Topological Sorting with Regular Constraints

Antoine Amarilli, Charles Paperman

Topological Sorting with Regular Constraints

Antoine Amarilli
LTCI, Télécom ParisTech, Université Paris-Saclay

Charles Paperman
Université de Lille

Abstract

We introduce the *constrained topological sorting problem* (CTS): given a regular language K and a directed acyclic graph G with labeled vertices, determine if G has a topological sort that forms a word in K. This natural problem applies to several settings, e.g., scheduling with costs or verifying concurrent programs. We consider the problem CTS[K] where the target language K is fixed, and study its complexity depending on K. We show that CTS[K] is tractable when K falls in several language families, e.g., *unions of monomials*, which can be used for pattern matching. However, we show that CTS[K] is NP-hard for $K = (ab)^*$ and introduce a *shuffle reduction* technique to show hardness for more languages. We also study the special case of the *constrained shuffle problem* (CSh), where the input graph is a disjoint union of strings, and show that CSh[K] is additionally tractable when K is a group language or a union of district group monomials. We conjecture that a dichotomy should hold on the complexity of CTS[K] or CSh[K] depending on K, and substantiate this by proving a coarser dichotomy under a different problem phrasing which ensures that tractable languages are closed under common operators.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Topological sorting; shuffle problem; regular language

Acknowledgements We thank Michaël Cadilhac and Pierre McKenzie for their fruitful insights.

1 Introduction

Many scheduling or ordering problems amount to computing a *topological sort* of a directed acyclic graph (DAG), i.e., a totally ordered sequence of the vertices that is compatible with the edge relation: when we enumerate a vertex, all its predecessors must have been enumerated first. However, in some settings, we need a topological sort satisfying additional constraints that cannot be expressed as edges. We formalize this problem as follows: the vertices of the DAG are labeled with some symbols from a finite alphabet A, and we want to find a topological sort that falls into a specific regular language. We call this the *constrained topological sort problem*, or CTS. For instance, if we fix the language $K = ab^*c$, and consider the example DAGs of Figure 1, then G_1 and G_2 have a topological sort that falls in K.

CTS relates to many applications. For instance, many *scheduling* applications use a dependency graph [1] of tasks, and it is often useful to express other constraints, e.g., some tasks must be performed by specific workers and we should not assign more than p successive tasks to the same worker. We can express this as a CTS-problem: label each task by the worker which can perform it, and consider the target regular language K containing all words where the same symbol is not repeated more than p times. In *concurrency* applications, we may consider a program with multiple threads, and want to verify that there is no linearization of its instructions that exhibits some unsafe behavior, e.g., executing a read before a write. To search for such a linearization, we can label each instruction with its type, and consider CTS with a target language describing the behavior that we wish to detect. CTS can also be used in uncertain data management tasks, to reason about the possible answers

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Daniel Marx, and Don Sannella; Article No. 41; pp. 41:1–41:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

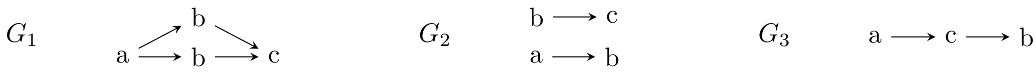

Figure 1 Example labeled DAGs on the alphabet $A = \{a, b, c\}$

of aggregate queries on uncertain ordered data [4]. It can also be equivalently phrased in the language of partial order theory: seeing the labeled DAG as a *labeled partial order* $<$, we ask if some *linear extension* achieves a word in K.

We thus believe that the CTS-problem is useful, and natural, but we are not aware of previous work studying it, except for a special case called the *shuffle problem*. This problem deals with the *interleaving* of strings, as studied, e.g., in concurrent programming languages [17, 20], computational biology [16], and formal languages [9, 7, 25]. Specifically, we are given a tuple of strings, and we must decide if they have some interleaving that falls in the target language K. This problem was known to be NP-complete [18, 31, 15] when the target language K is given as input (in addition to the tuple of strings), even when K consists of just one target string. To rephrase this shuffle problem in our context, we call *constrained shuffle problem* (CSh) the special case of CTS where we require input DAGs to be a union of directed path graphs (corresponding to the strings).

Our goal in this paper is to study the complexity of CTS and CSh. We assume that the target regular language K is fixed, and call CTS[K] and CSh[K] the corresponding problems, whose complexity is only a function of the input DAG (labeled on the alphabet A of K). Our central question is: *for which regular languages K are the problems* CTS[K] *or* CSh[K] *tractable?* More precisely, for each of these problems, we conjecture a dichotomy on K: the problem is either in NL or it is NP-complete. However, the tractability boundary is challenging to chart out, and we have not been able to prove these conjectures in full generality. In this paper, we present the results that we have obtained towards this end.

Paper structure. We formally define the CTS and CSh problems in Section 2 and state the conjecture. We then show the following results:

- In Section 3, we present our hardness results. We recall the results of [31] on the shuffle problem, and present a general *shuffle reduction* technique to show hardness for more languages. We use it in particular to show that CSh[$(ab)^*$], hence CTS[$(ab)^*$], are NP-hard, and extend this to several other languages.
- In Section 4, we present tractability results. We show that CTS[K], hence CSh[K], is in non-deterministic logspace (NL) when K is a union of *monomial languages*, i.e., of languages of the form $A_1^* a_1 \cdots A_{n-1}^* a_{n-1} A_n^*$, with the a_i being letters and the A_i being subalphabets. Such languages can be used for applications such as pattern matching, e.g., with the language $A^* u A^*$ for a fixed pattern $u \in A^*$. We also show tractability for other languages that are not of this form, e.g. $(ab)^* + A^* aa A^*$ and variants thereof, using different techniques such as Dilworth's theorem [8].
- In Section 5, we use our hardness and tractability results to show a coarser dichotomy result. Specifically, we give an alternative phrasing of the CTS and CSh problems using *semiautomata* and DAGs with *multi-letter* labels: this amounts to closing the tractable languages under intersection, inverse morphism, complement, and quotients. In this phrasing, when the semiautomaton is counter-free, we can show that the problems are either in NL or NP-complete. This dichotomy is effective, i.e., the criterion on the semiautomaton is decidable, and it turns out to be the same for CTS and CSh.
- In Section 6, we focus on the constrained shuffle problem, and lift the counter-free assumption of the previous section. We show that CSh[K] is tractable when K is a *group*

language or more generally a union of *district group monomials*. This tractability result is the main technical contribution of the paper, with a rather involved proof. It implies, e.g., that the following problem is in NL for any fixed finite group H: given $g \in H$ and words $w_1, \ldots, w_n$ of elements of H, decide whether there is an interleaving of the w_i which evaluates to g according to the group operation.

