}$. For any $w \in \mathcal{L}(\alpha_T)$ where $|w|_{\mathtt{a}} = 1$, we have that the $\mathtt{a}$-symbol must come between two $\mathtt{b}$-symbols (analogously to Case 1 and Case 2). It therefore follows that $\mathtt{bbba} \in \mathcal{L}(\varphi) \setminus \mathcal{L}(\alpha_T)$.

Case 4. $i = j = k$: Let $x = \alpha_i$. We know that for all $y \in \mathsf{vars}(\alpha)$ where $y \neq x$, we have that $\varepsilon \in T(y)$ since it is neither an $\mathtt{a}$-keeper, nor a $\mathtt{b}$-keeper. We hence claim that $T(x) \subseteq \mathcal{L}(\varphi)$. To prove this claim, let $w \in T(x) \setminus \mathcal{L}(\varphi)$. We can then define a substitution σ such that $\sigma(x) = w$ and $\sigma(y) = \varepsilon$ for all $y \in \mathsf{vars}(\alpha)$ where $y \neq x$. Consequently, for all $w \in T(x)$, we have that $|w|_{\mathtt{a}} \geq 1$ and $|w|_{\mathtt{b}} \geq 3$. Since otherwise, $\mathcal{L}(\varphi) \neq \mathcal{L}(\alpha_T)$. We now claim that $T(x) \cap (\mathtt{b} \cdot \mathtt{a}^* \cdot \mathtt{b} \cdot \mathtt{a}^* \cdot \mathtt{b}) = \mathtt{b} \cdot \mathtt{a}^n \cdot \mathtt{b} \cdot \mathtt{a}^n \cdot \mathtt{b}$. To prove this claim, assume the contrary. Then, there is some $w \in T(x)$ where $w = \mathtt{b} \cdot \mathtt{a}^p \cdot \mathtt{b} \cdot \mathtt{a}^q \cdot \mathtt{b}$ where $p, q \in \mathbb{N}$ and $q \neq p$. This is a contradiction, since $T(x) \subseteq \mathcal{L}(\varphi)$ does not hold. Consequently, $T(x)$ is not regular which is a contradiction.

Hence, $\mathcal{L}(\alpha_T) \neq \mathcal{L}(\varphi)$ and thus there does not exist a regularly typed pattern that can generate $\mathcal{L}(\varphi)$. Consequently, $\mathcal{L}(\varphi) \in \mathcal{L}(\mathsf{FC\text{-}CQ}) \setminus \mathcal{L}(\mathsf{PAT[REG]})$. □

The focus of the proof of Lemma 5 is giving a language that is in $\mathcal{L}(\mathsf{FC\text{-}CQ})$ but is not in $\mathcal{L}(\mathsf{PAT[REG]})$. This immediately implies the following:

- $\mathcal{L}(\mathsf{PAT[REG]}) \# \mathcal{L}(\mathsf{FC\text{-}UCQ})$ as FC-UCQ cannot represent all regular languages and we have given some $\mathcal{L} \in \mathcal{L}(\mathsf{FC\text{-}UCQ}) \setminus \mathcal{L}(\mathsf{PAT[REG]})$, and
- $\mathcal{L}(\mathsf{PAT[REG]}) \subset \mathcal{L}(\mathsf{FC[REG]\text{-}CQ})$ as each regularly typed pattern language can be easily written as an FC[REG]-CQ.

We now summarize our observations on the relative expressive power in the following Theorem.

Theorem 2 *Fig. 1 describes the relations between language classes.*

The exact relationship between FC-UCQ and FC[REG]-CQ remains open. From [4], we know that FC-UCQ cannot express all regular languages. Therefore, there are languages $L \subseteq \Sigma^*$ such that $L \in$ FC[REG]-CQ $\setminus$ FC-UCQ. The authors conjecture that $\mathcal{L}$(FC-UCQ) and $\mathcal{L}$(FC[REG]-CQ) are incomparable.

We leave the following problems open:

Open Problem 1. $\mathcal{L}(\textsf{FC-UCQ}) = \mathcal{L}(\textsf{EP-C})$?
Open Problem 2. $\mathcal{L}(\textsf{FC[REG]-CQ}) = \mathcal{L}(\textsf{FC[REG]-UCQ})$?
Open Problem 3. $\mathcal{L}(\textsf{FC[REG]-CQ}) = \mathcal{L}(\textsf{EP-C[REG]})$?

The authors conjecture that $\mathcal{L}(\textsf{FC[REG]-CQ}) \subset \mathcal{L}(\textsf{FC[REG]-UCQ})$, and therefore conjecture that $\mathcal{L}(\textsf{FC[REG]-CQ}) \subset \mathcal{L}(\textsf{EP-C[REG]})$.

3.2 Connection to Related Models

In the proof of Lemma 1, we showed how to re-write formulas of the form $(x \doteq \alpha)$ as $(\mathsf{w} \doteq p_x \cdot \alpha \cdot s_x) \wedge (\mathsf{w} \doteq p_x \cdot x \cdot s_x)$, where $p_x, s_x \in \Xi$ are new and unique variables. Hence, every FC-CQ-formula can be represented as $\varphi := \bigwedge_{i=1}^{n} (\mathsf{w} \doteq \alpha_i)$, and therefore can be understood as a system of word equations $\{(\mathsf{w} \doteq \alpha_1), (\mathsf{w} \doteq \alpha_2), \ldots, (\mathsf{w} \doteq \alpha_n)\}$ that generates the language

$$\{\sigma(\mathsf{w}) \mid \sigma \models (\mathsf{w} \doteq \alpha_i) \text{ for all } i \in [n]\}.$$

Consequently, FC-CQ languages can be seen as a natural extension of pattern languages, or as the languages of restricted cases of systems of word equations. FC[REG]-CQ analogously extends regularly typed pattern languages.

We further explore the connection between FC[REG]-CQ and related language generators by considering a restricted case of xregex. These are regular expressions that are extended with a repetition operator that allows for the definition of non-regular languages, and that is available in most modern regular expression implementations (see for example [19], which also contains a discussion of some peculiarities of these implementations).

Definition 4 We define the set of xregex recursively:

$$\gamma := \emptyset \mid \varepsilon \mid \mathsf{a} \mid (\gamma \cup \gamma) \mid (\gamma \cdot \gamma) \mid (\gamma)^* \mid x\{\gamma\} \mid \&x,$$

where $\mathsf{a} \in \Sigma$ and $x \in \Xi$. For the purposes of this short section on xregex, we assume that for every variable x that occurs in γ, we have that $x\{\lambda\}$, for some $\lambda \in$ xregex, must occur exactly once.

Informally, an xregex of the form $x\{\gamma\}$ matches the same words as γ, and also "stores" the matched word in x. Then, any occurrence of $\&x$ repeats the word stored in x.

Let $\gamma \in$ xregex. If $x\{\lambda\}$ is a subexpression of γ, and $\&y$ is a subexpression of λ, then we say that x *depends* on y. If x depends on y, and y depends on z, then x depends on z. We assume that for all $\gamma \in$ xregex, if x depends on y, then y cannot depend on x. Furthermore, we assume that x cannot depend on x. This avoids defining the language for troublesome expressions such as $x\{\mathtt{a} \cdot \&y \cdot \&x\} \cdot y\{\&x \cdot \mathtt{b}\}$.

Example 5 Let $\gamma_2 := x\{\mathtt{a} \cdot \Sigma^*\} \cdot (\&x)^*$. Then, $\mathcal{L}(\gamma_2)$ is the language of words of the form $(\mathtt{a} \cdot w)^n$ where $w \in \Sigma^*$ and $n \geq 1$.

While the term regular expression is sometimes used to refer to expressions that contain variables, we reserve the term regular expression for its more "classical" definition.

Definition 5 We call $\gamma \in \mathsf{xregex}$ a *regex path* if for every subexpression of the form $(\lambda)^*$, we have that λ is a regular expression, and for every subexpression of the form $(\lambda_1 \cup \lambda_2)$, we have that λ_1 and λ_2 are regular expressions. Let XRP denote the set of regex paths.

Regex paths were considered in [10] (in the context of document spanners). Since we only work with XRP, it is unnecessary for us to define the languages generated by an xregex. Therefore, we do not need to deal with cases such as $x\{\lambda\}$ is a subexpression of $(\gamma)^*$, or $((x\{\mathtt{aba}^*\}) \cup (\&x))$.

To define the language generated by an XRP, we use *ref-words*, which were originally introduced by Schmid [20].

Definition 6 The language defined by $\gamma \in \mathsf{XRP}$ is defined in a two-step process. First, we say that γ defines a language of *ref-words*, denoted by $\mathcal{R}(\gamma)$, over the alphabet $(\Sigma \cup \Delta)^*$, where $\Delta := \bigcup_{x \in \Xi}\{\vdash_x, \dashv_x, \&x\}$. We define $\mathcal{R}(\gamma)$ as follows:

- if $\gamma \in \Sigma \cup \{\&x \mid x \in \Xi\} \cup \{\emptyset\} \cup \{\varepsilon\}$, then $\mathcal{R}(\gamma) = \gamma$,
- if $\gamma = (\gamma_1 \cup \gamma_2)$, then $\mathcal{R}(\gamma) = \mathcal{R}(\gamma_1) \cup \mathcal{R}(\gamma_2)$,
- if $\gamma = (\gamma_1 \cdot \gamma_2)$, then $\mathcal{R}(\gamma) = \mathcal{R}(\gamma_1) \cdot \mathcal{R}(\gamma_2)$,
- if $\gamma = (\lambda)^*$ for some $\lambda \in \mathsf{XRP}$, then $\mathcal{R}(\gamma) = (\mathcal{R}(\lambda))^*$, and
- if $\gamma = x\{\lambda\}$ for some $\lambda \in \mathsf{XRP}$, then $\mathcal{R}(\gamma) = \vdash_x \mathcal{R}(\lambda) \dashv_x$.

Let $\mathsf{clr}\colon (\Sigma \cup \Delta)^* \to \Sigma^*$ be a partial morphism, such that $\mathsf{clr}(\mathtt{a}) = \mathtt{a}$ for all $\mathtt{a} \in \Sigma^*$, and $\mathsf{clr}(x) = \varepsilon$ otherwise.

Every $u \in \mathcal{R}(\gamma)$ gives rise to a word $w \in \Sigma^*$ using the following construction: Let $\bar{u} \in (\Sigma \cup \{\vdash_x, \dashv_x\})^*$ be defined as the word that results from iteratively replacing every occurrence of $\&x$ in γ with v, where $\vdash_x v \dashv_x \sqsubseteq u$ until no such $\&x$ occurs. Then $w := \mathsf{clr}(\bar{u})$. We call such a word w the *derivation* of u. Finally, we define

$$\mathcal{L}(\gamma) := \{w \in \Sigma^* \mid w \text{ is the derivation of some } u \in \mathcal{R}(\gamma)\}.$$

Note that many definitions for the semantics of xregex exist; for example, see [20, 21]. For our purposes, the definition for XRP languages given in Definition 6 is sufficient.

Using a proof similar to the proof of Lemma 3.19 in [10] and the proof of Lemma 3.6 in [8], we show the following:

Proposition 3 *For every $\gamma \in \mathsf{XRP}$, we can effectively construct a formula $\varphi \in \mathsf{FC[REG]\text{-}CQ}$, such that $\mathcal{L}(\varphi) = \mathcal{L}(\gamma)$.*

Proof Let $\gamma \in \mathsf{XRP}$. We construct $\mathsf{FC[REG]\text{-}CQ}$ such that $\mathcal{L}(\mathsf{FC[REG]\text{-}CQ}) = \mathcal{L}(\gamma)$ by an inductive proof on the definition of XRP. To aid in this construction, we represent γ as a rooted parse tree T_γ, which we define recursively. Let γ be the root of T_γ, then:

- If γ is a regular expression or $\gamma = \&x$ for some $x \in \Xi$, then γ is a leaf node,
- if $\gamma = (\gamma_1 \cdot \gamma_2)$ where either γ_1 or γ_2 contains a variable, then γ has γ_1 as a left child, and γ_2 as a right child, and
- if $\gamma = x\{\lambda\}$ for some $\lambda \in \mathsf{XRP}$, then γ has λ as a single child.

We then associate each node n of T_γ with a variable $v_n \in \Xi$ as follows: If n is of the form $x\{\lambda\}$, or $\&x$ for some $x \in \Xi$, then let $v_n := x$. Otherwise, let $v_n := z_n$, where z_n is a new variable that is unique to n.

Then, for every node n, we also associate an atom A_n (either a word equation, or a regular constraint) as follows:

- If n is a concatenation with left child l, and right child r, then we have that $A_n := (v_n \doteq v_l \cdot v_r)$,
- if n is of the form $x\{\lambda\}$, let $A_n := (v_n \doteq v_c)$, where c is the child of n. Recall that for this case, we have that $v_n := x$,
- if n is a regular expression λ, let $A_n := (v_n \mathrel{\dot{\in}} \lambda)$, and
- if n is of the form $\&x$, let $A_n := (v_n \doteq x)$.

Let V be the set of nodes in T_γ, and let r be the root of T_γ. Then, let

$$\varphi_\gamma := \bigwedge_{n \in V} A_n \wedge (\mathsf{w} \doteq v_r).$$

We prove the correctness of this construction inductively. First, for leaf nodes of T_γ that are regular expressions, the correctness follows immediately from the fact that we have simply represented the regular expression as a regular constraint. To represent leaf nodes of the form $\&x$ in φ_γ, we simply use the variable x.

Next, we consider non-leaf nodes of T_γ. If n is a non-leaf node, then either $n = x\{\lambda\}$ for some $\lambda \in \mathsf{XRP}$, or $n = (\lambda_1 \cdot \lambda_2)$. For the case where $n = (\lambda_1 \cdot \lambda_2)$, the fact that correctness holds follows from the fact that n is represented with the atom $(v_n \doteq v_l \cdot v_r)$, where v_l and v_r are the variables that represent λ_1 and λ_2 respectively. Lastly, if n is a non-leaf node of the form $x\{\lambda\}$, we have that n is associated with the atom $(x \doteq v_c)$, where v_c is the variable that represents λ.

To conclude the proof, we have the extra word equation $(\mathsf{w} \doteq v_r)$ where v_r is the variable that represents the root node γ, it follows that $\mathcal{L}(\varphi_\gamma) = \mathcal{L}(\gamma)$. □

Thus, a large and natural class of xregex can be represented by FC[REG]-CQ, and the conversion is lightweight and straightforward. If one is willing to go beyond FC[REG]-CQs, this could be adapted to larger classes of xregex, although simulating edge cases in the definition of the latter might make the formulas a bit unwieldy. But a big advantage of logic over xregex is that optimization techniques from relational first-order logic (such as acyclicity [8] or bounded-width [4]) directly lead to tractable fragments. Furthermore, since we are able to use conjunction directly in an FC[REG]-CQ-formula, FC[REG]-CQ can be used as a more compact representation of XRP.

4 Decision Problems

The first decision problem we shall look at is the membership problem for FC-CQ defined as follows: Given $w \in \Sigma^*$ and a formula $\varphi \in$ FC-CQ as input, decide whether $w \in \mathcal{L}(\varphi)$. But first, we define the class of *regular patterns*:

Definition 7 A pattern $\alpha \in (\Sigma \cup \Xi)^*$ is a *regular pattern* if $|\alpha|_x = 1$ for every variable $x \in \mathsf{vars}(\alpha)$. An FC-CQ consists only of regular patterns if it is of the form $\bigwedge_{i=1}^{n}(\mathsf{w} \doteq \alpha_i)$ where α_i is a regular pattern for each $i \in [n]$.

The regular patterns have desirable algorithmic properties (although this comes at the expense of expressive power). For example, the membership problem for regular pattern languages can be solved in linear time [22]. When extending regular patterns to FC-CQ, these desirable algorithmic properties do not carry over.

Theorem 4 *The membership problem for* FC-CQ *is* NP*-complete, and remains* NP*-hard even if the formula consists only of regular patterns and the input word is of length one.*

Proof The upper bound is trivial and we therefore focus on the NP-hardness proof for the restricted class of FC-CQs considered in the theorem's statement. To prove NP-hardness when the formula consists only of regular patterns, and the input word is of length one, we reduce from the NP-complete problem of *1-in-3-SAT*. An instance of 1-in-3 SAT consists of a conjunction of clauses $P := C_1 \wedge C_2 \wedge \cdots \wedge C_m$ and a set of propositional variables $\{y_1, y_2, \ldots, y_k\}$.