2 Problem Statement and Main Results

We give some preliminaries and define the two problems that we study. We fix a finite alphabet A, and call A^* the set of all finite words on A. For $w \in A^*$, we write $|w|$ for the *length* of w, and write $|w|_a$ for the number of occurrences of $a \in A$ in w. We denote the empty word by ϵ. A *labeled DAG* on the alphabet A, or *A-DAG*, is a triple $G = (V, E, \lambda)$ where (V, E) is a directed acyclic graph with vertex set $V = \{1, \ldots, n\}$ and edge set $E \subseteq V \times V$, and where $\lambda : V \to A$ is a function giving a label in A to each vertex in V. For $u \neq v$ in V, we say that u is an *ancestor* of v if there is a directed path from u to v in G, we say that u is a *descendant* of v if v is an ancestor of u, and otherwise we call u and v *incomparable*. A *topological sort* of G is a bijective function σ from $\{1, \ldots, n\}$ to V such that, for all $(u, v) \in E$, we have $\sigma^{-1}(u) < \sigma^{-1}(v)$. The word *achieved* by σ is $\lambda(\sigma) := \lambda(\sigma(1)) \cdots \lambda(\sigma(n)) \in A^*$.

The *constrained topological sort problem* CTS[K] for a fixed language $K \subseteq A^*$ (described, e.g., by a regular expression) is defined as follows: given an A-DAG G, determine if there is a topological sort σ of G such that $\lambda(\sigma) \in K$ (in which case we say that σ *achieves* K).

We now define the *constrained shuffle problem* (CSh). Given two words $u, v \in A^*$, the *shuffle* [31] of u and v, written $u \sqcup\!\sqcup v$, is the set of words that can be obtained by interleaving them. Formally, a word $w \in A^*$ is in $u \sqcup\!\sqcup v$ iff there is a partition $P \sqcup Q$ of $\{1, \ldots, |w|\}$ such that $w_P = u$ and $w_Q = v$, where w_P denotes the sub-word of w where we keep the letters at positions in P, and likewise for w_Q. The *shuffle* $\sqcup\!\sqcup(U)$ of a tuple of words U is defined by induction as follows: we set $\sqcup\!\sqcup() := \{\epsilon\}$, set $\sqcup\!\sqcup(u) := \{u\}$, and set $\sqcup\!\sqcup(u_1, \ldots, u_n, u_{n+1}) := \bigcup_{v \in \sqcup\!\sqcup(u_1, \ldots, u_n)} v \sqcup\!\sqcup u_{n+1}$. The *constrained shuffle problem* CSh[K] for a fixed language $K \subseteq A^*$ is defined as follows: given a tuple of words U, determine if $K \cap \sqcup\!\sqcup(U)$ is nonempty. Of course, CSh[K] is a special case of CTS[K]: we can code any tuple of words U as an A-DAG G_U by coding each $u \in U$ as a directed path graph $v_1 \to \cdots \to v_{|u|}$ with $\lambda(v_i) = u_i$ for all $1 \leq i \leq |u|$. Thus, we will equivalently see inputs to CSh as tuples of words (called *strings* in this context) or as A-DAGs that are unions of directed path graphs.

▶ **Example 2.1.** The problem CTS[$(ab)^*$] on an input $\{a, b\}$-DAG G asks if G has a topological sort starting with an a, ending with a b, and alternating between elements of each label. The problem CSh[$(aa + b)^*$] on a tuple U of strings on $\{a, b\}$ asks if there is an interleaving $w \in \sqcup\!\sqcup(U)$ such that all a^*-factors in w are of even length (e.g., *bbaabaaaa*, but not *baaabb*).

In this work, we study the complexity of the problems CTS[K] and CSh[K] depending on the language K. Clearly we can always solve these problems by guessing a topological sort (or an interleaving), and verifying that it achieves a word in K. Hence, the complexity is always in NP^K, that is, in non-deterministic PTIME with an oracle for the *word problem* of K, which we can call to test if an input word in is K:

▶ **Proposition 2.2.** *For any language K, the problems* CTS[K] *and* CSh[K] *are in* NP^K.

In particular, the problems are in NP when the language K is regular, because the word problem for regular languages is in PTIME. We will study regular languages in this work. We believe that regular languages can be classified depending on the complexity of these problems, and make the following *dichotomy conjecture*:

▶ **Conjecture 2.3.** *For every regular language K, the problem* $\mathrm{CTS}[K]$ *is either in NL or NP-complete. Likewise, the problem* $\mathrm{CSh}[K]$ *is either in NL or NP-complete.*

Towards this conjecture, we determine in this paper the complexity of CTS and CSh for several languages and classes. We first show in the next section that these problems are hard for some languages such as $(ab)^*$, and we then show tractability results in Section 4, and a coarser dichotomy result in Section 5 under an alternative phrasing of our problems.

3 Hardness Results

Our hardness results are based on the *shuffle problem* of formal language theory which asks, given a word $w \in A^*$ and a tuple U of words of A^*, whether $w \in ⧢(U)$. This problem is known to be NP-hard already on the alphabet $\{a, b\}$ (see [31]). The shuffle problem is different from CSh, because the target word of the shuffle problem is given as input, whereas the target regular language of CSh is fixed. However, the hardness of the shuffle problem directly implies the hardness of CSh, hence of CTS, for a well-chosen target language:

▶ **Proposition 3.1.** *Let $K_0 := (a_1a_2 + b_1b_2)^*$. The problem* $\mathrm{CSh}[K_0]$ *is NP-hard.*

Proof sketch. We can reduce a shuffle instance (w, U) to the instance $I := w_1 \cup U_2$ for $\mathrm{CSh}[K_0]$, where w_1 is w but adding the subscript 1 to all labels, and U_2 is defined analogously. A topological sort of I achieving K_0 must then alternate between w_1 and U_2, and enumerate letters with the same label (up to the subscript), witnessing that $w \in ⧢(U)$. ◀

In this section, we will refine this approach to show hardness for more languages. We first recall another initial hardness result from [31]. We then introduce a general *shuffle reduction* technique to show the hardness of languages by reducing from other hard languages. Last, we show that CTS and CSh are hard for the language $(ab)^*$ and for other languages.

Initial hard family. To bootstrap the hardness results of [31] on the shuffle problem (on input words) to our CSh-problem (on fixed languages), we generalize the definition of CSh to a *regular language family* $\mathcal{K}$, i.e., a (generally infinite) family of regular languages, each of which is described as a regular expression. The CSh-problem for $\mathcal{K}$, written $\mathrm{CSh}[\mathcal{K}]$, asks, given a regular expression $K \in \mathcal{K}$ and a set of strings U, whether $K \cap ⧢(U)$ is nonempty. In other words, we no longer fix one single target language but a family $\mathcal{K}$ of target languages, and the input chooses one target language from the family $\mathcal{K}$. The following is then shown in [31] by reducing from UNARY-3-PARTITION [12]:

▶ **Lemma 3.2.** **([31], Lemma 3.2)** *Let $\mathcal{K} := \{(a^ib^i)^* \mid i \in \mathbb{N}\}$. Then* $\mathrm{CSh}[\mathcal{K}]$ *is NP-hard.*

Shuffle reduction. Our goal in this section is to show the hardness of CTS and CSh for more languages, but we do not wish to prove hardness for every language from scratch. Instead, we will introduce a general tool called the *shuffle reduction* that allows us to leverage the hardness of a language K to show that another language K' is also hard. Specifically, if a language K shuffle-reduces to a language K', this will imply that there is a PTIME reduction from $\mathrm{CTS}[K]$ to $\mathrm{CTS}[K']$, and from $\mathrm{CSh}[K]$ to $\mathrm{CSh}[K']$.

The intuition for the shuffle reduction is as follows: to reduce from K to K', given an input A-DAG G, we build an A-DAG G' formed of G plus an additional directed path labeled by a word w. Thus, any topological sort σ' of G' must be the interleaving of w and of a topological sort σ of G. Now, if we require that σ' achieves K', the presence of w can

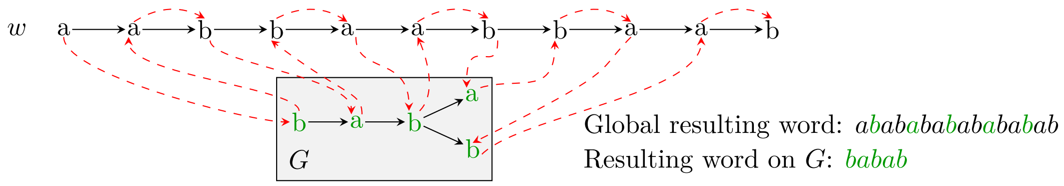

Figure 2 Example of a shuffle reduction from $K := (ba)^*b$ to $K' := (ab)^*$

impose specific conditions on σ. Intuitively, if w is sufficiently long and "far away" from all words of K', then σ' must "repair" w to a word of K' by inserting symbols from G, so the insertions performed by σ may need to be in a specific order, i.e., σ may be forced to achieve a word of K. This means that solving CTS[K'] on G' allows us to solve CTS[K] on G. This intuition is illustrated on Figure 2: to achieve a word of $K' := (ab)^*$ on the DAG G', a topological sort must enumerate elements from G to insert them at the appropriate positions in w, achieving a word of $K := (ba)^*b$. We call *filter sequence* a family of words like w that allow us to reduce any CTS[K]-instance to CTS[K']. Formally:

▶ **Definition 3.3 (Filter sequence).** Let K and K' be languages on an alphabet A. A *filter sequence* for K and K' is an infinite sequence (f_n) of words of A^* having the following property: for every $n \in \mathbb{N}$, for every word $v \in A^*$ such that $|v| = n$, we have $v \in K$ iff $(v \sqcup\!\sqcup f_n) \cap K' \neq \emptyset$.

In Figure 2, we can choose $f_5 := w$ when defining a filter sequence for $(ba)^*b$ and $(ab)^*$: indeed, if we interleave w with any DAG G of 5 vertices, then a topological sort σ of G achieves K iff some interleaving σ' of σ with w achieves K'. We can now define our reduction:

▶ **Definition 3.4 (Shuffle reduction).** We say that a language K *shuffle-reduces* to a language K' if there is a filter sequence (f_n) for K and K' such that the function $i \mapsto f_i$ is computable in PTIME (where i is given in unary).

We say that a regular language family $\mathcal{K}$ *shuffle-reduces* to K' if each K does, and if we can compute in PTIME the function $(K, i) \mapsto f_i^K$, which maps a regular expression K of $\mathcal{K}$ and an integer i in unary to the i-th word in a filter sequence (f_n^K) for K and K'.

▶ **Theorem 3.5.** *For any regular language family $\mathcal{K}$ and language K', if $\mathcal{K}$ shuffle-reduces to K' then we can reduce in PTIME from* CTS[$\mathcal{K}$] *to* CTS[K'], *and from* CSh[$\mathcal{K}$] *to* CSh[K'].

Hardness for $(ab)^*$. We now use the shuffle reduction and the language family of Lemma 3.2 to show the hardness of $(ab)^*$. This will be instrumental for our coarser dichotomy in Section 5:

▶ **Theorem 3.6.** *The problem* CSh[$(ab)^*$] *(hence* CTS[$(ab)^*$]*) is NP-hard.*

Proof sketch. We shuffle-reduce from the language family $\mathcal{K}$ of Lemma 3.2: for the language $K_B = (a^B b^B)^*$ of $\mathcal{K}$, we define the filter sequence for words of length $2Bn$ by $f_{2Bn}^B := (b^B a^B ab)^n$. This ensures that, when interleaving f_{2Bn}^B with a word v of length $2Bn$ to achieve a word of $(ab)^*$, we must use v to insert in f_{2Bn}^B the letters written in bold: $((\mathbf{a}b)^B(a\mathbf{b})^B ab)^n$. This can be done iff $v = (a^B b^B)^n$, i.e., iff $v \in K_B$. We conclude by Theorem 3.5. ◀

Other hard languages. From the hardness of $(ab)^*$, we can use the shuffle reduction to show hardness for many other languages. For instance, we can show hardness for any language u^*, where $u \in A^*$ is a word with two different letters:

▶ **Proposition 3.7.** *Let $u \in A^*$ such that $|u|_a > 0$ and $|u|_b > 0$ for $a \neq b$ in A. Then* $\mathrm{CSh}[u^*]$ *(hence* $\mathrm{CTS}[u^*]$*) is NP-hard.*

Proof sketch. We shuffle-reduce from $(ab)^*$ with the filter sequence $f_{2n} := (uu_{-a}uu_{-b}u)^n$, where u_{-a} (resp. u_{-b}) is u but removing one occurrence of a (resp. of b). If a word v with $|v| = 2n$ has an interleaving w with f_{2n} that falls in u^*, then in w we must intuitively insert one a from v in each u_{-a} and one b from v in each u_{-b}, so that $v = (ab)^n$. To formalize this, we first rotate u to ensure that its first and last letters are different. We then observe that, as w is in u^*, any factor w' of length $|u|$ of w must be such that $|w'|_a = |u|_a$ and $|w'|_b = |u|_b$. We then consider factors of w of length $|u|$ centered on the u_{-a} and u_{-b} in f_{2n}: we argue that in w we must have inserted at least one a in or around each u_{-a}, and at least one b in or around each u_{-b}, otherwise these factors do not have enough a's and enough b's. ◀