Each clause C_i is defined as a disjunction of exactly three distinct literals, where each literal is either y_j or $\neg y_j$ for $j \in [k]$. A satisfying assignment for P is an assignment $\tau\colon \{y_1, \ldots, y_k\} \to \{0, 1\}$ that satisfies P, and exactly one literal in every clause is evaluated to 1 (every other literal must evaluate to 0).

For every propositional variable y, let $\varphi_y := (\mathsf{w} \doteq y_F \cdot y_T)$ where $y_F, y_T \in \Xi$. For a literal ℓ, let $x_\ell := y_F$ if $\ell = \neg y$, and let $x_\ell := y_T$ otherwise; where $y_F, y_T \in \Xi$. Then, for a clause $C = (\ell_1 \vee \ell_2 \vee \ell_3)$ where ℓ_1, ℓ_2, ℓ_3 are literals, let $\psi_C := (\mathsf{w} \doteq x_{\ell_1} \cdot x_{\ell_2} \cdot x_{\ell_3})$.

Given an instance $P := C_1 \wedge C_2 \wedge \cdots \wedge C_m$ of 1-in-3-SAT with k variables, let

$$\varphi_P := \bigwedge_{i=1}^{k} \varphi_{y_i} \wedge \bigwedge_{j=1}^{m} \psi_{C_j}.$$

It is clear that φ_P only consists of regular patterns. Next, we prove that $\sigma \models \varphi_P$ where $\sigma(\mathsf{w}) = \mathtt{a}$ if and only if P is satisfiable: Assume there exists a substitution σ such that $\sigma \models \varphi$ and $\sigma(\mathsf{w}) = \mathtt{a}$. Since $\sigma(\mathsf{w}) = \mathtt{a}$, for every variable y_i for $i \in [k]$ we have that $\sigma(y_{i_F} \cdot y_{i_T}) = \mathtt{a}$, thus either $\sigma(y_{i_T}) = \mathtt{a}$ and $\sigma(y_{i_F}) = \varepsilon$, or vice versa. This encodes y_i as true or false.

We encode each clause C_i as φ_{C_i} which contains a concatenation of variables that correspond to the literals that appear in C_i. Since we have that $\sigma \models \psi_{C_j}$ for all $j \in [m]$, it follows that exactly one of the variables that encodes a literal of C_j must be $\mathtt{a}$, and thus the other literals must be ε. This corresponds to every clause in P being evaluated to true, and exactly one literal from every clause being evaluated to true. Hence, if $\sigma \models \varphi$ where $\sigma(\mathsf{w}) = \mathtt{a}$ if and only if P is satisfiable. □

We note that the proof of Theorem 4 is fairly standard and only requires minor tweaks from the proof that membership for pattern languages is NP-complete [23] even under strict restrictions [24–26]. We give a proof as we shall require that membership for FC-CQ remains NP-hard even if the input word is of length one for Corollary 1.

Our next focus is on static analysis problems.

Definition 8 For each of FC-CQ and FC[REG]-CQ, we define the following static analysis problems:

1. *Universality:* Is $\mathcal{L}(\varphi) = \Sigma^*$?
2. *Emptiness:* Is $\mathcal{L}(\varphi) = \emptyset$?
3. *Regularity:* Is $\mathcal{L}(\varphi)$ regular?
4. *Equivalence:* Is $\mathcal{L}(\varphi_1) = \mathcal{L}(\varphi_2)$?

First, we consider the complexity bounds of the universality problem for FC-CQ.

Corollary 1 *Universality is* NP*-complete for* FC-CQ.

Proof First recall Lemma 1. Given $\varphi \in$ FC-CQ, we have that $\mathcal{L}(\varphi) = \Sigma^*$ if and only if $\varepsilon, \mathtt{a} \in \mathcal{L}(\varphi)$. From Theorem 4, it follows that the upper bounds are trivial. For the lower bounds, consider the construction of a formula φ_P from an instance of 1-in-3-SAT P given in the proof of Theorem 4. Clearly $\varepsilon \in \mathcal{L}(\varphi_P)$. Therefore, to decide whether $\mathcal{L}(\varphi_P) = \Sigma^*$, it is necessary and sufficient to decide whether $\mathtt{a} \in \mathcal{L}(\varphi_P)$ which is NP-hard (Theorem 4). □

Proposition 5 *Emptiness is* PSPACE*-complete for* FC[REG]-CQ, *and emptiness is* NP-*hard and in* PSPACE *for* FC-CQ.

Proof Since FC-CQ is a strict subclass of FC[REG]-CQ, we prove the PSPACE upper bounds for FC-CQ and FC[REG]-CQ simultaneously.

PSPACE ***Upper Bounds*** The emptiness problem for $\mathcal{C} \in \{$FC-CQ, FC[REG]-CQ$\}$ is: Given $\varphi \in \mathcal{C}$, decide whether there exists any $w \in \Sigma^*$ such that $w \in \mathcal{L}(\varphi)$. However, this is equivalent to determining whether there exists some w safe substitution σ such that $\sigma \models \varphi$. This is known as the *Satisfiability problem*. Since the satisfiability problem is in PSPACE for EP-C[REG]; for example, see [27], it follows that the emptiness problem for $\mathcal{C}$ is in PSPACE, for any $\mathcal{C} \in \{$FC[REG]-CQ, FC-CQ$\}$.

FC[REG]-CQ ***Lower Bounds*** We reduce from the *satisfiability problem for word equations with regular constraints*. That is, given instances $(\alpha \doteq \beta, T)$ where $\alpha, \beta \in (\Sigma \cup \Xi)^*$ and T is a typing function that maps variables in α and β to regular languages, decide whether there exists a pattern substitution $\sigma : (\Sigma \cup \Xi)^* \to \Sigma^*$ such that $\sigma(\alpha) = \sigma(\beta)$ and $\sigma(x) \in T(x)$ for variables x that appear in α and β. Then, $(\alpha \doteq \beta, T)$ is satisfiable if and only if the language for $\varphi := (\mathsf{w} \doteq \alpha) \land (\mathsf{w} \doteq \beta) \land \bigwedge_{x \in \mathsf{vars}(\alpha\beta)} (x \mathrel{\dot\in} T(x))$ is non-empty. Since the satisfiability problem for word equations with regular constraints is PSPACE-complete (see Theorem 7 of [28]), the emptiness problem for FC[REG]-CQ is PSPACE-complete.

FC-CQ ***Lower Bounds***
We reduce from the pattern language membership problem, which is an NP-complete problem [23]. Given $w \in \Sigma^*$ and a pattern α, construct $\varphi := (\mathsf{w} \doteq w) \wedge (\mathsf{w} \doteq \alpha)$. If $w \in \mathcal{L}(\alpha)$, then $\mathcal{L}(\varphi) = \{w\}$. Otherwise, $\mathcal{L}(\varphi) = \emptyset$. Hence, FC-CQ-emptiness is NP-hard. $\square$

Showing the exact complexity bounds of the emptiness problem for FC-CQ seems rather difficult. This is because the emptiness problem for FC-CQ is two-way polynomial-time reducible from the satisfiability problem for word equations. More precisely, the word equation $(\alpha_L \doteq \alpha_R)$ is satisfiable if and only if the language generated by $\varphi := (\mathsf{w} \doteq \alpha_L) \wedge (\mathsf{w} \doteq \alpha_R)$ is non-empty. Furthermore, from $\varphi := \bigwedge_{i=1}^{n}(x_i \doteq \alpha_i)$, one can construct – in polynomial time – a word equation $(\alpha_L \doteq \alpha_R)$ such that $\mathcal{L}(\varphi) \neq \emptyset$ if and only if $(\alpha_L \doteq \alpha_R)$ is satisfiable; see Section 5.3 in Chapter 6 of [17]. Thus, the emptiness problem for FC-CQ is a reformulation of the big open problem as to whether word equation satisfiability is in NP. See [29, 30] for more details on word equation satisfiability.

Next, we consider universality for FC[REG]-CQ and regularity for FC-CQ. However, these problems require some technical preparations. We define so-called extended Turing machines, which were introduced by Freydenberger [31]. The following definitions and descriptions follows closely to the definition and description in [31]. However, some details that are not important for our use have been omitted. Refer to [31] for these omitted details. While these extended Turing machines were introduced for the particulars of so-called *extended regular expressions*, we observe that they are useful for our purposes. They have also been used to show universality for EP-FC is undecidable (Theorem 4.7 of [4]).

4.1 Extended Turing Machines

An *extended Turing machine* is denoted as a triple $\mathcal{X} := (Q, q_1, \delta)$ where $Q := \{q_1, q_2, \ldots, q_k\}$ for some $k \geq 1$ is the set of states, $q_1 \in Q$ is the initial state, and δ is the transition function.

Extended Turing machines have a tape alphabet of $\Gamma := \{0, 1\}$ where 0 is used as the blank symbol. Let us now define the transition function

$$\delta \colon \Gamma \times Q \to (\Gamma \times \{L, R\} \times Q) \cup \{\mathsf{HALT}\} \cup (\mathsf{CHECK}_R \times Q).$$

If $\delta(\mathtt{a}, q) = (\mathtt{b}, M, p)$ where $\mathtt{a}, \mathtt{b} \in \Gamma$ are tape symbols, $p, q \in Q$ are states, and $M \in \{L, R\}$, then the machine replaces the symbol under the head (in this case $\mathtt{a}$) with $\mathtt{b}$, moves the head either to the left or the right (depending on M), and enters state p. As a convention, we adopt the assumption from [31] that $\delta(0, q_1) = (0, L, q_2)$. If $\delta(\mathtt{a}, q) = \mathsf{HALT}$, then the machine halts execution, and accepts. We always assume that there is at least one $(\mathtt{a}, q) \in \Gamma \times Q$ such that $\delta(\mathtt{a}, q) = \mathsf{HALT}$.

We now need to look at what it means if $\delta(\mathtt{a}, q) = (\mathsf{CHECK}_R, p)$. If indeed $\delta(\mathtt{a}, q) = (\mathsf{CHECK}_R, p)$, then the machine immediately checks – without moving the head's position – whether the tape to the right of the head's current position only contains blank symbols. If the right-side of the tape is blank, then the machine moves to state p. Otherwise, the machine stays in state q. Hence, if the right-side of the tape is not blank, then the machine continuously remains in the same state, and thus we enter an infinite loop.

Without going into details, the CHECK_R-instruction is used in-place of meta-symbols used to mark the start and end of the input word. See [31] for more discussion on the use of the behaviour of the CHECK_R-instruction.

Let $\mathtt{a}^\omega$ denote the one-sided infinite word $(t_i)_{i=1}^{\infty}$ where $t_i = \mathtt{a}$ for all $i \in \mathbb{N}_+$. Let $\mathcal{X} := (Q, q_1, \delta)$ be an extended Turing machine. A configuration C for $\mathcal{X}$ is a tuple of the form $(w_L, w_R, \mathtt{a}, q)$, where we have:

- $w_L, w_R \in \Gamma^* 0^\omega$ are used to denote the tape to the left and right of the head's current position respectively (w_L is read from right to left, starting with the cell immediately to the left of the head's current position),
- $\mathtt{a} \in \Gamma$ is the symbol currently under the head of $\mathcal{X}$, and
- $q \in Q$ is the current state.

For two configurations C, C' we use $C \vdash_{\mathcal{X}} C'$ to denote that if $\mathcal{X}$ is in configuration C, then it immediately enters C'. For an example, see Fig. 2.

We define $\mathsf{dom}(\mathcal{X})$ for an extended Turing machine $\mathcal{X}$, as the set of all words $w \in \Gamma^* \cdot 0^\omega$ where, if $\mathcal{X}$ starts in the initial configuration $(0^\omega, w, \mathtt{a}, q_1)$ for some $\mathtt{a} \in \Gamma$, then $\mathcal{X}$ halts after a finite number of steps.

Lemma 6 (Freydenberger [31]) *Consider the following decision problems for extended Turing machines:*

1. *Emptiness: Given an extended Turing machine $\mathcal{X}$, is $\mathsf{dom}(\mathcal{X})$ empty?*
2. *Finiteness: Given an extended Turing machine $\mathcal{X}$, is $\mathsf{dom}(\mathcal{X})$ finite?*

Then emptiness is not semi-decidable, and finiteness is neither semi-decidable, nor co-semi-decidable.

We interpret the left-hand and right-hand side of the tape as natural numbers. For an infinite sequence $t := (t_i)_{i=1}^{\infty}$ over Γ, let

$$\mathsf{e}(t) := \sum_{i=0}^{\infty} 2^i \mathsf{e}(t_i),$$

where $\mathsf{e}(x) = x$ for $x \in \{0, 1\}$. Therefore, $\mathsf{e}(w_L)$ and $\mathsf{e}(w_R)$ can be thought of as a binary number, where the cell closest to the head is the least significant bit. For

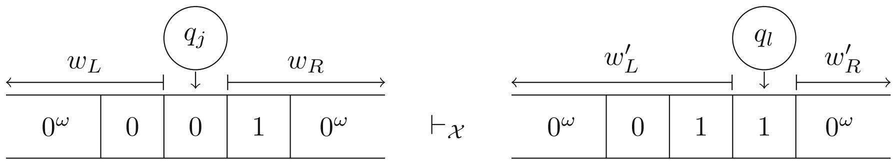

Fig. 2 This figure illustrates two configurations of some extended Turing machine $\mathcal{X}$. The configuration on the left shows $\mathcal{X}$ currently in state q_j with the head reading 0. If $\delta(0, q_j) = (1, R, q_l)$, then it follows that the configuration on the right is the immediate successor configuration. Note that $w'_L = 1 \cdot 0 \cdot 0^\omega$

example, if $w_L = 01101 \cdot 0^\omega$, then $\mathsf{e}(w_L) = 22$. Note that since there can only be a finite number of 1s on the left-hand side and the right-hand side of the tape, the function e is well defined.

For any configuration $C = (w_L, w_R, \mathtt{a}, q_i)$ of $\mathcal{X}$, we define an encoding function enc that encodes C over the alphabet $\{0, \#\}$ as follows:

$$\mathsf{enc}(w_L, w_R, \mathtt{a}, q_i) := 0^{\mathsf{e}(w_L)+1} \cdot \# \cdot 0^{\mathsf{e}(w_R)+1} \cdot \# \cdot 0^{\mathsf{e}(\mathtt{a})+1} \cdot \# \cdot 0^i.$$

If $(C_i)_{i=1}^n$ is a sequence of configurations for $\mathcal{X}$, we say that $(C_i)_{i=1}^n$ is an *accepting run* if C_1 is an initial configuration, C_n is a halting configuration, and $C_i \vdash_{\mathcal{X}} C_{i+1}$ for all $i \in [n-1]$.

Observation 1 (Freydenberger [31]) *Let $\mathcal{X} := (Q, q_1, \delta)$ be an extended Turing machine in the configuration $C = (w_L, w_R, \mathtt{a}, q_i)$, and $\delta(\mathtt{a}, q_i) = (\mathtt{b}, M, q_j)$, where $\mathtt{a}, \mathtt{b} \in \Gamma$ are tape symbols, $q_i, q_j \in Q$ are states, and $M \in \{L, R\}$. For the unique successor state $C' = (w'_L, w'_R, \mathtt{a}', q_j)$ where $C \vdash_{\mathcal{X}} C'$, we have that:*

$$\begin{array}{llll} \mathit{If}\, M = L: & \mathsf{e}(w'_L) = \mathsf{e}(w_L)\ \mathsf{div}\ 2, & \mathsf{e}(w'_R) = 2\mathsf{e}(w_R) + \mathtt{b}, & \mathtt{a}' = \mathsf{e}(w_L)\ \mathsf{mod}\ 2, \\ \mathit{if}\, M = R: & \mathsf{e}(w'_L) = 2\mathsf{e}(w_L) + \mathtt{b}, & \mathsf{e}(w'_R) = \mathsf{e}(w_R)\ \mathsf{div}\ 2, & \mathtt{a}' = \mathsf{e}(w_R)\ \mathsf{mod}\ 2, \end{array}$$

where div *denotes integer division, and* mod *denotes the modulo operation.*

For an extended Turing machine $\mathcal{X}$, let us define a language $\mathsf{VALC}(\mathcal{X}) \subseteq \Sigma^*$.

$$\mathsf{VALC}(\mathcal{X}) := \{\#\#\mathsf{enc}(C_1)\#\# \cdots \#\#\mathsf{enc}(C_n)\#\# \mid (C_i)_{i=1}^n \text{ is an accepting run}\}.$$

Thus, $\mathsf{VALC}(\mathcal{X})$ encodes the "computational history" of every accepting run of $\mathcal{X}$. Let us also define $\mathsf{INVALC}(\mathcal{X}) := \Sigma^* \setminus \mathsf{VALC}(\mathcal{X})$. The language $\mathsf{INVALC}(\mathcal{X})$ can be thought of as the language of computational histories of every erroneous run of $\mathcal{X}$. We distinguish between two types of errors that prohibits a word from being in $\mathsf{VALC}(\mathcal{X})$, and hence is in $\mathsf{INVALC}(\mathcal{X})$:

1. *Structural errors.* A word $w \in \Sigma^*$ contains a structural error if it does not encode any sequence of configurations that starts with a valid initial state for $\mathcal{X}$, and ends with a valid halting state for $\mathcal{X}$.
2. *Behavioural errors.* A word $w \in \Sigma^*$ contains a behavioural error if it encodes a sequence of configurations $(C_i)_{i=1}^n$ for some $n \geq 1$, but $C_i \vdash_{\mathcal{X}} C_{i+1}$ does not hold for some $i \in [n-1]$. These behavioural errors can further be distinguished between three types of errors:

   (a) *State errors*. The state in C_{i+1} is incorrect, or C_i is a halting configuration.
   (b) *Head errors*. The head in C_{i+1} reads the wrong symbol.
   (c) *Tape errors*. The tape in C_{i+1} does not follow from C_i.