We can also use the shuffle reduction to show hardness for other languages, e.g., $(aa+bb)^*$:

▶ **Proposition 3.8.** *Let $L := (aa + bb)^*$. The problem* $\mathrm{CSh}[L]$ *(hence* $\mathrm{CTS}[L]$*) is NP-hard.*

Proof sketch. We do again a shuffle reduction from $(ab)^*$, with the filter sequence $f_{2n} = (ab)^n$. If a word v with $|v| = 2n$ is such that $v \sqcup\!\sqcup f_{2n}$ intersects $(aa + bb)^*$ nontrivially, it must intuitively insert a's and b's in f_{2n} alternatively, so it must be $(ab)^n$. Note that a similar proof would also show hardness for the language $(a^i + b^j)^*$ for any choice of $i, j \geq 2$. ◀

We show a last result that does not use the shuffle reduction but an easy consideration on the number of letter occurrences. This result will be useful in Section 5:

▶ **Proposition 3.9.** *The problem* $\mathrm{CSh}[(ab + b)^*]$ *(hence* $\mathrm{CTS}[(ab + b)^*]$*) is NP-hard.*

Proof. We describe an easy PTIME reduction from $\mathrm{CSh}[(ab)^*]$ to $\mathrm{CSh}[(ab + b)^*]$. Given an instance I, check if the number of a-labeled and b-labeled vertices is the same, and fail if it is not. Otherwise, then I achieves a word of $(ab + b)^*$ iff it achieves one of $(ab)^*$, because we must enumerate one a-labeled vertex with each b-labeled vertex. ◀

We believe that the shuffle reduction applies to many other languages, though we do not know how to characterize them. In particular, we believe that the following could be shown with the shuffle reduction, generalizing all the above hardness results except Proposition 3.8:

▶ **Conjecture 3.10.** *Let F be a finite language such that, for some letter $a \in A$, the language F contains no power of a but contains a word which contains a. Then* $\mathrm{CSh}[F^*]$ *is NP-hard.*

4 Tractability Results

Having shown hardness for several languages, we now present our tractability results. We will also rely on some of these results to show our coarser dichotomy result in the next section.

Closure under union. The first observation on tractable languages is that they are closed under union, as follows (recalling the definition of CTS and CSh for language *families*):

▶ **Lemma 4.1.** *For any finite family of languages $\mathcal{K}$, there is a logspace reduction from* $\mathrm{CTS}[\bigcup \mathcal{K}]$ *to* $\mathrm{CTS}[\mathcal{K}]$*, and likewise from* $\mathrm{CSh}[\bigcup \mathcal{K}]$ *to* $\mathrm{CSh}[\mathcal{K}]$*.*

Proof. To solve a problem for the language $\bigcup \mathcal{K}$ on an input instance I, simply enumerate the languages $K' \in \mathcal{K}$, and solve the problem on I for each K'. Clearly I is a positive instance of the problem for $\bigcup \mathcal{K}$ iff I is a positive instance of the problem for one of the K'. ◀

▶ **Corollary 4.2.** *For any finite family of languages $\mathcal{K}$, if* $\mathrm{CTS}[K']$ *is in NL for each* $K' \in \mathcal{K}$, *then so is* $\mathrm{CTS}[\bigcup \mathcal{K}]$. *The same is true of the CSh-problem.*

Clearly, tractability is also preserved under the *reverse operator*, i.e., reversing the order of words in a language; however tractable languages are *not* closed under many usual operators, as we will show in Section 5. Still, closure under union will often be useful in the sequel.

Monomials. We will now show that CTS is tractable for an important family of languages (and unions of such languages): the *monomial* languages. Having fixed the alphabet A, a *monomial* is a language of the form $A_1^* a_1 A_2^* a_2 \cdots a_n A_{n+1}^*$ with $a_i \in A$ and $A_i \subseteq A$ for all i. In particular, we may have $A_i = \emptyset$ so that $A_i^* = \epsilon$: hence, for every word $u \in A^*$, the language $A^* u A^*$ is a monomial language, which intuitively tests whether a word contains the pattern u. Several decidable algebraic and logical characterizations of these languages are known; in particular, unions of monomials are exactly the languages that are definable in the first-order logic fragment $\Sigma_2[<]$ of formulas with quantifier prefix $\exists^*\forall^*$, and it is decidable to check if a regular language is in this class [23, 21]. We show:

▶ **Theorem 4.3.** *For any monomial language K, the problem* $\mathrm{CTS}[K]$ *is in NL.*

Proof sketch. Let K be $A_1^* a_1 A_2^* a_2 \cdots A_n^* a_n A_{n+1}^*$. We can first guess in NL the vertices $v_1, \ldots, v_n$ to which the $a_1, \ldots, a_n$ are mapped, so all that remains is to check, for each such guess, whether we can match the remaining vertices to the A_i. We proceed by induction on n. The base case of $n = 0$ (i.e., $K = A_1^*$) is trivial. For the induction step, using the fact that NL = co-NL (see [14, 27]), we check that the descendants of the last element v_n are all in A_{n+1}^*, and then we compute the set S of vertices that *must* be enumerated before v_n: they are the ancestors of the v_i, and the ancestors of any vertex labeled by a letter in $A \setminus A_{n+1}$. We then use the induction hypothesis to check in NL whether S has a topological sort that achieves a word in $A_1^* a_1 \ldots A_{n-1}^* a_{n-1} A_n^*$. ◀

Tractability based on width. While unions of monomials are a natural class, it turns out that they do not cover all tractable languages. In particular, we can show:

▶ **Proposition 4.4.** *Let $A := \{a, b\}$ and $K := (ab)^* + A^* aa A^*$. The problem* $\mathrm{CTS}[K]$ *(hence* $\mathrm{CSh}[K]$*) is in NL.*

This result is not covered by Theorem 4.3, because we can show that K cannot be expressed as a union of monomials (see Appendix C.1); and the proof technique is different.

Proof. Let G be an input A-DAG. We first check in NL if G contains two incomparable vertices $v_1 \neq v_2$ such that $\lambda(v_1) = \lambda(v_2) = a$. If yes, we conclude that G is a positive instance, as we can clearly achieve K by enumerating v_1 and v_2 contiguously.

If there are no two such vertices, we check in NL if there are two comparable a-labeled vertices $v_1 \neq v_2$ that can be enumerated contiguously, i.e., there is an edge $v_1 \to v_2$ but no vertex w that is *between* v_1 and v_2, i.e., is a descendant of v_1 and an ancestor of v_2. If there are two such vertices v_1 and v_2, we conclude again that G is a positive instance.

Otherwise, our first test implies that G induces a total order on the a-labeled vertices, and our second test implies that any two consecutive a-labeled vertices in this order must have at least one b-labeled vertex between them. This ensures that no topological sort achieves $A^* aa A^*$, so it suffices to test whether one can achieve $(ab)^*$. Clearly this is the case iff all consecutive pairs of a-labeled vertices have exactly one b-labeled vertex between them, and there is exactly one additional b-labeled vertex that can be enumerated after the last a-labeled vertex. We can test this in NL, which concludes the proof. ◀

Intuitively, the language of Proposition 4.4 is tractable because it is easy to solve unless the input instance has a very restricted structure, namely, all a's are comparable. We do not know whether this result generalizes to $(ab)^* + A^*a^iA^*$ for $i > 2$. However, following the intuition of this proof, we can show the tractability of a similar kind of regular languages:

▶ **Proposition 4.5.** *Let $A := \{a, b\}$, let K' be a regular language, let $i \in \mathbb{N}$, and let $K := K' + A^*(a^i + b^i)A^*$. The problem* CTS$[K]$ *(hence* CSh$[K]$*) is in NL.*

As in Proposition 4.4, CTS is trivial for the languages in this proposition unless the input A-DAG G has a restricted shape. Here, the requirement is on the *width* of G, i.e., the maximal cardinality of a subset of pairwise incomparable vertices (called an *antichain*), so we can show Proposition 4.5 by distinguishing two cases depending on the width of G:

Proof sketch. We test in NL whether the input A-DAG G contains an antichain C of size $2i$: if it does, then at least i vertices in C must have the same label, and we can enumerate them in succession to achieve $A^*a^iA^*$ or $A^*b^iA^*$, so G is a positive instance. Otherwise, G has width $< 2i$, and Dilworth's theorem [8] implies that its elements can be partitioned into chains, so that CTS can be solved in NL following a dynamic algorithm on them. ◀

Other tractable case. We close the section with another example of a regular language which is tractable for the CSh-problem for what appears to be a unrelated reason.

▶ **Proposition 4.6.** *Let $A := \{a, b\}$ and $K := (aa + b)^*$. The problem* CSh$[K]$ *is in NL.*

This is in contrast to $(aa + bb)^*$, for which we showed intractability (Proposition 3.8). We do not know the complexity of the CTS-problem for $(aa + b)^*$, or the complexity for either problem of languages of the form $(a^i + b)^*$ for $i > 2$.

Proof sketch. We show that the existence of a suitable topological sort can be rephrased to an NL-testable equivalent condition, namely, there is no string in the input instance whose number of odd "blocks" of a-labeled elements dominates the total number of a-labeled elements available in the other strings. If the condition fails, then we easily establish that no suitable topological sort can be constructed: indeed, eliminating each odd block of a's in the dominating string requires one a from the other strings. If the condition holds, we can simplify the input strings and show that a greedy algorithm can find a topological sort by picking pairs of a's in the two current heaviest strings. ◀

5 A Coarser Dichotomy Theorem

In the two previous sections, we have established some intractability and tractability results about the constrained topological sort and constrained shuffle problems for various languages. Remember that our end goal would be to characterize the tractable and intractable languages, and show a dichotomy (Conjecture 2.3). This is difficult, and one reason is that the class of tractable languages is not "well-behaved": while it is closed under the union operator (Corollary 4.2), it is is *not* closed under intersection, complement, and other common operations. This makes it difficult to study tractable languages using algebraic language theory [22].

▶ **Proposition 5.1.** *We have the following counterexamples to closure:*

- ***Quotient.*** *There exists a word $u \in A^*$ and a regular language K such that* CSh$[K]$ *is in NL but* CSh$\left[u^{-1}K\right]$ *is NP-hard.*

- ***Intersection.*** *There exists two regular languages* K_1 *and* K_2 *such that* $\mathrm{CTS}[K_1]$ *and* $\mathrm{CTS}[K_2]$ *are both in PTIME but* $\mathrm{CSh}[K_1 \cap K_2]$ *is NP-hard*
- ***Complement.*** *There exists a regular language* K *such that* $\mathrm{CTS}[K]$ *is in NL, but* $\mathrm{CSh}[A^* \setminus K]$ *is NP-hard.*
- ***Inverse of morphism.*** *There exists a regular language* K *and morphism* φ *such that* $\mathrm{CTS}[K]$ *is in NL but* $\mathrm{CSh}\big[\varphi^{-1}(K)\big]$ *is NP-hard.*

The three last results of this proposition also apply to the constrained topological sort problem, but the first one does not, and in fact CTS-tractable languages *are* closed under quotients. This observation implies that there are regular languages K such that $\mathrm{CSh}[K]$ is tractable but $\mathrm{CTS}[K]$ is NP-hard; one concrete example is $K := b^*A^* + aaA^* + (ab)^*$ (see Appendix D.1). We sketch the proof of Proposition 5.1:

Proof sketch. For each operation, we use $(ab)^*$ as our NP-hard language (by Theorem 3.6).

For quotient, we take $K := bA^* + aaA^* + (ab)^*$, and $u := ab$. We have $u^{-1}K = (ab)^*$, but $\mathrm{CSh}[K]$ is in NL because any shuffle instance with more than one string satisfies K.

For intersection, we take $K_1 := (ab)^*(\epsilon + bA^*)$ and $K_2 := (ab)^*(\epsilon + aaA^*)$. We have $K_1 \cap K_2 = (ab)^*$, but $\mathrm{CSh}[K_1]$ and $\mathrm{CSh}[K_2]$ are in PTIME using an ad-hoc greedy algorithm.

For complement, we take $K := bA^* \cup A^*a \cup A^*aaA^* \cup A^*bbA^*$. As K is a union of monomials, we know by Theorem 4.3 that $\mathrm{CTS}[K]$ is in NL, but we have $A^* \setminus K = (ab)^*$.

For inverse of morphism, we take $A := \{a, b\}$ and $K := (ab)^* + A^*(a^3 + b^3)A^*$. We know that $\mathrm{CTS}[K]$ is in PTIME by Proposition 4.5. Now, defining $\varphi : A^* \to A^*$ by $\varphi(a) := aba$ and $\varphi(b) := bab$, we have $\varphi^{-1}(K) = (ab)^*$ because no word in the image of φ has three identical consecutive symbols. ◀

Proposition 5.1 suggests that tractable languages would be easier to study algebraically if we ensured that they were closed under all these operations, i.e., if they formed a variety [22]. In this section, we enforce this by moving to an alternative phrasing of the CTS and CSh problems. This allows us to leverage algebraic techniques and show a dichotomy theorem in this alternative phrasing, under an additional *counter-free* assumption. We first present the alternative phrasing, and then present the additional assumption and our dichotomy result.