From [31], we know that for a given extended Turing machine $\mathcal{X}$, the language $\mathsf{INVALC}(\mathcal{X})$ is regular if and only if $\mathsf{dom}(\mathcal{X})$ is finite. Since finiteness of $\mathcal{X}$ is undecidable (again, see [31]), it follows that given $\mathcal{X}$, it is undecidable to determine whether $\mathsf{INVALC}(\mathcal{X})$ is regular. Similarly, due to the fact that the emptiness problem for extended Turing machines is undecidable, determining whether $\mathsf{INVALC}(\mathcal{X}) = \Sigma^*$ is undecidable.

Recall Corollary 1, which states that the universality problem for FC-CQ is NP-complete. Somewhat surprisingly, when we add regular constraints, the universality problem becomes undecidable.

Theorem 6 *For* FC[REG]-CQ, *universality is not semi-decidable, and regularity is neither semi-decidable, nor co-semi-decidable.*

To prove Theorem 6, we give a construction that takes an extended Turing machine $\mathcal{X}$, and returns $\varphi \in$ FC[REG]-CQ such that $\mathcal{L}(\varphi) = \mathsf{INVALC}(\mathcal{X})$. The proof of Theorem 6 is given in Section 4.2.

Observing Corollary 1, we know that the universality problem for FC-CQs is NP-complete. Therefore, we cannot effectively construct a formula $\varphi \in$ FC-CQ such that $\mathcal{L}(\varphi) = \mathsf{INVALC}(\mathcal{X})$ for a given extended Turing machine $\mathcal{X}$ (otherwise, the emptiness problem for extended Turing machines would be NP-complete). However, using a similar proof idea to Theorem 6, we are able to conclude that the regularity problem for FC-CQ is undecidable.

Theorem 7 *The regularity problem for* FC-CQ *is neither semi-decidable, nor co-semi-decidable.*

In order to prove Theorems 6 and 7, we look at all possible errors that prohibit some $w \in \Sigma^*$ from being in $\mathsf{VALC}(\mathcal{X})$. Then, each of these errors can be encoded as an FC[REG]-CQ or an FC-CQ. However, due to the fact that FC[REG]-CQ and FC-CQ do not have disjunction, we require some encoding gadgets to simulate disjunction using concatenation. Referring back to Example 4, we can see a simple example of simulating disjunction in FC[REG]-CQ. The full proof is given in Section 4.3.

To prove Theorem 7, we show how to convert an extended Turing machine $\mathcal{X}$ into $\varphi \in$ FC-CQ, such that $\mathcal{L}(\varphi) = 0 \cdot \# \cdot 0 \cdot \#^3 \cdot \mathsf{INVALC}(\mathcal{X}) \cdot \#^3$. Thus:

Corollary 2 *The equivalence problem for* FC-CQ *is neither semi-decidable, nor co-semi-decidable.*

Proof From the proof of Theorem 7, we can construct $\varphi \in$ FC-CQ from a given extended Turing machine $\mathcal{X}$, such that $\mathcal{L}(\varphi) = 0 \cdot \# \cdot 0 \cdot \#^3 \cdot \mathsf{INVALC}(\mathcal{X}) \cdot \#^3$. Thus, determining $\mathcal{L}(\varphi) = \mathcal{L}(\psi)$ where $\psi := (\mathsf{w} \doteq 0 \cdot \# \cdot 0 \cdot \#^3 \cdot x \cdot \#^3)$ for some $x \in \Xi$ is neither semi-decidable, nor co-semi-decidable. □

While these undecidability results are themselves of interest, in Section 5, we consider their implications with regards to *minimization* and *non-recursive trade-offs*.

The proofs of Theorems 6 and 7 are given in Sections 4.2 and 4.3 respectively. These proofs are rather lengthy, therefore the reader may wish to skip to the consequences of these results given in Section 5.

4.2 Proof of Theorem 6

Lemma 7 *Given an extended Turing machine $\mathcal{X}$, one can effectively construct $\varphi \in$* FC[REG]-CQ *such that $\mathcal{L}(\varphi) = \mathsf{INVALC}(\mathcal{X})$.*

Proof Let $\mathcal{X}$ be an extended Turing machine. Let $\Sigma := \{\#, 0\}$. We construct φ to simulate a disjunction of errors, each of which prohibits $w \in \Sigma^*$ from being in $\mathsf{VALC}(\mathcal{X})$. Let

$$\varphi := (\mathsf{w} \doteq x_{\mathsf{error}} \cdot x_{\mathsf{tape}}) \wedge (x_{\mathsf{error}} \mathrel{\dot{\in}} \gamma_{\mathsf{error}}) \wedge \big(x_{\mathsf{tape}} \mathrel{\dot{\in}} (\gamma'_{\mathsf{struc}} \vee \varepsilon)\big) \wedge \psi_{\mathsf{tape}},$$

where all errors except for tape errors are "pushed" into the regular constraint γ_{error}. The regular constraint γ'_{struc} accepts all encoded runs that do not contain a structural error (we shall define this regular expression when handling structural errors). We shall ensure that there is some σ such that $\sigma \models \varphi$ where $\sigma(\mathsf{w}) = \varepsilon$. Thus, we can choose between a tape error, or some other error, by considering those substitutions where $\sigma \models \varphi$ and either $\sigma(x_{\mathsf{error}}) \neq \varepsilon$ or $\sigma(x_{\mathsf{tape}}) \neq \varepsilon$. If $\sigma(x_{\mathsf{error}}) \neq \varepsilon$ and $\sigma(x_{\mathsf{tape}}) \neq \varepsilon$ both hold, then we shall ensure that there is a tape error with the definition of the subformula ψ_{tape}.

As one can observe from the proof of Theorem 14 in [31], all errors except for tape errors can be encoded as a regular expression. We give a formal proof here for completeness sake, and due to the fact that this proof can be considered a primer for the proof of Theorem 7.

Before looking at the actual construction, we define a useful partition on $\Gamma \times Q$. Given an extended Turing machine $\mathcal{X} := (Q, q_1, \delta)$, we define the following sets:

$$\begin{aligned} S_{\mathsf{HALT}} :=& \{(\mathsf{a}, q_j) \in \Gamma \times Q \mid \delta(\mathsf{a}, q_j) = \mathsf{HALT}\}, \\ S_{\mathsf{L,R}} :=& \{(\mathsf{a}, q_j) \in \Gamma \times Q \mid \delta(\mathsf{a}, q_j) \in (\Gamma \times \{L, R\} \times Q)\}, \text{ and} \\ S_{\mathsf{CHECK}} :=& \{(\mathsf{a}, q_j) \in \Gamma \times Q \mid \delta(\mathsf{a}, q_j) \in (\mathsf{CHECK}_R \times Q)\}. \end{aligned}$$

Structural Errors To handle structural errors, we first construct a regular expression γ'_{struc} that accepts all words that do not have a structural error. Then, we use the compliment of this regular language to handle said structural errors.

First, let us consider the following regular expression:

$$\gamma_1 := \#\#(0^+\#0^+\#0(0 \vee \varepsilon)\#0^+\#\#)^+.$$

We have that $\mathcal{L}(\gamma_1)$ provides the "backbone" for a sequence of configurations. We ensure that the first configuration is a starting configuration using

$$\gamma_2 := \#\#0\#0^+\#0(0 \vee \varepsilon)\#0\#\# \cdot \Sigma^*.$$

Likewise, we ensure that the final configuration is a halting configuration. Let

$$\gamma_3 := \bigvee_{(\mathsf{a}, q_j) \in S_{\mathsf{HALT}}} \big(\Sigma^*\#0^{\mathsf{e}(\mathsf{a})+1}\#0^j\#\#\big).$$

Lastly, we only ensure valid states are used. Let

$$\gamma_4 := \#\#(0^+\#0^+\#0^+\#0 \cdot \bigvee_{m \in |Q|} (0^m) \cdot \#\#)^+.$$

Notice that due to the definition of enc, if we have $\#00^m\#\#$, then m encodes the state of the current configuration. Using the occurrence of ## as a way to parse out the individual elements of a configuration shall be commonly used throughout this proof and the proof of Theorem 7.

Let us now define γ'_{struc} such that

$$\mathcal{L}(\gamma'_{\mathsf{struc}}) = \mathcal{L}(\gamma_1) \cap \mathcal{L}(\gamma_2) \cap \mathcal{L}(\gamma_3) \cap \mathcal{L}(\gamma_4).$$

We have that γ'_{struc} defines those sequence of configurations that start with an initial configuration of $\mathcal{X}$, ends with a halting configuration of $\mathcal{X}$, and only uses states from $\mathcal{X}$. This regular expression can be constructed, due to the fact that regular languages are effectively closed under intersection. [3]

Then, let γ_{struc} such that $\mathcal{L}(\gamma_{\mathsf{struc}}) = \Sigma^* \setminus \mathcal{L}(\gamma'_{\mathsf{struc}})$. Thus, γ_{struc} handles all structural errors. Notice that $\varepsilon \in \mathcal{L}(\gamma_{\mathsf{struc}})$.

State Errors To handle state errors, we define γ_{state} as:

$$\gamma_{\mathsf{state}} := \gamma_{\mathsf{state}}^{\mathsf{halt}} \vee \gamma_{\mathsf{state}}^{L,R} \vee \gamma_{\mathsf{state}}^{\mathsf{check}},$$

where $\gamma_{\mathsf{state}}^{\mathsf{halt}}$, $\gamma_{\mathsf{state}}^{L,R}$, and $\gamma_{\mathsf{state}}^{\mathsf{check}}$ are to be defined. Each of these regular expressions deal with a certain type of instructions. For example, $\gamma_{\mathsf{state}}^{L,R}$ handles state errors for those instructions that cause the head to move.

Let

$$\gamma_{\mathsf{state}}^{\mathsf{halt}} := \bigvee_{(\mathtt{a},q_j)\in S_{\mathsf{halt}}} \left(\Sigma^*\#0^{\mathsf{e}(\mathtt{a})+1}\#0^j\#\#0\Sigma^*\right).$$

It follows that for all $w \in \mathcal{L}(\gamma_{\mathsf{state}}^{\mathsf{halt}})$, either w has a structural error, or w has a halting configuration that has a successor configuration.

To help with other state errors, we define a useful regular language $\gamma_{\neq j}$ for every $j \in Q$, such that $\mathcal{L}(\gamma_{\neq j}) := \{0^k \mid k \neq j\}$.

Next, we define $\gamma_{\mathsf{state}}^{L,R}$ as follows

$$\gamma_{\mathsf{state}}^{L,R} := \bigvee_{(\mathtt{a},q_j)\in S_{\mathsf{L,R}}} \left(\Sigma^*\#0^{\mathsf{e}(\mathtt{a})+1}\#0^j\#\#0^+\#0^+\#0^+\#\gamma_{\neq j'}\#\#\Sigma^*\right),$$

where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, M, q_{j'})$ for all $(\mathtt{a}, q_j) \in S_{\mathsf{L,R}}$. Thus, for any $w \in \mathcal{L}(\gamma_{\mathsf{state}}^{L,R})$, it follows that $w \in \mathsf{INVALC}(\mathcal{X})$ since either there is a structural error, or the state in one configuration does not follow from the head symbol-state pair in the previous configuration.

The last type of state error we need to handle are for those $(\mathtt{a}, q_j) \in \Gamma \times Q$ such that $\delta(\mathtt{a}, q_j) = (\mathsf{CHECK}_R, q_l)$. To that end, we define

$$\begin{aligned}\gamma_{\mathsf{state}}^{\mathtt{a},q_j} :=&(\Sigma^*\#0\#0^{\mathsf{e}(\mathtt{a})+1}\#0^j\#\#0^+\#0^+\#0^+\#\gamma_{\neq l}\#\#\Sigma^*)\vee\\ &(\Sigma^*\#00^+\#0^{\mathsf{e}(\mathtt{a})+1}\#0^j\#\#0^+\#0^+\#0^+\#\gamma_{\neq j}\#\#\Sigma^*).\end{aligned}$$

[3] Note that the regular expression γ'_{struc} is the same as the regular constraint placed on the variable x_{tape}.

Recall that if $\delta(\mathtt{a}, q_j) = (\mathsf{CHECK}_R, q_l)$, then we have two cases: If $w_R = 0^\omega$, then $\mathcal{X}$ moves into state q_l. Otherwise, we have that $w_R \neq 0^\omega$, and $\mathcal{X}$ remains in q_j which leads to an "infinite loop". Thus, $\gamma_{\mathsf{state}}^{\mathtt{a},q_j}$ handles CHECK_R instructions by moving to the incorrect state for the cases where $w_R = 0^\omega$ and $w_R \neq 0^\omega$.

Then, let

$$\gamma_{\mathsf{state}}^{\mathsf{check}} := \bigvee_{(\mathtt{a},q_j)\in S_{\mathsf{state}}^{\mathsf{check}}} \gamma_{\mathsf{state}}^{\mathtt{a},q_j}.$$

We have now considered every state error, and encoded these errors in one of three regular expressions ($\gamma_{\mathsf{state}}^{\mathsf{halt}}$, $\gamma_{\mathsf{state}}^{L,R}$, and $\gamma_{\mathsf{state}}^{\mathsf{check}}$). Since γ_{state} is a disjunction of these three regular expressions, we have handled these state errors.

Head Errors If $(w_L, w_R, \mathtt{a}, q_j) \vdash_{\mathcal{X}} (w'_L, w'_R, \mathtt{a}', q_l)$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, L, q_l)$, then from Observation 1, we know that $\mathtt{a}' = \mathsf{e}(w_L) \bmod 2$.

Therefore, for all $(\mathtt{a}, q_j)$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, L, q_l)$, we define

$$\begin{aligned}\gamma_{\mathsf{head}}^{\mathtt{a},q_j} :=& (\Sigma^*\#0(00)^*\#0^+\#0^{\mathsf{e}(\mathtt{a})+1}\#0^j\#\#0^+\#0^+\#00\#\Sigma^*)\vee\\ & (\Sigma^*\#00(00)^*\#0^+\#0^{\mathsf{e}(\mathtt{a})+1}\#0^j\#\#0^+\#0^+\#0\#\Sigma^*).\end{aligned}$$

Thus, for both parities of w_L, we have a parity error in the head symbol.

We do the analogous encoding for those $(\mathtt{a}, q_j)$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, R, q_l)$:

$$\begin{aligned}\gamma_{\mathsf{head}}^{\mathtt{a},q_j} :=& (\Sigma^*\#0(00)^*\#0^{\mathsf{e}(\mathtt{a})+1}\#0^j\#\#0^+\#0^+\#00\#\Sigma^*)\vee\\ & (\Sigma^*\#00(00)^*\#0^{\mathsf{e}(\mathtt{a})+1}\#0^j\#\#0^+\#0^+\#0\#\Sigma^*).\end{aligned}$$

This expression comes from considering Observation 1, and encoding the errors for the new head symbol for both parities of $\mathsf{e}(w_R)$.

To conclude encoding head errors, we consider the CHECK_R instruction. From the definition, CHECK_R instructions do not change the head. Thus, for every $(\mathtt{a}, q_j) \in \Gamma \times Q$ where $\delta(\mathtt{a}, q_j) = (\mathsf{CHECK}_R, q_l)$, we have

$$\gamma_{\mathsf{head}}^{\mathtt{a},q_j} := \Sigma^*\#0^{\mathsf{e}(\mathtt{a})+1}\#0^j\#\#0^+\#0^+\# \cdot \gamma_{\neq \mathsf{e}(\mathtt{a})+1} \cdot \#\Sigma^*.$$

Since the new head symbol is not the same as the previous one, we have handled head errors for the CHECK_R instruction. While we defined the regular expressions $\gamma_{\neq j}$ with state errors in mind, since $0^{\mathsf{e}(\mathtt{a})+1} \in \{0, 00\}$, we can reuse these regular expressions in $\gamma_{\mathtt{a},q_j}^{\mathsf{head}}$, as can be seen above.

Now, let us define

$$\gamma_{\mathsf{head}} := \bigvee_{(\mathtt{a},q_j)\in(S_{\mathsf{L,R}}\cup S_{\mathsf{CHECK}})} \gamma_{\mathsf{head}}^{\mathtt{a},q_j}.$$

It follows that $w \in \mathcal{L}(\gamma_{\mathsf{head}})$ if and only if w contains a head error or a structural error.

Now let us define

$$\gamma_{\mathsf{error}} := \gamma_{\mathsf{struc}} \vee \gamma_{\mathsf{state}} \vee \gamma_{\mathsf{head}}.$$

Therefore, γ_{error} contains all words that have a structural error, a state error, or a head error.

Tape Errors Recall that

$$\varphi := (\mathsf{w} \doteq x_{\mathsf{error}} \cdot x_{\mathsf{tape}}) \wedge (x_{\mathsf{error}} \mathrel{\dot\in} \gamma_{\mathsf{error}}) \wedge (x_{\mathsf{tape}} \mathrel{\dot\in} (\gamma'_{\mathsf{struc}} \vee \varepsilon)) \wedge \psi_{\mathsf{tape}},$$

where all errors except for tape errors are "pushed" into the regular constraint γ_{error}. Also recall that γ'_{struc} defines the language of all encoded runs that do not contain a structural error. To handle tape errors, we consider those substitutions σ where $\sigma \models \varphi$ and $\sigma(x_{\mathsf{tape}}) \neq \varepsilon$. Thus, $\sigma(x_{\mathsf{tape}}) \in \gamma'_{\mathsf{struc}}$.

We define ψ_{tape} as follows:

$$\begin{aligned}\psi_{\mathsf{tape}} := (x_{\mathsf{tape}} \doteq x_{\#}x_{\#}x_0x_{\#} \cdot x) \wedge (x_{\mathsf{tape}} \doteq \beta_1\beta_2\cdots\beta_\rho) \wedge (x \mathrel{\dot\in} (\varepsilon \vee \gamma_{\mathsf{structured}}))\\ \wedge(x_{\#} \mathrel{\dot\in} (\# \vee \varepsilon)) \wedge (x_{\#} \doteq x_{\#,1} \cdot x_{\#,2} \cdots x_{\#,\rho})\\ \wedge(x_0 \mathrel{\dot\in} (0 \vee \varepsilon)) \wedge (x_0 \doteq x_{0,1}x_{0,2} \cdots x_{0,\rho}) \wedge \psi_{+}\\ \wedge \bigwedge_{i=1}^{\rho}\Big(\big(x_{\#0,i} \mathrel{\dot\in} (\#0 \vee \varepsilon)\big) \wedge (x_{\#0,i} \doteq x_{\#,i} \cdot x_{0,i})\Big),\end{aligned}$$

where $\gamma_{\mathsf{structured}} := (0^+\#0(0 \vee \varepsilon)\#0^+\#\#) \cdot (0^+\#0^+\#0^+\#0^+\#\#)^*$, and $\beta_i \in \Xi^*$ for $i \in [\rho]$ are terminal-free patterns to be defined. The regular constraint $\gamma_{\mathsf{structured}}$ accepts a sequence of configurations without the ##0# prefix. It follows that since we are considering the case where $\sigma(x_{\mathsf{tape}}) \neq \varepsilon$, we have that $\sigma(x_{\mathsf{tape}}) \in \gamma'_{\mathsf{struc}}$. Consequently, $\sigma(x_{\mathsf{tape}}) \in (\#\#0\#0^+\#0(0 \vee \varepsilon)\#0\#\#) \cdot \Sigma^*$. Thus, it follows that $\sigma(x) \in \gamma_{\mathsf{structured}}$, $\sigma(x_{\#}) = \#$ and $\sigma(x_0) = 0$ must hold, otherwise $\sigma(x_{\mathsf{tape}}) \notin \gamma'_{\mathsf{struc}}$.

Since we know that $\sigma(x_{\#}) = \#$ and $\sigma(x_0) = 0$ for any σ where $\sigma \models \varphi$, we have that there is exactly one $i \in [\rho]$ such that $\sigma(x_{0,i}) = 0$ and $\sigma(x_{\#,i}) = \#$ due to the subformula:

$$(x_{\#} \mathrel{\dot\in} (\# \vee \varepsilon)) \wedge (x_{\#} \doteq x_{\#,1} \cdot x_{\#,2} \cdots x_{\#,\rho}) \wedge (x_0 \mathrel{\dot\in} (0 \vee \varepsilon)) \wedge (x_0 \doteq x_{0,1}x_{0,2} \cdots x_{0,\rho}).$$

For all $i' \in [\rho] \setminus \{i\}$, we have that $\sigma(x_{0,i'}) = \varepsilon$ and $\sigma(x_{\#,i'}) = \varepsilon$. Notice that it cannot hold that $\sigma(x_{\#,i}) = \#$ and $\sigma(x_{0,i'}) = 0$ where $i, i' \in [\rho]$ and $i \neq i'$ due to the previous observation along with the subformula

$$\bigwedge_{i=1}^{\rho}\big((x_{\#0,i} \mathrel{\dot\in} (\#0 \vee \varepsilon)) \wedge (x_{\#0,i} \doteq x_{\#,i} \cdot x_{0,i})\big).$$

If indeed $\sigma(x_{0,i}) = 0$ and $\sigma(x_{\#,i}) = \#$ for some substitution $\sigma \models \varphi$ and $i \in [\rho]$, then we call i the *selected error* for σ. Therefore, $\rho \in \mathbb{N}$ can be seen as the number of different patterns needed to cover all the tape errors.

The last thing to do before handling the tape errors is to define ψ_{+}. This subformula states that if i is the selected error, then certain variables must be replaced with 0^+.

Let

$$\psi_+ := \bigwedge_{i=1}^{\rho} \bigwedge_{r=1}^{3} \big((z_{i,r} \doteq x_{0,i} \cdot z'_{i,r}) \wedge (z_{i,r} \doteq z'_{i,r} \cdot x_{0,i}) \big).$$

It follows that if $i \in [\rho]$ is the selected error for σ, then $\sigma(z_{i,1}), \sigma(z_{i,2}), \sigma(z_{i,3}) \in 0^+$. This is from a combinatorics on words observation: If $vu = uv$ for $u, v \in \Sigma^*$, then there is some $z \in \Sigma^*$ and $k_1, k_2 \in \mathbb{N}$ such that $u = z^{k_1}$ and $v = z^{k_2}$ (for example, see Proposition 1.3.2 in Lothaire [16]). Assume that $\sigma(x_{0,i}) = 0$, then $\sigma(z_{i,1}) = \sigma(z'_{i,1})\sigma(x_{0,i}) = \sigma(x_{0,i})\sigma(z'_{i,1})$ and thus $\sigma(z'_{i,1}) \in 0^*$. It therefore follows that $\sigma(z_{i,1}) \in 0^+$ holds. Notice that if i is not the selected error, then $\sigma(z_{i,r}) = \sigma(z_{i,r'})$ for $r \in [3]$, and $\sigma(z_{i,r}) = \varepsilon$ can hold.

We are now ready to define the patterns that encode the tape errors. The patterns β_i for $i \in [\rho]$ are used to encode the actual tape errors. For $i \neq j$ where $i, j \in [\rho]$, we have $\mathsf{vars}(\beta_i) \cap \mathsf{vars}(\beta_j) = \emptyset$. While defining each pattern β_i for $i \in [\rho]$, we assume that i is the selected error. Therefore, we can assume that $x_{\#,i} = \#$, $x_{0,i} = 0$, and $z_{i,1}, z_{i,2}, z_{i,3} \in \mathcal{L}(0^+)$.

A *tape error* describes where we have two configurations $C = (w_L, w_R, \mathtt{a}, q_j)$ and $C' = (w'_L, w'_R, \mathtt{a}, q_j)$, but w'_L or w'_R is not what is expected. We look at each type of instruction, and encode a tape error for each.

The CHECK_R-instruction: As it is the simplest to encode, due to the fact that $w_L = w'_L$ and $w_R = w'_R$ should both hold, we start with the CHECK_R-instruction.

First, we express an error by considering when $\mathsf{e}(w_L) < \mathsf{e}(w'_L)$. For every $(\mathtt{a}, q_j) \in S_{\mathsf{CHECK}}$, we have a unique $i \in [\rho]$ such that[4]

$$\beta_i := y_i \cdot x_{\#,i} \cdot \underbrace{z_{i,1}}_{\mathsf{e}(w_L)+1} \cdot x_{\#,i} \cdot z_{i,2} \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e}(\mathtt{a})+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2 \cdot \underbrace{z_{i,1} z_{i,3}}_{\mathsf{e}(w'_L)+1} \cdot y'_i.$$

As $z_{i,3} \in \mathcal{L}(0^+)$, we have that $|0^{\mathsf{e}(w_L)}| < |0^{\mathsf{e}(w'_L)}|$ and due to the fact that CHECK_R does not change the tape, this encodes an error.

Next, we deal with when $\mathsf{e}(w_L) > \mathsf{e}(w'_L)$. For this case, there is a unique $i \in [\rho]$ for every $(\mathtt{a}, q_j) \in S_{\mathsf{CHECK}}$ such that

$$\beta_i := y_i \cdot \underbrace{z_{i,1} z_{i,2}}_{\mathsf{e}(w_L)+1} \cdot x_{\#,i} \cdot z_{i,3} \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e}(\mathtt{a})+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2 \cdot \underbrace{z_{i,1}}_{\mathsf{e}(w'_L)+1} \cdot x_{\#,i} \cdot y'_i.$$

The above pattern hence encodes $\mathsf{e}(w_L) > \mathsf{e}(w'_L)$.

[4] Note that the underbraces in the following patterns are used for intuition purposes. For example, the variable $z_{i,1}$ represents $0^{\mathsf{e}(w_L)+1}$. However, to try and avoid notational clutter, we simply use $\mathsf{e}(w_L) + 1$ instead.

As with w_L, we also have the analogous case where $w_R \neq w'_R$. To that end, there are unique values $i, i' \in [\rho]$ for every $(\mathsf{a}, q_j) \in S_{\mathsf{CHECK}}$ such that

$$\begin{aligned}\beta_i :=& y_i x_{\#,i} \cdot \overbrace{z_{i,1}}^{\mathsf{e}(w_R)+1} \cdot x_{\#,i} \cdot \gamma_i \cdot x_{\#,i} \cdot \overbrace{z_{i,1} \cdot z_{i,3}}^{\mathsf{e}(w'_R)+1} \cdot x_{\#,i} y'_i,\\ \beta_{i'} :=& y_{i'} x_{\#,i'} \cdot \underbrace{z_{i',1} z_{i',2}}_{\mathsf{e}(w_R)+1} \cdot x_{\#,i'} \cdot \gamma_{i'} \cdot x_{\#,i'} \cdot \underbrace{z_{i',1}}_{\mathsf{e}(w'_R)+1} \cdot x_{\#,i'} y'_{i'},\end{aligned}$$

where $\gamma_k := (x_{0,k})^{\mathsf{e}(\mathsf{a})+1} \cdot x_{\#,k} \cdot (x_{0,k})^j \cdot (x_{\#,k})^2 \cdot z_{k,2}$ for $k \in \{i, i'\}$. The reasoning behind these errors (when $w_R \neq w'_R$) follows from the case where $w_L \neq w'_L$. That is, we deal with the two case where $\mathsf{e}(w_R) < \mathsf{e}(w'_R)$, and where $\mathsf{e}(w_R) > \mathsf{e}(w'_R)$ with β_i and $\beta_{i'}$ respectively.

This concludes our look at tape errors for CHECK_R instructions. For intuition, for every symbol-state pair that leads to the CHECK_R instruction, we have four patterns. Two patterns deal with the case where $\mathsf{e}(w_L) \neq \mathsf{e}(w'_L)$, and two patterns deal with the case where $\mathsf{e}(w_R) \neq \mathsf{e}(w'_R)$.

Head-movement instructions: To deal with the instructions that cause the head to move, we partition $S_{\mathsf{L,R}}$ further. Let

$$\begin{aligned}S_{\mathsf{L}} :=& \{(\mathsf{a}, q_j) \in \Gamma \times Q \mid \delta(\mathsf{a}, q_j) \in (\Gamma \times \{L\} \times Q)\}, \text{ and}\\ S_{\mathsf{R}} :=& \{(\mathsf{a}, q_j) \in \Gamma \times Q \mid \delta(\mathsf{a}, q_j) \in (\Gamma \times \{R\} \times Q)\}.\end{aligned}$$

Let us now consider $\delta(\mathsf{a}, q_j) = (\mathsf{b}, L, q_l)$. We know from Observation 1 that $\mathsf{e}(w'_L) = \mathsf{e}(w_L) \operatorname{div} 2$. First, we consider when $\mathsf{e}(w'_L)$ is too large. This is split into two very similar cases (depending on the parity of w_L). For each $(\mathsf{a}, q_j) \in S_{\mathsf{L}}$, we have unique values $i, i', i'' \in [\rho]$ such that

$$\begin{aligned}\beta_i :=& y_i \cdot x_{\#,i} \overbrace{(x_{0,i})^2 \cdot (z_{i,1})^2}^{\mathsf{e}(w_L)+1} \cdot \gamma_i \cdot \overbrace{x_{0,i} z_{i,1} z_{i,3}}^{\mathsf{e}(w'_L)+1} \cdot y'_i,\\ \beta_{i'} :=& y_{i'} \cdot x_{\#,i'} \underbrace{x_{0,i'} \cdot (z_{i',1})^2}_{\mathsf{e}(w_L)+1} \cdot \gamma_{i'} \cdot \underbrace{x_{0,i'} z_{i',1} z_{i',3}}_{\mathsf{e}(w'_L)+1} \cdot y'_{i'}.\end{aligned}$$

We also need to handle the special case where $\mathsf{e}(w_L) = 0$:

$$\beta_{i''} := y''_i \cdot x_{\#,i''} \underbrace{x_{0,i''}}_{\mathsf{e}(w_L)+1} \cdot \gamma_{i''} \cdot \underbrace{x_{0,i''} \cdot z_{i'',1}}_{\mathsf{e}(w'_L)+1} \cdot y'_{i''}$$

where γ_k for $k \in \{i, i', i''\}$ is defined as

$$\gamma_k := x_{\#,k} \cdot z_{k,2} \cdot x_{\#,k} \cdot (x_{0,k})^{\mathsf{e}(\mathsf{a}+1)} \cdot x_{\#,k} \cdot (x_{0,k})^j \cdot (x_{\#,k})^2.$$

Since $z'_{k,r} \in 0^+$ for $k \in \{i, i', i''\}$ and $r \in [3]$, we have that $\mathsf{e}(w'_L) > \mathsf{e}(w_L) \operatorname{div} 2$.

To handle the case where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, L, q_l)$ and w'_L is too small, we instead encode this as w_L being too large. Since $\mathsf{e}(w'_L) = \mathsf{e}(w_L) \operatorname{div} 2$, we have that $\mathsf{e}(w'_L)$ is too small if $\mathsf{e}(w_L) > 2\mathsf{e}(w'_L)+1$. Thus, for each $(\mathtt{a}, q_j) \in S_{\mathsf{L}}$, we have a unique $i \in [\rho]$ such that

$$\beta_i := y_i \cdot \underbrace{(z_{i,1})^2}_{\mathsf{e}(w_L)+1} \cdot x_{\#,i} \cdot z_{i,2} \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e}(\mathtt{a}+1)} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2 \cdot \underbrace{z_{i,1}}_{\mathsf{e}(w'_L)+1} \cdot x_{\#,i} \cdot y'_i.$$

Note that in β_i the right-most occurrence of $z_{i,1}$ encodes $0 \cdot 0^{\mathsf{e}(w'_L)}$. Thus, the occurrence of $(z_{i,1})^2$ encodes $(00^{\mathsf{e}(w'_L)})^2$. Hence, $(z_{i,1})^2$ encodes $2\mathsf{e}(w'_L) + 2$.