Alternative phrasing. The first change in our alternative phrasing is that the input DAG G will now be an A^**-DAG*, i.e., a DAG labeled with words of A^* rather than letters of A. As before, a topological sort σ of G *achieves* a word $\lambda(\sigma) \in A^*$ obtained by concatenating the λ-images of the vertices of G in the order of σ: but vertex labels are now "atomic" words whose letters cannot be interleaved with anything else. The *multi-letter* CTS and CSh problems are the variants defined with A^*-DAGs; intuitively, this ensures that tractable languages are closed under inverse morphisms.

The second change is that we will not fix one single target language, but a *semiautomaton* [13], i.e., an automaton where initial and final states are not specified. Formally, a semiautomaton is a tuple (Q, A, δ) where Q is the set of states, A is the alphabet, and $\delta : Q \times A \to Q$ is the transition function; we extend δ to words as usual by setting $\delta(q, \epsilon) := q$ and $\delta(q, u_1 \cdots u_{n+1}) := \delta(\delta(q, u_1), u_2 \cdots u_{n+1})$. We will fix the target semiautomaton, and the initial and final states will be given in the input instance (in addition to the DAG). This enforces closure under quotients (by choosing the initial and final states) and complement (by toggling the final states). Further, to impose closure under intersection, the input instance will specify a *set* of pairs of initial-final states, with a logical AND over them. The question is to determine whether the input DAG achieves a word accepted by *all* the corresponding automata; and this enforces closure under intersection.

We can now summarize the formal definition of our problem variants. The *multi-letter* CTS-*problem for a fixed semiautomaton* $S = (Q, A, \delta)$ takes as input an A^*-DAG and a set $\{(i_1, F_1), \ldots, (i_k, F_k)\}$ of initial-final state pairs, where $i_j \in Q$ and $F_j \subseteq Q$ for all $1 \leq j \leq k$. The input is accepted if there is a topological sort σ of G such that, for all $1 \leq j \leq k$, the word $\lambda(\sigma)$ is accepted by the automaton (Q, A, δ, i_j, F_j), i.e., $\delta(i_j, \lambda(\sigma)) \in F_j$. The *multi-letter* CSh-*problem for a fixed semiautomaton* is defined in the same way, imposing that the input A^*-DAG is a union of directed path graphs.

Dichotomy result. Our dichotomy will apply to the multi-letter CTS and CSh problem for semiautomata. However, we will need to make an additional assumption, namely, that the semiautomaton is *counter-free*. This assumption means that our dichotomy will only apply to a well-known subset of regular languages, namely, the *star-free languages*, that are better understood algebraically; it excludes in particular the tricky case of *group languages* that we will study separately in Section 6. Formally, a semiautomaton is *counter-free* if, for every state q and word $u \in A^*$, if $\delta(q, u^n) = q$ for some $n > 1$, then we have $\delta(q, u) = q$. Under the counter-free assumption, we can prove the following dichotomy, using our hardness and tractability results in Sections 3 and 4:

▶ **Theorem 5.2.** *Let S be a counter-free semiautomaton. Then the multi-letter CSh-problem and CTS-problem for S are either both in NL, or both NP-complete. The dichotomy is effective: given S, it is PSPACE-complete to decide which case applies.*

We conclude the section by introducing some technical tools used for this result and for Section 6, and by giving a proof sketch. The criterion of the dichotomy on S is phrased in terms of the *transition monoid* of S, which we now define (see, e.g., [22] for details). Remember that a monoid is a set that has an associative binary operation and a neutral element. The *transition monoid* $T(S)$ of a semiautomaton $S = (Q, A, \delta)$ is the set of functions $f : Q \to Q$ that are "achieved" by S in the following sense: there is a word $u \in A^*$ such that $\delta(q, u) = f(q)$ for all $q \in Q$. In particular, the neutral element is the identity function, which is achieved by taking $u := \epsilon$; and the binary operation on $T(S)$ is function composition, which is associative. Note that the transition monoid is finite and can be computed from S.

We assumed that S is counter-free, and this is equivalent [19] to saying that $T(S)$ is in the class $\mathbf{A}$ of *aperiodic* finite monoids (formally defined by the equation $x^{\omega+1} = x^{\omega}$ where ω is the idempotent power [22] of the monoid). Within $\mathbf{A}$, our dichotomy criterion on $T(S)$ is based on a certain subclass of $\mathbf{A}$, called $\mathbf{DA}$ (see [29]): S is tractable iff $T(S)$ is in $\mathbf{DA}$, and it is PSPACE-complete [30] to test whether this holds (using the formal definition of $\mathbf{DA}$ by the equation $(xy)^{\omega}x(xy)^{\omega} = (xy)^{\omega}$). We can now sketch the proof of Theorem 5.2:

Proof sketch. We first show that if $T(S)$ is in $\mathbf{DA}$ then the multi-letter CTS and CSh problems for S are in NL. For this, we rely on one characterization of $\mathbf{DA}$ (from [29]): if $T(S)$ is in $\mathbf{DA}$ then the regular languages recognized by S (for any set of initial-final states) are unions of *unambiguous monomials*, in particular they are unions of monomials, so we have tractability by Corollary 4.2 and Theorem 4.3.

For the converse direction, we use a second characterization of $\mathbf{DA}$ (from [28]): if $T(S)$ is *not* in $\mathbf{DA}$ then there is a choice of initial-final state pairs for which S computes a language K whose inverse image by some morphism is either $(ab)^*$ or $(ab + b)^*$. We know that these languages are intractable (Theorem 3.6 and Proposition 3.9) so we conclude by showing a PTIME reduction from one of these two languages: this is possible in our alternative problem phrasing, in particular using the multi-letter labels to invert the morphism. ◀

6 Lifting the Counter-Free Assumption for CSh

Our dichotomy theorem in the previous section (Theorem 5.2) was shown for an alternative phrasing of our problems (with semiautomata and multi-letter inputs), and made the additional assumption that the input semiautomaton is counter-free. In this section, we study how to lift the counter-free assumption. In exchange for this, we restrict our study to the constrained shuffle problem (CSh) rather than CTS.