Now, let us deal with errors on the right-hand side of the tape. This is more complicated since we have to deal with what is being written to the tape.

First, let us deal with w'_R being too large. We know from Observation 1 that if $\delta(\mathtt{a}, q_j) = (\mathtt{b}, L, q_l)$, then $\mathsf{e}(w'_R) = 2\mathsf{e}(w_R) + \mathtt{b}$. Hence, for each $(\mathtt{a}, q_j) \in S_{\mathsf{L}}$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, L, q_l)$, we have unique values $i, i' \in [\rho]$ such that

$$\beta_i := y_i \cdot x_{\#,i} \cdot \underbrace{x_{0,i} z_{i,1}}_{\mathsf{e}(w_R)+1} \cdot x_{\#,i} \cdot \gamma_i \cdot x_{\#,i} \cdot \underbrace{(x_{0,i})^{\mathsf{e}(\mathtt{b})+1} (z_{i,1})^2 z_{i,2}}_{\mathsf{e}(w'_R)+1} \cdot y'_i.$$

where

$$\gamma_i := (x_{0,i})^{\mathsf{e}(\mathtt{a})+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2 \cdot z_{i,3}.$$

Therefore, β_i defines the case where $0 \cdot 0^{\mathsf{e}(w'_R)} = 0 \cdot 0^{\mathsf{e}(\mathtt{b})} \cdot 0^{2\mathsf{e}(w_R)} \cdot 0^m$, for some $m \geq 1$. Therefore, if i is the selected error, then $w \in \mathsf{INVALC}(\mathcal{X})$.

To deal with $w_R = 0$, we define

$$\beta_{i'} := y_{i'} \cdot x_{\#,i'} \cdot x_{0,i'} \cdot x_{\#,i'} \cdot (x_{0,i'})^{\mathsf{e}(\mathtt{a})+1} \cdot x_{\#,i'} \cdot (x_{0,i'})^j \cdot (x_{\#,i'})^2 \cdot z_{i',3} \cdot x_{\#,i'} \cdot (x_{0,i'})^{\mathsf{e}(\mathtt{b})+2} \cdot y'_{i'},$$

Next, let us consider when w'_R is too small. First, we just deal with the case where $\mathsf{e}(w'_R)$ is of the wrong parity. To that end, for each $(\mathtt{a}, q_j) \in S_{\mathsf{L}}$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, L, q_l)$, we have unique values $i, i' \in [\rho]$ such that

$$\begin{aligned} \beta_i &:= \gamma_i \cdot x_{\#,i} \cdot x_{0,i} (z_{i,2})^2 (x_{0,i})^{1-\mathsf{e}(\mathtt{b})} \cdot x_{\#,i} \cdot y'_i, \\ \beta_{i'} &:= \gamma_{i'} \cdot x_{\#,i'} \cdot x_{0,i'} (x_{0,i'})^{1-\mathsf{e}(\mathtt{b})} \cdot x_{\#,i'} \cdot y'_{i'}, \end{aligned}$$

where γ_k for $k \in \{i, i'\}$ is defined as:

$$\gamma_k := y_k \cdot x_{\#,k} \cdot (x_{0,k})^{\mathsf{e}(\mathtt{a})+1} \cdot x_{\#,k} \cdot (x_{0,k})^j \cdot (x_{\#,k})^2 \cdot z_{k,1}.$$

We have that β_i and $\beta_{i'}$ together encode

$$\Sigma^* \cdot \#0^{\mathsf{e}(\mathtt{a})+1}\#0^j\#\#0^+(00)^*0^{1-\mathsf{e}(\mathtt{b})}\#\Sigma^*.$$

Since from Observation 1, $\mathsf{e}(w'_R) \bmod 2 = \mathtt{b}$ should hold, if i or i' is the selected error, then we have a parity error of the type just encoded.

We now handle the case where w'_R is too small and is of the correct parity. Again, we encode w'_R being too small as w_R being too large. For each $(\mathtt{a}, q_j) \in S_\mathsf{L}$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, L, q_l)$, we have unique $i, i' \in [\rho]$ where

$$\beta_i := y_i \cdot \overbrace{x_{0,i} z_{i,1} z_{i,2}}^{\mathsf{e}(w_R)+1} \cdot x_{\#,i} \cdot \gamma_k \cdot x_{\#,i} \cdot \overbrace{x_{0,i}(z_{i,1})^2 (x_{0,i})^{\mathsf{e}(\mathtt{b})}}^{\mathsf{e}(w'_R)+1} \cdot x_{\#,i} y'_i,$$
$$\beta_{i'} := y_{i'} \cdot \underbrace{x_{0,i'} z_{i',2}}_{\mathsf{e}(w_R)+1} \cdot x_{\#,i'} \cdot \gamma_k \cdot x_{\#,i'} \cdot \underbrace{x_{0,i'}(x_{0,i'})^{\mathsf{e}(\mathtt{b})}}_{\mathsf{e}(w'_R)+1} \cdot x_{\#,i'} y'_{i'},$$

where for $k \in \{i, i'\}$ the subpattern γ_k is defined as

$$\gamma_k := (x_{0,k})^{\mathsf{e}(\mathtt{a})+1} \cdot x_{\#,k} \cdot (x_{0,k})^j \cdot (x_{\#,k})^2 \cdot z''_k.$$

Thus β_i and $\beta_{i'}$ together, encode

$$\Sigma^* 0 0^m 0^n \# 0^{\mathsf{e}(\mathtt{a})+1} \# 0^j \#\# 0^+ \# 0 0^{2m} 0^{\mathsf{e}(\mathtt{b})} \# \Sigma^*,$$

where $m \geq 0$, and $n \geq 1$. It is clear from the above representation of β_i and $\beta_{i'}$, we have that $\mathsf{e}(w'_R) < 2\mathsf{e}(w_R) + b$. Thus, we have handled this type of error.

The final type of error that we need to encode are for those $(\mathtt{a}, q_j) \in \Gamma \times Q$ such that $\delta(\mathtt{a}, q_j) = (\mathtt{b}, R, q_l)$. However, recalling Observation 1, this case is symmetrical to when $\delta(\mathtt{a}, q_j) = (\mathtt{b}, L, q_l)$, where the roles for w_L, w'_L and w_R, w'_R are reversed. Thus, it is clear that we can handle the tape errors for $\delta(\mathtt{a}, q_j) = (\mathtt{b}, R, q_l)$ the same way we handled the tape errors for $\delta(\mathtt{a}, q_j) = (\mathtt{b}, L, q_l)$, with the minor necessary changes.

Correctness Now, we show that $w \in \mathsf{INVALC}(\mathcal{X})$ if and only if $w \in \mathcal{L}(\varphi)$, where φ is as defined above.

Only if direction: For each $w \in \mathsf{INVALC}(\mathcal{X})$, there is at least one error that prohibits $w \in \mathsf{VALC}(\mathcal{X})$ from holding. We have defined a regular expression γ_{error} such that $\mathcal{L}(\gamma_{\mathsf{error}})$ handles all but the tape errors. Therefore, for any $w \in \mathsf{INVALC}(\mathcal{X})$, if w contains a non-tape error, then $w \in \mathcal{L}(\varphi)$.

If $w \in \Sigma^*$ only contains a tape error, then for some $i \in [\rho]$, there is some pattern β_i that we have defined such that for some substitution $\sigma \models \varphi$, we have that $\sigma(\mathsf{w}) = \sigma(\beta_i) = w$. Note that $\sigma(\beta_{i'}) = \varepsilon$ can hold for all $i' \in [\rho] \setminus \{i\}$ due to the fact that we know $\sigma(x_{\#,i'}) = \sigma(x_{0,i'}) = \varepsilon$ whenever i' is not the selected error, and all other variables in $\beta_{i'}$ do not have regular constraints.

If direction: Let $w \in \mathcal{L}(\varphi)$ and let σ be a substitution such that $\sigma \models \varphi$ and $\sigma(\mathsf{w}) = w$. If $\sigma(x_{\mathsf{tape}}) = \varepsilon$, then $w \in \gamma_{\mathsf{error}}$ must hold. Therefore, $w \in \mathsf{INVALC}(\mathcal{X})$. If $\sigma(x_{\mathsf{tape}}) \neq \varepsilon$, then $\sigma(x_{\mathsf{tape}}) \in \gamma'_{\mathsf{struc}}$ must hold. However, if this is the case, then there is some selected error $i \in [\rho]$ for σ and therefore:

- $\sigma(x_{0,i}) = 0$,

- $\sigma(x_{\#,i}) = \#$, and
- $\sigma(z_{i,r}) \in 0^+$ for $r \in \{1, 2, 3\}$.

Since for every $i \in [\rho]$ we have that β_i uses the variables $x_{0,i}$, $x_{\#,i}$ and $z_{i,r}$ for $r \in [3]$ to encode a tape error, and these variables are not mapped to the empty word, we have that w has a tape error. Therefore, $w \in \mathsf{INVALC}(\mathcal{X})$. Note that if $\sigma(x_{\mathsf{error}}) \neq \varepsilon$, then $\sigma(\mathsf{w})$ must contain a tape error due to the fact that the above reasoning still holds no matter what x_{error} is substituted with.

To conclude this proof, notice that if $\sigma \models \varphi$ where $\sigma(\mathsf{w}) = w$, then $w \in \mathcal{L}(\varphi)$. Therefore, we have shown that $w \in \mathsf{INVALC}(\mathcal{X})$ if and only if $w \in \mathcal{L}(\varphi)$, in turn showing that $\mathcal{L}(\varphi) = \mathsf{INVALC}(\mathcal{X})$. □

Actual Proof of Theorem 6

Theorem 6 *For* FC[REG]-CQ*, universality is not semi-decidable, and regularity is neither semi-decidable, nor co-semi-decidable.*

Proof From Theorem 7 we know that given an extended Turing machine $\mathcal{X}$, one can effectively construct $\varphi \in$ FC[REG]-CQ such that $\mathcal{L}(\varphi) = \mathsf{INVALC}(\mathcal{X})$. Therefore, if universality for FC[REG]-CQ were semi-decidable, then emptiness for extended Turing machines would be semi-decidable, which is a contradiction. If regularity for FC[REG]-CQ were semi-decidable or co-semi-decidable, then finiteness for Turing machines would be semi-decidable or co-semi-decidable. □

4.3 Proof of Theorem 7

Theorem 7 *The regularity problem for* FC-CQ *is neither semi-decidable, nor co-semi-decidable.*

Proof Let $\mathcal{X} := (Q, q_1, \delta)$ be an extended Turing machine, and let $\Sigma := \{0, \#\}$. From $\mathcal{X}$, we construct $\varphi \in$ FC-CQ such that $\mathcal{L}(\varphi)$ is regular if and only if $\mathsf{INVALC}(\mathcal{X})$ is regular. More formally, we define φ such that:

$$\mathcal{L}(\varphi) = \{0 \cdot \# \cdot 0 \cdot \#^3 \cdot w \cdot \#^3 \mid w \in \mathsf{INVALC}(\mathcal{X})\}.$$

In other words, we have fixed words $w_1, w_2 \in \{0, \#\}^+$ such that for each extended Turing machine $\mathcal{X}$, there exists $\varphi \in$ FC-CQ such that $\mathcal{L}(\varphi) = w_1 \cdot \mathsf{INVALC}(\mathcal{X}) \cdot w_2$, where $w_1 = 0\#0\#^3$ and $w_2 = \#^3$.

Instead of first defining the formula φ, and then proving correctness, we take a different approach and define a set of regularly typed patterns $\{(\beta_i, T_i) \mid i \in [\mu]\}$, where $\mu \in \mathbb{N}$ depends on $\mathcal{X}$ and for each $i, j \in [\mu]$ where $i \neq j$, we have that $\mathsf{vars}(\beta_i) \cap \mathsf{vars}(\beta_j) = \emptyset$. This set is defined such that $\cup_{i=1}^{\mu} \mathcal{L}(\beta_i, T_i) = \mathsf{INVALC}(\mathcal{X})$, where T_i is the typing function that maps variables to regular languages. We assume T_i is defined as follows:

- $T_i(x_{\mathtt{a},i}) := \{\mathtt{a}\}$ for all $\mathtt{a} \in \Sigma$,
- $T_i(y_i) := \Sigma^*$ and $T_i(y_i') := \Sigma^*$, and
- $T_i(z_{i,r}) := 0^+$ for each $r \in [4]$.

Each β_i is defined only using variables shown above. Then, the definitions of the patterns used to define the errors are somewhat similar to the tape errors in the proof of Lemma 7. We note that we use patterns such as $x_{c,i}$ for $c \in \Sigma$ rather than terminal symbols due to the fact that they may also need to be the emptyword. This shall be used when we eventually define the formula that encodes $\mathsf{INVALC}(\mathcal{X})$.

As with many proofs of this sort, we use a finite number of "rules" for a word $w \in \Sigma^*$ to contain an error, and thus $w \in \mathsf{INVALC}(\mathcal{X})$. Each rule is encoded as some (β_i, T_i). The encoding of the rules follows closely to the proof of Theorem 14 in [31].

We partition $[\mu]$ into sets of contiguous numbers for different types of errors.

1. $E_{\mathsf{struc}} := \{1, 2, \dots, e_1\}$ denote the structural errors,
2. $E_{\mathsf{state}} := \{e_1 + 1, e_1 + 2, \dots, e_2\}$ denotes the state errors,
3. $E_{\mathsf{head}} := \{e_2 + 1, e_2 + 2, \dots, e_3\}$ denotes the head errors, and
4. $E_{\mathsf{tape}} := \{e_3 + 1, e_3 + 2, \dots, \mu\}$ denotes the tape errors.

For example, for every $i \in E_{\mathsf{struc}}$, we have some specific structural error that causes a word to be in $\mathsf{INVALC}(\mathcal{X})$. The error associated to i is encoded in (β_i, T_i).

Structural Errors

First, we define β_i for $i \in [7]$:

$$\begin{aligned}
\beta_1 &:= \varepsilon,\\
\beta_2 &:= x_{0,2} \cdot y_2,\\
\beta_3 &:= y_3 \cdot x_{0,3},\\
\beta_4 &:= x_{\#,4} \cdot x_{0,4} \cdot y_4,\\
\beta_5 &:= y_5 \cdot x_{0,5} \cdot x_{\#,5},\\
\beta_6 &:= x_{\#,6},\\
\beta_7 &:= x_{\#,7} \cdot x_{\#,7}.
\end{aligned}$$

Up to this point, if $w \notin \mathcal{L}(\beta_i, T_i)$ for all $i \in [7]$, then $w = u\#\#$ and $w = \#\#v$ for some $u, v \in \Sigma^*$.

We now define β_8 to β_{12}:

$$\begin{aligned}
\beta_8 &:= y_8 \cdot (x_{\#,8})^3 \cdot y_8,\\
\beta_9 &:= y_9 \cdot x_{\#,9} \cdot x_{\#,9} \cdot z_{9,1} \cdot x_{\#,9} \cdot x_{\#,9} \cdot y_9,\\
\beta_{10} &:= y_{10} \cdot x_{\#,10} \cdot x_{\#,10} \cdot z_{10,1} \cdot x_{\#,10} \cdot z_{10,2} \cdot x_{\#,10} \cdot x_{\#,10} \cdot y_{10},\\
\beta_{11} &:= y_{11} \cdot x_{\#,11} \cdot x_{\#,11} \cdot z_{11,1} \cdot x_{\#,11} \cdot z_{11,2} \cdot x_{\#,11} \cdot z_{11,3} \cdot x_{\#,11} \cdot x_{\#,11} \cdot y_{11},\\
\beta_{12} &:= y_{12} \cdot x_{0,12} \cdot x_{\#,12} \cdot z_{12,1} \cdot x_{\#,12} \cdot z_{12,2} \cdot x_{\#,12} \cdot z_{12,3} \cdot x_{\#,12} \cdot x_{0,12} \cdot y_{12}.
\end{aligned}$$

After having dealt with the above cases, we have that if $w \notin \mathcal{L}(\beta_i, T_i)$ for all $i \in [12]$, then

$$w \in \mathcal{L}(\#\#(0^+\#0^+\#0^+\#0^+\#\#)^+).$$

Next, we deal with the case where the state does not encode a state from $\{q_1, \dots, q_k\}$ where $k \geq 1$. Notice that for some word $w \in \mathsf{VALC}(\mathcal{X})$, we have that if $\# \cdot 0^m \cdot \#\#$ is a factor of w, then m must be an encoding of a state. If $m > k$, then this configuration uses a state that is not in the state set of $\mathcal{X}$. We now define β_{13}:

$$\beta_{13} := \underbrace{y_{13}}_{\Sigma^*} \cdot \underbrace{x_{\#,13}}_{\#} \cdot \underbrace{(x_{0,13})^k \cdot z_{13,1}}_{0^m} \cdot \underbrace{(x_{\#,13})^2}_{\#\#} \cdot \underbrace{y'_{13}}_{\Sigma^*} .$$

This patterns deals with the case where there is a state q encoded in a configuration, yet q is not in the set of states of $\mathcal{X}$. This is due to the fact that $T_{13}(z_{13,1}) \in 0^+$, and therefore $(x_{0,13})^k \cdot z_{13,1} = 0^m$ for some $m > |Q|$.

Now, we deal with the error that the symbol the head is currently reading is not valid. For this, we define β_{14}:

$$\beta_{14} := y_{14} \cdot (x_{0,14})^3 \cdot x_{\#,14} \cdot z_{14,1} \cdot (x_{\#,14})^2 \cdot y'_{14}.$$

Thus, if $w \in (\beta_{14}, T_{14})$, then $w \in \mathcal{L}(\Sigma^* \cdot 000 \cdot \# \cdot 0^+ \cdot \#\# \cdot \Sigma^*)$. Therefore, the head is reading a symbol that is not part of the tape alphabet.

Next, we look at when the initial configuration is not correct: To that end, we define β_{15} and β_{16}:

$$\begin{aligned}\beta_{15} &:= (x_{\#,15})^2 \cdot (x_{0,15})^2 \cdot y_{15},\\ \beta_{16} &:= (x_{\#,16})^2 \cdot z_{16,1} \cdot x_{\#,16} \cdot z_{16,2} \cdot x_{\#,16} \cdot z_{16,3} \cdot x_{\#,16} \cdot (x_{0,16})^2 \cdot y_{16}.\end{aligned}$$

The above patterns deals with those $w \in \Sigma^*$ that are of the form:

$$\begin{aligned}&\#\#00\Sigma^*,\\ &\#\#0^+\#0^+\#0^+\#00\Sigma^*.\end{aligned}$$

These cases encode the possible errors with the initial configuration. This is because valid initial configurations are of the form $\#\#0\#0^+\#0^{\mathsf{e(a)}+1}\#0\#\#$, for some $\mathsf{a} \in \Gamma$. Thus, if $\mathsf{e}(w_L) \neq 0$, then we have an error (handled by β_{15}), and if the state is not q_1, then we have an error (handled by β_{16}). We have already handled the case where the head symbol is incorrect with β_{14}.

To conclude the encoding of structural errors, we look at when the final configuration is incorrect. For this, we consider $i \in E_{\mathsf{struc}} \setminus [16]$, each of which are uniquely mapped to some $(\mathsf{a}, q_j) \in \Gamma \times Q$ such that $\delta(\mathsf{a}, q_j) \neq \mathsf{HALT}$. For each $i \in E_{\mathsf{struc}} \setminus [16]$ we construct a pattern β_i as follows:

$$\beta_i := y_i \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e(a)}+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2.$$

Thus, for any $i \in E_{\mathsf{struc}} \setminus [16]$, if $w \in (\beta_i, T_i)$, then $w \in \mathcal{L}(\Sigma^*\#0^{\mathsf{e(a)}+1}\#0^j\#\#)$ where if $\mathcal{X}$ is in state q_j and reads the symbol a, then $\mathcal{X}$ does not halt. That is, the run finishes on a configuration that should not halt.

Thus, if $w \notin \mathcal{L}(\beta_i, T_i)$ for all $i \in E_{\mathsf{struc}}$, then:

- $w \in \mathcal{L}(\#\#(0^+\#0^+\#0(0 \vee \varepsilon)\#0^j\#\#)^+)$ for $j \leq |Q|$,

- w starts with a valid initial configuration, and
- w ends with a valid halting configuration.

Consequently, if $w \in \mathcal{L}(\beta_i, T_i)$ for some $i \in E_{\mathsf{struc}}$, then w contains a structural error. This concludes our encoding of structural errors.

Behavioural Errors Consider two possible configurations $C = (w_L, w_R, \mathsf{a}, q_j)$ and $C' = (w'_L, w'_R, \mathsf{a}', q_l)$. We have three types of behavioural errors such that $C \vdash_{\mathcal{X}} C'$ does not hold, and these error types need to be accounted for.

1. State errors. $\delta(\mathsf{a}, q_j) = \mathsf{HALT}$ or the encoding of state q_l is incorrect,
2. Head errors. The encoding of the head symbol a' is incorrect, and
3. Tape errors. The left-side of the tape w'_L or the right-side of the tape w'_R contains an error.

For each of these errors we consider each type of instruction, and consider an error.

Recall the following partition of $[\mu] \setminus E_{\mathsf{struc}}$:

- $E_{\mathsf{state}} := \{e_1 + 1, e_1 + 2, \ldots, e_2\}$ denotes the state errors,
- $E_{\mathsf{head}} := \{e_2 + 1, e_2 + 2, \ldots, e_3\}$ denotes the head errors, and
- $E_{\mathsf{tape}} := \{e_3 + 1, e_3 + 2, \ldots, \mu\}$ denotes the tape errors.

Each $i \in E_{\mathsf{state}}$ is mapped to a certain state error encoded with β_i. The analogous holds for E_{head} and E_{tape}.

Before looking at encoding the errors, we define a useful partition on $\Gamma \times Q$. Given an extended Turing machine $\mathcal{X} := (Q, q_1, \delta)$, we define the following sets:

$$\begin{aligned} S_{\mathsf{HALT}} :=& \{(\mathsf{a}, q_j) \in \Gamma \times Q \mid \delta(\mathsf{a}, q_j) = \mathsf{HALT}\}, \\ S_{\mathsf{L,R}} :=& \{(\mathsf{a}, q_j) \in \Gamma \times Q \mid \delta(\mathsf{a}, q_j) \in (\Gamma \times \{L, R\} \times Q)\}, \text{ and} \\ S_{\mathsf{CHECK}} :=& \{(\mathsf{a}, q_j) \in \Gamma \times Q \mid \delta(\mathsf{a}, q_j) \in (\mathsf{CHECK}_R \times Q)\}. \end{aligned}$$

State Errors For two configurations $C = (w_L, w_R, \mathsf{a}, q_j)$ and $C' = (w'_L, w'_R, \mathsf{a}', q_l)$, we say a state error for $C \vdash_{\mathcal{X}} C'$ has occurred if on reading a in state q_j, the machine $\mathcal{X}$ goes into a state that is not q_l. We also deal with the case where $\mathcal{X}$ continues its computation, even though configuration C leads to HALT.

<u>The HALT-instruction:</u> Let $E^{\mathsf{halt}}_{\mathsf{state}} \subseteq E_{\mathsf{state}}$. For every $(\mathsf{a}, q_j) \in S_{\mathsf{HALT}}$, there is a unique $i \in E^{\mathsf{halt}}_{\mathsf{state}}$ such that

$$\beta_i := y_i \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e(a)}+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2 \cdot x_{0,i} \cdot y'_i.$$

It follows that for each $(\mathsf{a}, q_j) \in S_{\mathsf{HALT}}$, there exists a unique $i \in E^{\mathsf{halt}}_{\mathsf{state}}$ such that (β_i, T_i) encodes

$$\Sigma^* \cdot \#0^{\mathsf{e(a)}+1}\#0^j\#\#0 \cdot \Sigma^*.$$

We use ## as a separator between two configurations. Thus, we encode a configuration where the head is reading an a, is in state q_j, and has a successor configuration which is an error since $\delta(\mathsf{a}, q_j) = \mathsf{HALT}$. In these β_i, we do not consider the encoding of the left and right side of the tape, nor do we encode the “successor configuration”.

While the error encoded by (β_i, T_i) may coincide with other errors, we have sufficient criteria for the word to have the specified state error.

Head movement instructions: Next, let us consider the cases where $\mathcal{X}$ moves into the wrong state. Let $E^{\mathsf{state}}_{\mathsf{state}}$ be a large enough subset of E_{state}, where $E^{\mathsf{state}}_{\mathsf{state}} \cap E^{\mathsf{halt}}_{\mathsf{state}} = \emptyset$. For $(\mathtt{a}, q_j) \in \Gamma \times Q$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, M, q_l)$ for $M \in \{L, R\}$, $\mathtt{b} \in \Gamma$, and $q_j, q_l \in Q$, we associate $l \in \mathbb{N}$ elements of $E^{\mathsf{state}}_{\mathsf{state}}$. We use $l - 1$ of these elements to encode when the state in the configuration is smaller than l, and we use one to encode when the state is larger than l.

We first look at when the state is larger than l. To help with the readability of this, we first define the pattern

$$\gamma_i := y_i \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{enc}(\mathtt{a})+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2,$$

Since we encode each state as a number, we first consider the case where the state is "too large". For each $(\mathtt{a}, q_j) \in \Gamma \times Q$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, M, q_l)$ where $M \in \{L, R\}$, $\mathtt{b} \in \Gamma$, and $q_j, q_l \in Q$, we have a unique $i \in E^{\mathsf{state}}_{\mathsf{state}}$ such that

$$\beta_i := \gamma_i \cdot z_{i,1} \cdot x_{\#,i} \cdot z_{i,2} \cdot x_{\#,i} \cdot z_{i,3} \cdot x_{\#,i} \cdot (x_{0,i})^{l+1} \cdot y_i'.$$

The above pattern encodes words of the form

$$\underbrace{\Sigma^* \cdot \#0^{\mathsf{enc}(a)+1}\#0^j\#\#}_{\gamma_i} \cdot 0^+\#0^+\#0^+\#0^{l+1} \cdot \Sigma^*.$$

We now deal with when the value representing the state is "too small". To realize this behaviour, we look at all values between 1 and $l - 1$ inclusive. That is, for every $(\mathtt{a}, q_j) \in \Gamma \times Q$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, M, q_l)$, and every $p \in [l - 1]$, we have a unique $i \in E^{\mathsf{state}}_{\mathsf{state}}$ such that

$$\beta_i := \gamma_i \cdot z_{i,1} \cdot x_{\#,i} \cdot z_{i,2} \cdot x_{\#,i} \cdot z_{i,3} \cdot x_{\#,i} \cdot (x_{0,i})^p \cdot x_{\#,i} \cdot y_i',$$

where γ_i is as defined above (with the correct i values for the variable indices). This deals with the case where the encoding section is of the form

$$\underbrace{\Sigma^* \cdot \#0^{\mathsf{enc}(a)+1}\#0^i\#\#}_{\gamma_i} \cdot 0^+\#0^+\#0^+\#0^p\# \cdot \Sigma^*$$

for some $p \in [l - 1]$.

Thus, for each pair $(\mathtt{a}, q_j) \in S_{\mathsf{L,R}}$, we have a set of patterns, each of which encodes an error in which the state in one configuration does not follow from the previous configuration.

The CHECK_R-instruction: The only state error left to deal with is when $\delta(\mathtt{a}, q_j) = (\mathsf{CHECK}_R, q_l)$ for some $q_l \in Q$. Recall that if $\delta(\mathtt{a}, q_j) = (\mathsf{CHECK}_R, q_l)$, then we have two cases: If $w_R = 0^\omega$, then $\mathcal{X}$ moves into state q_l. Otherwise, we have that $w_R \neq 0^\omega$, and $\mathcal{X}$ remains in q_j which leads to an "infinite loop". Let $E_{\mathsf{state}}^{\mathsf{check}} \subseteq E_{\mathsf{state}}$ such that $E_{\mathsf{state}}^{\mathsf{halt}}$, $E_{\mathsf{state}}^{\mathsf{state}}$, and $E_{\mathsf{state}}^{\mathsf{check}}$ forms a partition on E_{state}.

Let us consider the case where $\mathsf{e}(w_R) = 0$, and therefore $w_R = 0^\omega$. For every $(\mathsf{a}, q_j) \in S_{\mathsf{CHECK}}$ where $\delta(\mathsf{a}, q_j) = (\mathsf{CHECK}_R, q_l)$, have a unique $i \in E_{\mathsf{state}}^{\mathsf{check}}$ such that

$$\beta_i := \gamma_i \cdot z_{i,1} \cdot x_{\#,i} \cdot z_{i,2} \cdot x_{\#,i} \cdot z_{i,3} \cdot x_{\#,i} \cdot (x_{0,i})^{l+1} \cdot y_i',$$

where

$$\gamma_i := y_i \cdot x_{\#,i} \cdot x_{0,i} \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e(a)}+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2.$$

This γ_i is used as a prefix of the β_i that models an error, and describes

$$\Sigma^*\#0\#0^{\mathsf{e(a)}+1}\#0^j\#\#.$$

Thus, $\mathsf{e}(w_R) = 0$. The pattern β_i encodes the fact that the encoded state is too large by encoding the following language:

$$\underbrace{\Sigma^*\#0\#0^{\mathsf{e(a)}+1}\#0^j\#\#}_{\gamma_i}0^+\#0^+\#0^+\#0^{l+1}\Sigma^*,$$

and thus we have an error.

Now we deal with when the state is too small. For every $(\mathsf{a}, q_j) \in S_{\mathsf{CHECK}}$ where $\delta(\mathsf{a}, q_j) = (\mathsf{CHECK}_R, q_l)$, and every $p \in [l-1]$, we have a unique $i \in E_{\mathsf{state}}^{\mathsf{check}}$ such that

$$\beta_i := \gamma_i \cdot z_{i,1} \cdot x_{\#,i} \cdot z_{i,2} \cdot x_{\#,i} \cdot z_{i,3} \cdot x_{\#,i} \cdot (x_{0,i})^p \cdot (x_{\#,i})^2 \cdot y_i',$$

where

$$\gamma_i := y_i \cdot x_{\#,i} \cdot x_{0,i} \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e(a)}+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2.$$

Since, we have that after the configuration encoded by γ_i, the machine should move into state q_l (because γ_i encodes the case where $\mathsf{e}(w_R) = 0$), if we move into some state q_p where $p < l$, we have an error.

Next, we deal with the case where $\mathsf{e}(w_R) \neq 0$, and therefore $w_R \neq 0^\omega$. This is analogous to how we deal with the case when $w_R \neq 0^\omega$. For every $(\mathsf{a}, q_j) \in S_{\mathsf{CHECK}}$ where $\delta(\mathsf{a}, q_j) = (\mathsf{CHECK}_R, q_l)$, have a unique $i \in E_{\mathsf{state}}^{\mathsf{check}}$ such that

$$\beta_i := \gamma_i \cdot z_{i,2} \cdot x_{\#,i} \cdot z_{i,3} \cdot x_{\#,i} \cdot z_{i,4} \cdot x_{\#,i} \cdot (x_{0,i})^{j+1} \cdot y_i',$$

where

$$\gamma_i := y_i \cdot x_{\#,i} \cdot x_{0,i} z_{i,1} \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e(a)}+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2.$$

Note that $z_{i,1} \in 0^+$ is used for $0^{\mathsf{e}(w_L)}$, and thus $w_R \neq 0^\omega$. Therefore, the above β_i we deal with the case where the new state p is greater than j.

Next, we deal with $p < j$. for every $(\mathtt{a}, q_j) \in S_{\mathsf{CHECK}}$ where $\delta(\mathtt{a}, q_j) = (\mathsf{CHECK}_R, q_l)$, and every $1 \leq p < j$, we have a unique $i \in E^{\mathsf{check}}_{\mathsf{state}}$ such that

$$\beta_i := \gamma_i \cdot z_{i,2} \cdot x_{\#,i} \cdot z_{i,3} \cdot x_{\#,i} \cdot z_{i,4} \cdot x_{\#,i} \cdot (x_{0,i})^p \cdot x_{\#,i} \cdot y'_i,$$

where

$$\gamma_i := y_i \cdot x_{\#,i} \cdot x_{0,i} z_{i,1} \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e}(\mathtt{a})+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2.$$

This concludes the state errors.

Head Errors In the previous case (looking at state errors), we have handled all cases for which a halting configuration is followed by another configuration. Thus, for this case (looking at head errors), we can safely ignore such "halting errors".

Consider the configurations $(w_L, w_R, \mathtt{a}, q_j)$ and $(w'_L, w'_R, \mathtt{a}', q_l)$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, M, q_l)$ and $C \vdash_{\mathcal{X}} C'$. Utilizing Observation 1, we can use the parity of $\mathsf{e}(w_L)$ or $\mathsf{e}(w_R)$ to determine the symbol under the head in a successor configuration. We shall use this observation to encode head errors.