To extend Theorem 5.2 for the CSh-problem, we will again classify the semiautomata S based on their transition monoid $T(S)$. However, instead of $\mathbf{DA}$, we will use the two classes $\mathbf{DO}$ and $\mathbf{DS}$ introduced in [26] (formally $\mathbf{DO}$ is defined by the equation $(xy)^\omega(yx)^\omega(xy)^\omega = (xy)^\omega$ and $\mathbf{DS}$ by the equation $((xy)^\omega(yx)^\omega(xy)^\omega)^\omega = (xy)^\omega$ for ω the idempotent power). Both $\mathbf{DO}$ and $\mathbf{DS}$ are supersets of $\mathbf{DA}$, specifically we have $\mathbf{DA} \subseteq \mathbf{DO} \subseteq \mathbf{DS}$, and we can test in PSPACE in S whether $T(S)$ is in each of these classes [30]. Our main result is then:

▶ **Theorem 6.1.** *Let S be a semiautomaton. If $T(S)$ is in $\mathbf{DO}$, then the multi-letter CSh-problem for S is in NL. If $T(S)$ is not in $\mathbf{DS}$, then it is NP-complete.*

This result generalizes Theorem 5.2 for the CSh-problem, because both $\mathbf{DO}$ and $\mathbf{DS}$ collapse to $\mathbf{DA}$ for aperiodic monoids (see [26] and [2, Chapter 8]); formally, $\mathbf{DO} \cap \mathbf{A} = \mathbf{DS} \cap \mathbf{A} = \mathbf{DA}$. However, $\mathbf{DO}$ covers more languages than $\mathbf{DA}$: the main technical challenge to prove Theorem 6.1 is to show that CSh is tractable for these languages. One important example are the *group languages* over A: these are the regular languages recognized, for some choice of initial-final state pairs, by a semiautomaton S over A such that $T(S)$ is a group. A more general example are *district group monomials*, which are the languages of the form $K_1 a_1 \cdots K_n a_n K_{n+1}$ where, for all i, we have $a_i \in A$ and K_i is a group language over some alphabet $A_i \subseteq A$. Note that district group monomials are more expressive than the *group monomials* defined in earlier work [24] (which set $A_i := A$ for all i), and they also generalize the monomials that we studied in Section 4 (any A_i^* is trivially a group language over A_i, even though it is not a group language over A). In fact, to prove Theorem 6.1, what we need is to generalize Theorem 4.3 (for CSh) from monomials to district group monomials:

▶ **Theorem 6.2.** *Let K be a district group monomial. Then* $\mathrm{CSh}[K]$ *is in NL.*

Note that this theorem, like Theorem 4.3, applies to the original phrasing of CSh, not the alternative phrasing with semiautomata and multi-letter DAGs. Thus, Theorem 6.2 implies that the original CSh-problem is tractable for many languages that we had not covered previously, e.g., $(ab^*a + b)^*c(ba^*b + a)^*$, the language testing whether there is one c preceded by an even number of a and followed by an even number of b. The proof of Theorem 6.2 is our main technical achievement, and we sketch it below (see Appendix E for details):

Proof sketch. We focus on the simpler case of a group language, for a finite group H. The problem can be rephrased directly in terms of H: given a tuple I of strings over H and a target element $g \in H$, determine if there is an interleaving of I that evaluates to g under the group operation. Our approach partitions H into the *rare* elements H_{rare}, that occur in a constant number of strings, and the *frequent* elements H_{freq}, that occur in sufficiently many strings. For the frequent elements, we can build a large antichain C from the strings where they occur, with each element of H_{freq} occuring many times in C. Now, as topological sorts can choose any order on C, they can intuitively achieve all elements of the subgroup $\langle H_{\mathrm{freq}}\rangle$ generated by H_{freq}, except that they cannot change "commutative information", e.g., the parity of the number of elements. We formalize the notion of "commutative information" using relational morphisms, and prove an *antichain lemma* that captures our intuition that all elements of $\langle H_{\mathrm{freq}} \rangle$ with the right commutative information can be achieved.

For the rare elements, we can simply follow a dynamic algorithm on the constantly many strings where they occur. However, we must account for the possibility of inserting elements of $\langle H_{\mathrm{freq}} \rangle$ from the other strings, and we must show that it suffices to do constantly many insertions, so that it was sufficient to impose a constant lower bound on $|C|$. We formalize this as an *insertion lemma*, which we prove using Ramsey's theorem. ◀

We close the section by commenting on the two main limitations of Theorem 6.1. The first limitation is that it is not a dichotomy: it does not cover the semiautomata with transition monoid in $\mathbf{DS} \setminus \mathbf{DO}$. We do not know if the corresponding languages are tractable or not; we have not identified intractable cases, but we can show tractability, e.g., for $(a^+b^+a^+b^+)^*$, the language of words with an even number of subfactors of the form a^+b^+.

▶ **Proposition 6.3.** *Let* $K = (a^+b^+a^+b^+)^*$*. Then* $\mathrm{CSh}[K]$ *is in NL.*

However, it would be difficult to show tractability for all of **DS**, because **DS** is still poorly understood in algebraic language theory. For instance, characterizing the languages with a syntactic monoid in **DS** has been open for over 20 years [2, Open problem 14, page 442].

The second limitation of Theorems 6.1 and 6.2 is that they only apply to CSh. New problems arise with CTS: for instance, an $\{a, b\}$-DAG G may contain large antichains C_a and C_b of a-labeled and b-labeled vertices, and yet contain no antichain with many a-labeled and b-labeled vertices (e.g., if G is the series composition of C_a and C_b). The missing proof ingredient seems to be an analogue of Dilworth's theorem for *labeled* DAGs (see also [3]).

7 Conclusion and Open Problems

We have studied the complexity of two problems, constrained topological sort (CTS) and constrained shuffle (CSh): fixing a regular language K, given a labeled DAG (for CTS) or a tuple of strings (for CSh), we ask if the input DAG has a topological sort achieving K. We have shown tractability and intractability for several regular languages using a variety of techniques. These results yield a coarser dichotomy (Theorem 5.2) in an alternate problem phrasing that imposes some closure assumptions.

Our work leaves the main dichotomy conjecture open (Conjecture 2.3). Even in the alternate problem phrasing of Theorem 5.2, our dichotomy only covers counter-free semiautomata: the restriction is lifted in Section 6 but only for CSh, and with a gap between tractability and intractability. In the original phrasing, there are many concrete languages that we do not understand: Does Proposition 4.4 extend to $(ab)^* + A^*a^iA^*$ for $i > 2$? Does Proposition 4.6 extend to $(a^i + b)^*$ for $i > 2$, or to CTS rather than CSh? Can we show Conjecture 3.10?