Head movement instructions: For each $(\mathtt{a}, q_j) \in \Gamma \times Q$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, L, q_l)$, there is a unique $i \in E_{\mathsf{head}}$ such that

$$\gamma_i := y_i \cdot x_{\#,i} \cdot x_{0,i} \cdot x_{\#,i} \cdot z_{i,1} \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e}(\mathtt{a})+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2.$$

The above pattern γ_i deals with the case where $\mathsf{e}(w_L) = 0$. We now define β_i for each such i as

$$\beta_i := \gamma_i \cdot z_{i,2} \cdot x_{\#,i} \cdot z_{i,3} \cdot x_{\#,i} \cdot (x_{0,i})^2 \cdot x_{\#,i} \cdot y'_i.$$

Thus, β_i expresses encoding sections of the form:

$$\underbrace{\Sigma^* \cdot \#0\#0^+\#0^{\mathsf{e}(a)+1}\#0^j\#\#}_{\gamma_i} \cdot 0^+\#0^+\#00\#\Sigma^*.$$

As the head should be moving left, and $\mathsf{e}(w_L) = 0$, it is an error for $0^{\mathsf{e}(\mathtt{a}')+1} = 00$ to hold.

We look at the more general case, where we have that $\mathsf{e}(w_L)$ is even, and greater than zero. Again, for each $(\mathtt{a}, q_j) \in \Gamma \times Q$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, L, q_l)$, there is a unique $i \in E_{\mathsf{head}}$ such that

$$\gamma_i := y_i \cdot x_{\#,i} \cdot \underbrace{x_{0,i}(z_{i,1})^2}_{\mathsf{e}(w_L)+1} \cdot x_{\#,i} \cdot z_{i,2} \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e}(\mathtt{a})+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2,$$

and β_i is defined as follows

$$\beta_i := \gamma_i \cdot z_{i,3} \cdot x_{\#,i} \cdot z_{i,4} \cdot x_{\#,i} \cdot \underbrace{(x_{0,i})^2}_{\mathsf{e}(\mathtt{a}')+1} \cdot x_{\#,i} \cdot y'_i.$$

For intuition, this deals with configurations of the form $(w_L, w_R, \mathtt{a}, q_j)$ where $\mathsf{e}(w_L)$ is even by splitting it up into two cases – either $0^{\mathsf{e}(w_L)+1} = 0$, or $0^{\mathsf{e}(w_L)+1} \in (00)^+$ by using $x_{0,i}$ and $x_{0,i}(z_{i,1})^2$ respectively. For both cases, $\mathsf{e}(w_L)$ is even, and thus $\mathsf{e}(\mathtt{a}')$ should be 0. However, since $0^{\mathsf{e}(\mathtt{a}')+1} = 00$, we have encoded an error.

Next, we deal with the analogous case for when $\mathsf{e}(w_L)$ is odd. For each $(\mathtt{a}, q_j) \in \Gamma \times Q$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, L, q_l)$, we associate some unique $i \in E_{\mathsf{head}}$ and define

$$\gamma_i := y_i \cdot x_{\#,i} \cdot \underbrace{(z_{i,1})^2}_{\mathsf{e}(w_L)+1} \cdot x_{\#,i} \cdot z_{i,2} \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e}(\mathtt{a})+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2.$$

In γ_i, we have encoded $0^{\mathsf{e}(w_L)+1}$ as $(z_{i,1})^2$. Therefore, $0^{\mathsf{e}(w_L)+1} = (00)^+$, which means $\mathsf{e}(w_L)$ is odd. We now define β_i as follows

$$\beta_i := \gamma_i \cdot z_{i,3} \cdot x_{\#,i} \cdot z_{i,4} \cdot x_{\#,i} \cdot \underbrace{x_{0,i}}_{\mathsf{e}(\mathtt{a}')+1} \cdot x_{\#,i} \cdot y_i'.$$

As $\mathsf{e}(w_L)$ is odd, we should have that $0^{\mathsf{e}(\mathtt{a}')+1} = 00$ holds, however, as we have encoded that $0^{\mathsf{e}(\mathtt{a}')+1} = x_{0,i}$, we have a head error.

Next, let us go over the details of encoding head errors when the head moves to the right. That is, for those $(\mathtt{a}, q_j) \in \Gamma \times Q$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, R, q_l)$ we associate some unique $i \in E_{\mathsf{head}}$ and define

$$\gamma_i := y_i \cdot x_{\#,i} \cdot \underbrace{x_{0,i}}_{\mathsf{e}(w_R)+1} \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e}(\mathtt{a})+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2.$$

The pattern defined above deals with the case where $\mathsf{e}(w_R) = 0$. Therefore, in the correct successor configuration, we should have that the symbol under the head is $0 \bmod 2$. We now encode the error of this kind in the subsequent β_i.

$$\beta_i := \gamma_i \cdot z_{i,1} \cdot x_{\#,i} \cdot z_{i,2} \cdot x_{\#,i} \cdot \underbrace{(x_{0,i})^2}_{\mathsf{e}(\mathtt{a}')+1} \cdot x_{\#,i} \cdot y_i'.$$

Analogously to when we move the head to the left, let us now consider the case where we move the head to the right, and $\mathsf{e}(w_R)$ is of the form $(00)^+$. For each $(\mathtt{a}, q_j) \in \Gamma \times Q$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, R, q_l)$ we have a unique $i \in E_{\mathsf{head}}$ where

$$\gamma_i := y_i \cdot x_{\#,i} \cdot \underbrace{x_{0,i} \cdot (z_{i,1})^2}_{\mathsf{e}(w_R)+1} \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e}(\mathtt{a})+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2.$$

We use γ_i as a prefix of β_i, which is defined as

$$\beta_i := \gamma_i \cdot z_{i,2} \cdot x_{\#,i} \cdot z_{i,3} \cdot x_{\#,i} \cdot \underbrace{(x_{0,i})^2}_{\mathsf{e}(\mathtt{a}')+1} \cdot x_{\#,i} \cdot y_i'.$$

Due to the fact that $\mathtt{a}' = \mathsf{e}(w_R) \bmod 2$ should hold and $\mathsf{e}(w_R)$ is even, it should hold that $\mathsf{e}(\mathtt{a}') = 0$. However since we have encoded $0^{\mathsf{e}(\mathtt{a}')+1} = 00$, we have an error.

Let us now consider the case where $\mathsf{e}(w_R)$ is odd. For each $(\mathtt{a}, q_j) \in \Gamma \times Q$ where $\delta(\mathtt{a}, q_j) = (\mathtt{b}, R, q_l)$ we have some unique $i \in E_{\mathsf{head}}$ such that

$$\begin{aligned}\gamma_i :=& y_i \cdot x_{\#,i} \cdot \overbrace{(z_{i,1})^2}^{\mathsf{e}(w_R)+1} \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e}(a)+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2, \text{ and}\\ \beta_i :=& \gamma_i \cdot z_{i,2} \cdot x_{\#,i} \cdot z_{i,3} \cdot x_{\#,i} \cdot \underbrace{x_{0,i}}_{\mathsf{e}(\mathtt{a}')+1} \cdot x_{\#,i} \cdot y_i'.\end{aligned}$$

The correctness of this behaviour is analogous to previous cases.

The CHECK_R-instruction: The last head error we must deal with is when $\delta(\mathtt{a}, q_j) = (\mathsf{CHECK}_R, q_l)$. From the definition of (CHECK_R, q_l), the symbol under the head does not change. Therefore if indeed it does change, we have an error. To that end, for each $(\mathtt{a}, q_j) \in S_{\mathsf{CHECK}}$, we have $i \in E_{\mathsf{head}}$ such that

$$\beta_i := y_i \cdot x_{\#,i} \cdot (x_{0,i})^{\mathsf{e}(\mathtt{a})+1} \cdot x_{\#,i} \cdot (x_{0,i})^j \cdot (x_{\#,i})^2 \cdot z_{i,1} \cdot x_{\#,i} \cdot z_{i,2} \cdot x_{\#,i} \cdot (x_{0,i})^{c+1} \cdot x_{\#,i} \cdot y_i',$$

where if $\mathtt{a} = 1$, then $c = 0$ and if $\mathtt{a} = 0$, then $c = 1$. This encodes a change in the symbol under the head, which is an error if $\delta(\mathtt{a}, q_j) = \mathsf{CHECK}_R(q_l)$.

Tape Errors In the proof of Lemma 7, we define ρ many patterns to encode the tape errors. The patterns in E_{tape} are identical to the patterns used to encode tape errors in the proof of Lemma 7, with the necessary changes to the indices. More formally, let $\{\beta_i \mid i \in [\rho]\}$ be the set of patterns used to define tape errors for $\mathcal{X}$, as defined in Lemma 7. Let $f : [\rho] \to E_{\mathsf{tape}}$ be a bijective function. Then, for each $i \in [\rho]$, let $\beta_{f(i)}$ be β_i, where the only difference is that $f(i)$ is used for the indices of the variables, rather than i. Since we have the same assumptions for the variables $x_{\mathtt{a},i}$, $z_{i,r}$ and y_i, y_i', it is clear that each β_i for $i \in E_{\mathsf{tape}}$ encodes a tape error.

Defining the Formula Let

$$\varphi := \varphi_{0\#} \wedge \varphi_{\mathsf{err}} \wedge \varphi_{\mathsf{type}} \wedge \varphi_{+}.$$

First, let

$$\varphi_{0\#} := (\mathsf{w} \doteq 0 \cdot \# \cdot 0 \cdot z) \wedge (z \doteq \#^3 \cdot z' \cdot \#^3).$$

This $\varphi_{0\#}$ ensures that any $w \in \mathcal{L}(\varphi)$ is of the form $0\#0\#^3\Sigma^*\#^3$.

Let φ_{err} be defined by

$$\varphi_{\mathsf{err}} := (z \doteq \alpha_1\alpha_2 \cdots \alpha_\mu) \wedge \bigwedge_{i=1}^{\mu} (\alpha_i \doteq (x_{\#,i})^3 \cdot \beta_i \cdot (x_{\#,i})^3),$$

where each β_i encodes an error. It is clear that for any substitution σ such that $\sigma \models \varphi$, we have that $\sigma(\mathsf{w}) = 0\#0\#^3u\#^3 = 0\#0 \cdot \sigma(\alpha_1\alpha_2 \cdots \alpha_\mu)$ for some $u \in \Sigma^*$. Consequently, we have that $\sigma(\alpha_1\alpha_2 \cdots \alpha_\mu) = \#^3u\#^3$.

Next, we consider φ_{type} and φ_{+} which deals with types on the variables. The formula φ_{type} ensures that for any $i \in [\mu]$ and any substitution σ where $\sigma \models \varphi$ we have that $\sigma(x_{0,i}) \in \{\varepsilon, 0\}$ and $\sigma(x_{\#,i}) \in \{\varepsilon, \#\}$. Furthermore, we ensure that there is exactly one $i \in [\mu]$ such that $\sigma(x_{\#,i} \cdot x_{0,i}) = \#0$ and for all $i' \in [\mu] \setminus \{i\}$ we have that $\sigma(x_{\#,i'} \cdot x_{0,i'}) = \varepsilon$.

$$\varphi_{\mathsf{type}} := (x_0 \doteq 0) \wedge (x_0 \doteq x_{0,1} \cdot x_{0,2} \cdots x_{0,\mu}) \wedge (x_\# \doteq \#) \wedge (x_\# \doteq x_{\#,1} \cdot x_{\#,2} \cdots x_{\#,\mu}) \\ \wedge (x' \doteq 0 \cdot \# \cdot 0) \wedge \big(x' \doteq \prod_{i=1}^{\mu} (x_{0,i} \cdot x_{\#,i} \cdot x_{0,i})\big).$$

Let σ be a substitution where $\sigma \models \varphi$. Since we have that $\sigma(x_0) = 0$ and $\sigma(x_0) = \sigma(x_{0,i} \cdots x_{0,\mu})$ it follows that there is exactly one $i \in [\mu]$ such that $\sigma(x_{0,i}) = 0$ and $\sigma(x_{0,i'}) = \varepsilon$ for all $i' \in [\mu] \setminus \{i\}$. The analogous reasoning states that there is exactly one $i \in [\mu]$ such that $\sigma(x_{\#,i}) = \#$. Furthermore, $\sigma(x_{\#,i}) \neq \varepsilon$ if and only if $\sigma(x_{0,i}) \neq \varepsilon$ due to the fact that $\prod_{i=1}^{\mu} \sigma(x_{0,i} \cdot x_{\#,i} \cdot x_{0,i}) = 0 \cdot \# \cdot 0$. That is, if $\sigma(x_{0,i}) \neq \varepsilon$ and $\sigma(x_{\#,i'}) \neq \varepsilon$ for some $i, i' \in [\mu]$ where, without loss of generality, $i < i'$, then

$$0\#0 = \sigma(x') = \prod_{i=1}^{\mu} \sigma(x_{0,i} \cdot x_{\#,i} \cdot x_{0,i}) = 00 \cdot \#,$$

due to the fact that there is exactly one $i \in [\mu]$ such that $\sigma(x_{\mathtt{a},i}) = \mathtt{a}$ for each $\mathtt{a} \in \Sigma$. If this is the case, then $\sigma \models \varphi$ does not hold.

Thus, φ_{type} states that for any substitution σ where $\sigma \models \varphi$ there is exactly one $i \in [\mu]$ such that for all $\mathtt{a} \in \Sigma$, we have $\sigma(x_{\mathtt{a},i}) = \mathtt{a}$, and for all $j \in [\mu] \setminus \{i\}$, we have $x_{\mathtt{a},j} = \varepsilon$. If this does indeed hold, then we say that $i \in [\mu]$ is the *selected error* for σ.

The last subformula φ_{+} deals with the type for variables of the form $z_{i,1}$, $z_{i,2}$, etc. For this, we define:

$$\varphi_{+} := \bigwedge_{i=1}^{\mu} \bigwedge_{r=1}^{4} \big((z_{i,r} \doteq x_{0,i} \cdot z'_{i,r}) \wedge (z_{i,r} \doteq z'_{i,r} \cdot x_{0,j})\big).$$

Thus, if $\sigma(x_{0,i}) = 0$, then $z_{i,1}, z_{i,2}, z_{i,3}, z_{i,4} \in \mathcal{L}(0^{+})$.

Therefore, if $i \in [\mu]$ is the selected error for σ, then

- $\sigma(x_{0,i}) = 0$,
- $\sigma(x_{\#,i}) = \#$, and
- $\sigma(z_{i,r}) \in 0^{+}$ for $r \in [4]$.

Consequently, if $i \in [\mu]$ is the selected error for σ, then $\sigma(\beta_i) \in \mathsf{INVALC}(\mathcal{X})$. This is because each β_i uses these variables (which are not mapped to the empty-word if i is the selected error for σ) to encode some error that prohibits $\sigma(\beta_i)$ from being in $\mathsf{VALC}(\mathcal{X})$.

Correctness For this correctness proof, we show that

$$\mathcal{L}(\varphi) := 0\#0\#^3\mathsf{INVALC}(\mathcal{X})\#^3.$$

In other words, $0\#0\#^3w\#^3 \in \mathcal{L}(\varphi)$ if and only if $w \in \mathsf{INVALC}(\mathcal{X})$.

If direction: Let $v = 0\#0\#^3w\#^3$ where $w \in \mathsf{INVALC}(\mathcal{X})$. Since $w \in \mathsf{INVALC}(\mathcal{X})$, the word must contain at least one error. For every possible $w \in \mathsf{INVALC}(\mathcal{X})$, we have some $i \in [\mu]$ such that $w \in \mathcal{L}(\beta_i, T_i)$. Thus, there exists $i \in [\mu]$ and $\sigma \models \varphi$ where i is the selected error for σ and

$$\sigma(\mathsf{w}) = 0 \cdot \# \cdot 0 \cdot \#^3 \cdot \sigma(\beta_i) \cdot \#^3,$$

and $\sigma(\mathsf{w}) = v$. It follows that $\sigma(\alpha_{i'}) = \varepsilon$ for all $i' \in [\mu] \setminus \{i\}$. Thus, we have dealt with one direction for the correctness proof.

Only if direction: Let $v \in \mathcal{L}(\varphi)$. From the structure of φ, we know that $v = 0\#0\#^3w\#^3$ for some $w \in \Sigma^*$. Furthermore, we know that for any substitution σ where $\sigma \models \varphi$ such that $\sigma(\mathsf{w}) = v$, we have that $\#^3w\#^3 = \sigma(\alpha_1\alpha_2\cdots\alpha_\mu)$.

Let $v = 0\#0\#^3w\#^3$ where $v \in \mathcal{L}(\varphi)$. Let $\sigma \models \varphi$ where $\sigma(\mathsf{w}) = v$, and let $i \in [\mu]$ be the selected error for σ. We now look at two cases.

- *Case 1.* For all $j \in [\mu] \setminus \{i\}$, we have that $\sigma(\alpha_j) = \varepsilon$.
- *Case 2.* There exists $j \in [\mu] \setminus \{i\}$ such that $\sigma(\alpha_j) \neq \varepsilon$.

Recall that since i is the selected error for σ, we have that $\sigma(x_{0,i}) = 0, \sigma(x_{\#,i}) = \#$, and $z_{i,r} \in 0^+$ for $r \in [4]$.

Case 1: For all $j \in [\mu] \setminus \{i\}$, we have $\sigma(\alpha_j) = \varepsilon$. Therefore,

- $\sigma(\mathsf{w}) = 0 \cdot \# \cdot 0 \cdot \sigma(z)$,
- $\sigma(z) = \sigma(\alpha_i)$, and
- $\sigma(\alpha_i) = \#^3 \cdot \sigma(\beta_i) \cdot \#^3$.

Consequently, $\sigma(\mathsf{w}) = 0\#0\#^3 \cdot \sigma(\beta_i) \cdot \#^3$ where $\sigma(\beta_i) \in \mathsf{INVALC}(\mathcal{X})$.

Case 2: Let $j \in [\mu] \setminus \{i\}$ such that $\sigma(\alpha_j) \neq \varepsilon$. For any $\sigma \models \varphi$, we know that:

$$\sigma(\mathsf{w}) = 0 \cdot \# \cdot 0 \cdot \#^3 \cdot v \cdot \#^3, \text{ and}$$
$$\sigma(\mathsf{w}) = 0 \cdot \# \cdot 0 \cdot \sigma(z),$$

where $\sigma(z) = \sigma(\alpha_1 \cdot \alpha_2 \cdots \alpha_i \cdots \alpha_\mu)$. Thus,

$$\sigma(\alpha_1 \cdot \alpha_2 \cdots \alpha_{i-1}) \cdot \sigma(\alpha_i) \cdot \sigma(\alpha_{i+1} \cdots \alpha_\mu) = \#^3 \cdot v \cdot \#^3.$$

Recall that i is the selected error for σ and therefore $\sigma(\alpha_i) \in \#^3 \cdot \Sigma^* \cdot \#^3$. Because there exists some $j \in [\mu] \setminus \{i\}$ where $\sigma(\alpha_j) \neq \varepsilon$, we have that

$$w_1 \cdot \underbrace{\#^3 \cdot \sigma(\beta_i) \cdot \#^3}_{\sigma(\alpha_i)} \cdot w_2 = \#^3 \cdot v \cdot \#^3,$$

where $\sigma(\alpha_1 \cdots \alpha_{i-1}) = w_1$ and $\sigma(\alpha_{i+1} \cdots \alpha_\mu) = w_2$, and $w_1 \cdot w_2 \neq \varepsilon$. The rest of the proof for this direction shows that $v \in \mathsf{INVALC}(\mathcal{X})$.

- Let $|w_1| > 3$. Then, $w_1 = \#^3 w_1'$ where $w_1' \in \Sigma^+$. It therefore must hold that $\sigma(\alpha_1 \cdots \alpha_i) = \#^3 w_1' \#^3 \sigma(\beta_i) \#^3$, and thus we have a structural error due to the occurrence of $\#^3$ in w.
- Let $|w_2| > 3$. Then, $w_2 = w_2' \#^3$ where $w_2' \in \Sigma^+$. Thus, we have that $\sigma(\alpha_i \cdots \alpha_\mu) = \#^3 \sigma(\beta_i) \#^3 w_2' \#^3$, and we again have a structural error.
- Let $|w_1| \leq 3$ and $|w_2| \leq 3$. Then, $\sigma(\alpha_1 \cdots \alpha_\mu) = \#^{k_1} \cdot \sigma(\beta_i) \cdot \#^{k_2}$ for some $k_1, k_2 \geq 3$, where $\sigma(\beta_i) \in \mathsf{INVALC}(\mathcal{X})$. Therefore, $w \in \mathsf{INVALC}(\mathcal{X})$.

Thus, we have shown that $v \in \mathcal{L}(\varphi)$ where $v = 0 \cdot \# \cdot 0 \cdot \#^3 \cdot w \cdot \#^3$ if and only if $w \in \mathsf{INVALC}(\mathcal{X})$.

To conclude, given an extended Turing machine $\mathcal{X}$, we construct $\varphi \in \mathsf{FC\text{-}CQ}$ such that

$$\mathcal{L}(\varphi) := \{0 \cdot \# \cdot 0 \cdot \#^3 \cdot w \cdot \#^3 \mid w \in \mathsf{INVALC}(\mathcal{X})\}.$$

Since regular languages are closed under quotients, it is thus clear that $\mathcal{L}(\varphi)$ is regular if and only if $\mathsf{INVALC}(\mathcal{X})$ is regular. Therefore, observing Lemma 6, the regularity problem for FC-CQ is undecidable. □

5 Descriptional Complexity

In this section, we consider some of the consequences of the aforementioned undecidability results. First, we look at *minimization*. To examine the problem of minimization, we first must discuss what complexity measure we wish to minimize for. Instead of giving an explicit measure, such as the length, we give the more general definition of a *complexity measure* from Kutrib [32].

Definition 9 Let $\mathcal{F} \in \{\mathsf{FC\text{-}CQ}, \mathsf{REG}\}$, where REG is the set of regular expressions. A *complexity measure* for $\mathcal{F}$ is a recursive function $c\colon \mathcal{F} \to \mathbb{N}$ such that the elements of $\mathcal{F}$ can be enumerated effectively in increasing order, and for every $n \in \mathbb{N}$, there exist finitely many $\varphi \in \mathcal{F}$ with $c(\varphi) = n$.

From Theorem 7, we conclude that FC-CQ cannot be minimized effectively:

Corollary 3 *Let c be a complexity measure for* FC-CQ*. There is no algorithm that given $\varphi \in$* FC-CQ*, constructs $\psi \in$* FC-CQ *such that $\mathcal{L}(\varphi) = \mathcal{L}(\psi)$ and ψ is c-minimal.*

Proof Consider $\varphi \in \mathsf{FC\text{-}CQ}$ that was constructed in the proof of Theorem 7 from an extended Turing machine. We remind the reader that for a given extended Turing machine $\mathcal{X}$, we construct φ such that:

$$\mathcal{L}(\varphi) = \{0 \cdot \# \cdot 0 \cdot \#^3 \cdot w \cdot \#^3 \mid w \in \mathsf{INVALC}(\mathcal{X})\}.$$

Now consider the following formula:

$$\psi := (\mathsf{w} \doteq 0\#0\#^3 x \#^3).$$

From ψ and φ, we have that $\mathcal{L}(\varphi) = \mathcal{L}(\psi)$ if and only if $\mathsf{INVALC}(\mathcal{X}) = \Sigma^*$. As determining whether $\mathsf{INVALC}(\mathcal{X}) = \Sigma^*$ is undecidable (recall Lemma 6), it follows that deciding whether $\mathcal{L}(\varphi) = \mathcal{L}(\psi)$ is undecidable.

Since ψ is fixed, the set of c-minimal queries $\psi' \in \mathsf{FC[REG]\text{-}CQ}$ such that $\mathcal{L}(\psi) = \mathcal{L}(\psi')$ is finite, there exists a recursive function to find such a c-minimal formula ψ'. Now assume that there exists an algorithm P that given φ, gives an equivalent φ' such that $c(\varphi')$ is minimal. Then, there would be an algorithm that determines whether $\mathsf{INVALC}(\mathcal{X}) = \Sigma^*$ by checking whether φ' is equivalent to ψ'. Consequently, the assumption that P exists cannot hold. ◻

Given complexity measures c_1 and c_2 for FC-CQ and REG (respectively), we say that there is a *non-recursive trade-off* from FC-CQ to REG if for every recursive function $f : \mathbb{N} \to \mathbb{N}$, there exists $\varphi \in \mathsf{FC\text{-}CQ}$ such that $\mathcal{L}(\varphi) \in \mathcal{L}(\mathsf{REG})$, but $c_2(\gamma) > f(c_1(\varphi))$ holds for every $\gamma \in \mathsf{REG}$ with $\mathcal{L}(\varphi) = \mathcal{L}(\gamma)$.

Hartmanis' meta theorem [33] allows us to draw conclusions about the relative succinctness of models from certain undecidability results. Thus, we can conclude the following.

Theorem 8 *The trade-off from* FC-CQ *to* REG *is non-recursive.*

Proof This proof follows from Hartmanis' meta theorem [33] which states the following: For two systems A and B of representations, given a representation $r \in B$, if it is not co-semi-decidable whether there exists an equivalent representation $r' \in A$, then there is a non-recursive trade-off from B to A. See [32] for more details. From the proof of Theorem 7 we know that determining whether $\mathcal{L}(\varphi)$ is regular for a given $\varphi \in \mathsf{FC\text{-}CQ}$ is not decidable, semi-decidable, or co-semi-decidable. Thus, we can invoke Hartmanis' meta theorem, to determine that there is a non-recursive trade-off from FC-CQ to regular expressions. ◻

Less formally, Theorem 8 states that even for those FC-CQs that generate a regular language, the size blowup from the FC-CQ to the regular expression that accepts the same language is not bounded by any recursive function.

While Theorem 8 also shows that the trade-off from FC[REG]-CQ to regular expressions is non-recursive, this seems less surprising. Purely from the definition of FC-CQ, it does not seem like FC-CQ should be able to generate many complicated regular languages. Thus, the fact that the size blowup from FC-CQ to an equivalent regular expression is not bounded by any recursive function highlights the deceptive complexity of languages generated by FC-CQ.

6 Conclusions

This paper studies FC-CQ and FC[REG]-CQ, with a particular focus on language theoretic questions. Regarding the expressive power, Fig. 1 gives an inclusion diagram. However, there are still many open problems, such as whether $\mathcal{L}(\mathsf{FC[REG]\text{-}CQ})$ is closed under union. Furthermore, it is not known whether $\mathcal{L}(\mathsf{EP\text{-}C}) = \mathcal{L}(\mathsf{FC\text{-}UCQ})$. As $\mathcal{L}(\mathsf{FC\text{-}UCQ}) = \mathcal{L}(\mathsf{EP\text{-}FC})$, this question is particularly fundamental, as it asks whether the finite model restriction of FC decreases the expressive power compared to the theory of concatenation (see [4]).

With regards to decision problems, we show that the membership problem for FC-CQ is NP-complete, and remains NP-hard even if the input word is of length one, and the formula only consists of regular patterns. Restrictions like acyclicity (see [8]) and bounded width (see [4]) lead to tractable fragments; however, there is still a lot of research to be done on identifying further tractable fragments and comparing their expressive power. The main technical contribution of this paper is that the universality problem for FC[REG]-CQ, and the regularity problem for FC-CQ is undecidable.

Acknowledgements This work was funded by EPSRC grant EP/T033762/1. The authors would like to thank Joel D. Day, Anthony W. Lin, Ana Sălăgean, and the anonymous reviewers for their helpful feedback.

Author Contributions Both authors are equally responsible for the content and reviewed the submission of this manuscript.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no Conflict of interest.

References

1. Karhumäki, J., Mignosi, F., Plandowski, W.: The expressibility of languages and relations by word equations. J. ACM **47**(3), 483–505 (2000)
2. Diekert, V., Gutierrez, C., Hagenah, C.: The existential theory of equations with rational constraints in free groups is PSPACE-complete. Inf. Comput. **202**(2), 105–140 (2005)
3. Durnev, V.G.: Undecidability of the positive ∀∃ 3-theory of a free semigroup. Sib. Math. J. **36**(5), 917–929 (1995)
4. Freydenberger, D.D., Peterfreund, L.: The theory of concatenation over finite models. In: Proceedings of ICALP 2021, pp. 130–113017 (2021)
5. Fagin, R., Kimelfeld, B., Reiss, F., Vansummeren, S.: Document spanners: A formal approach to information extraction. J. ACM **62**(2), 12 (2015)
6. Freydenberger, D.D., Kimelfeld, B., Peterfreund, L.: Joining extractions of regular expressions. In: Proceedings of PODS 2018, pp. 137–149 (2018)
7. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases vol. 8. Addison-Wesley Reading, Reading (1995)
8. Freydenberger, D.D., Thompson, S.M.: Splitting spanner atoms: A tool for acyclic core spanners. In: Proceedings of ICDT 2022, pp. 6–1618 (2022)
9. Freydenberger, D.D.: A logic for document spanners. Theory Comput. Syst. **63**(7), 1679–1754 (2019)
10. Freydenberger, D.D., Holldack, M.: Document spanners: From expressive power to decision problems. Theory Comput. Syst. **62**(4), 854–898 (2018)
11. Thompson, S.M., Freydenberger, D.D.: Generalized core spanner inexpressibility via Ehrenfeucht-Fraïssé games for FC. arXiv:2306.16364 (2023)
12. Angluin, D.: Finding patterns common to a set of strings. J. Comput. Syst. Sci. **21**(1), 46–62 (1980)
13. Geilke, M., Zilles, S.: Polynomial-time algorithms for learning typed pattern languages. In: International Conference on Language and Automata Theory and Applications, pp. 277–288 (2012)

14. Koshiba, T.: Typed pattern languages and their learnability. In: European Conference on Computational Learning Theory, pp. 367–379 (1995)
15. Schmid, M.L.: Inside the class of regex languages. In: Proceedings of DLT 2012, pp. 73–84 (2012)
16. Lothaire, M.: Combinatorics on Words, vol. 17. Cambridge University Press, Cambridge (1997)
17. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Volume 1: Word, Language, Grammar. Springer, Berlin (1997)
18. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, vol. 3. Addison-Wesley Longman Publishing Co., Inc, New York (2006)
19. Freydenberger, D.D., Schmid, M.L.: Deterministic regular expressions with back-references. J. Comput. Syst. Sci. **105**, 1–39 (2019)
20. Schmid, M.L.: Characterising REGEX languages by regular languages equipped with factor-referencing. Inf. Comput. **249**, 1–17 (2016)
21. Carle, B., Narendran, P.: On extended regular expressions. In: Proceedings of LATA 2009, pp. 279–289 (2009)
22. Shinohara, T.: Polynomial time inference of extended regular pattern languages. In: RIMS Symposia on Software Science and Engineering, pp. 115–127 (1983)
23. Ehrenfreucht, A., Rozenberg, G.: Finding a homomorphism between two words is NP-complete. Inf. Process. Lett. **9**(2), 86–88 (1979)
24. Manea, F., Schmid, M.L.: Matching patterns with variables. In: Combinatorics on Words: 12th International Conference, WORDS 2019, Loughborough, UK, September 9–13, 2019, Proceedings 12, pp. 1–27. Springer (2019)
25. Fernau, H., Schmid, M.L.: Pattern matching with variables: A multivariate complexity analysis. Inf. Comput. **242**, 287–305 (2015)
26. Fernau, H., Schmid, M.L., Villanger, Y.: On the parameterised complexity of string morphism problems. Theory Comput. Syst. **59**(1), 24–51 (2016)
27. Plandowski, W.: Satisfiability of word equations with constants is in PSPACE. J. ACM **51**(3), 483–496 (2004)
28. Manea, F., Nowotka, D., Schmid, M.L.: On the solvability problem for restricted classes of word equations. In: International Conference on Developments in Language Theory, pp. 306–318. Springer (2016)
29. Day, J.D., Manea, F., Nowotka, D.: The hardness of solving simple word equations. In: Proceedings of MFCS 2017, pp. 18–11814 (2017)
30. Diekert, V., Robson, J.M.: Quadratic word equations. Jewels are Forever: Contributions on Theoretical Computer Science in Honor of Arto Salomaa, pp. 314–326 (1999)
31. Freydenberger, D.D.: Extended regular expressions: Succinctness and decidability. Theory Comput. Syst. **53**(2), 159–193 (2013)
32. Kutrib, M.: The phenomenon of non-recursive trade-offs. Int. J. Found. Comput. Sci. **16**(05), 957–973 (2005)
33. Hartmanis, J.: On Gödel speed-up and succinctness of language representations. Theor. Comput. Sci. **26**(3), 335–342 (1983)