Another direction would be to connect CSh and CTS to the framework of *constraint satisfaction problems* (CSP) [10], which studies the complexity of homomorphism problems for fixed "constraints" (right-hand-side of the homomorphism). If this were possible, it could lead to a better understanding of our tractable and hard cases. However, CTS does not seem easy to rephrase in CSP terms: topological sorts and regular language constraints seems hard to express in terms of homomorphisms, even in extensions such as *temporal* CSPs [5, 6].

One last question would be to investigate CTS and CSh for *non-regular* languages. The simplest example is the Dyck language, which appears to be NP-hard for CTS (at least in the multi-letter setting), but tractable for CSh, via a connection to scheduling; see [11], problem SS7. More generally, CTS and CSh could be studied, e.g., for context-free languages, where the complexity landscape may be equally enigmatic.

References

1 Kunal Agrawal, Jing Li, Kefu Lu, and Benjamin Moseley. Scheduling parallel DAG jobs online to minimize average flow time. In *Proc. SODA*, 2016.

2 J. Almeida. *Finite Semigroups and Universal Algebra*. Series in algebra. World Scientific, 1994.

3 Antoine Amarilli. Generalization of Dilworth's theorem for labeled DAGs, 2016. `https://cstheory.stackexchange.com/q/37062`.

4 Antoine Amarilli, M. Lamine Ba, Daniel Deutch, and Pierre Senellart. Possible and certain answers for queries over order-incomplete data. In *Proc. TIME*, 2017.

5 Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *JACM*, 57(2):9, 2010.

6 Manuel Bodirsky, Barnaby Martin, and Antoine Mottet. Discrete temporal constraint satisfaction problems, 2015. `arXiv:1503.08572`.

7 Sam Buss and Michael Soltys. Unshuffling a square is NP-hard. *JCSS*, 80(4), 2014.

8 Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 1950.

9 Joey Eremondi, Oscar H Ibarra, and Ian McQuillan. On the complexity and decidability of some problems involving shuffle, 2016. `arXiv:1606.01199`.

10 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM J. Comput.*, 28(1), 1998.

11 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

12 Michael R. Garey and David S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM J. Comput*, 1975.

13 M. Holcombe. *Algebraic Automata Theory*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1982.

14 Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput*, 17(5), 1988.

15 David S. Johnson. The NP-completeness column: an ongoing guide. *Journal of Algorithms*, 5(2), 1984.

16 John Kececioglu and Dan Gusfield. Reconstructing a history of recombinations from a set of sequences. *Discrete Applied Mathematics*, 88(1-3), 1998.

17 Takayuki Kimura. An algebraic system for process structuring and interprocess communication. In *Proc. STOC*, 1976.

18 Anthony Mansfield. On the computational complexity of a merge recognition problem. *Discrete Applied Mathematics*, 5(1), 1983.

19 Robert McNaughton and Seymour Papert. *Counter-Free Automata*. MIT Press, 1971.

20 W. F. Ogden, W. E. Riddle, and W.C. Rounds. Complexity of expressions allowing concurrency. In *Proc. POPL*, 1978.

21 Charles Paperman. Semigroup online, 2018. `https://www.paperman.name/semigroup/`.

22 Jean-Éric Pin. Syntactic semigroups. In *Handbook of formal languages, Vol. 1*, pages 679–746. Springer, Berlin, 1997.

23 Jean-Éric Pin and Pascal Weil. Polynominal closure and unambiguous product. *TCS*, 30(4), 1997.

24 Jean-Éric Pin. Polynomial closure of group languages and open sets of the Hall topology. *TCS*, 169(2), 1996.

25 Romeo Rizzi and Stéphane Vialette. On recognizing words that are squares for the shuffle product. *TCS*, 2017.

26 M. P. Schützenberger. Sur le produit de concaténation non ambigu. *Semigroup forum*, 13, 1976/77.

27 Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3), 1988.

28 Pascal Tesson and Denis Thérien. The computing power of programs over finite monoids. *J. Autom. Lang. Comb.*, 7(2), 2001.

29 Pascal Tesson and Denis Thérien. Diamonds are forever: the variety DA. *Semigroups, algorithms, automata and languages*, 1, 2002.

30 Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *Proc. STOC*, 1998.

31 Manfred K. Warmuth and David Haussler. On the complexity of iterated shuffle. *JCSS*, 28(3), 1984.

A Proofs for Section 2 (Problem Statement and Main Results)

▶ **Proposition 2.2.** *For any language K, the problems* $\mathrm{CTS}[K]$ *and* $\mathrm{CSh}[K]$ *are in* NP^K*.*

Proof. As explained in the main text, we guess a permutation σ of the input vertices, check that it respects the order constraints, and use the oracle for the word problem to check that the word achieved by σ is in K. ◀

B Proofs for Section 3 (Hardness Results)

B.1 Proof of Proposition 3.1: Direct Hardness Result

▶ **Proposition 3.1.** *Let $K_0 := (a_1 a_2 + b_1 b_2)^*$. The problem* $\mathrm{CSh}[K_0]$ *is NP-hard.*

Proof. We construct the CSh-instance I in PTIME from the input instance to the shuffle problem as explained in the proof sketch, and argue for correctness in more detail. It is clear that, to achieve K_0, a topological sort σ of I must enumerate alternatively a letter with subscript 1 and a letter with subscript 2, so it must enumerate alternatively from w_1 and from U_2, and the definition of K_0 ensures that the two letters in w and U corresponding to the enumerated letters in w_1 and U_2 must have the same label in $\{a, b\}$. Hence, considering the restriction σ' of σ to U_2, the interleaving of U that corresponds to σ' witnesses that $w \in ⧢(U)$.

Conversely, if $w \in ⧢(U)$, starting from a witnessing topological sort σ' of U_2, it is clear that we can construct a topological sort σ of I that achieves K_0, by enumerating the elements of w_1 alternatively with the elements of U_2 according to σ'. This shows correctness and concludes the proof. ◀

B.2 Proof of Lemma 3.2: Initial Hard Family

▶ **Lemma 3.2.** **([WH84], Lemma 3.2)** *Let $\mathcal{K} := \{(a^i b^i)^* \mid i \in \mathbb{N}\}$. Then* $\mathrm{CSh}[\mathcal{K}]$ *is NP-hard.*

Proof. For completeness, we summarize here the proof of [WH84]: see the statement and proof of Lemma 3.2 in [WH84] for details. The reduction is from UNARY-3-PARTITION: given a tuple E of $3m$ positive integers written in unary, such that $B := \frac{1}{m}\left(\sum_{1\le i\le 3m} n_i\right)$ is an integer, and such that $B/4 < e < B/2$ for each $e \in E$, decide whether E can be partitioned into m triples, with each triple summing to B. This problem is NP-hard [GJ75]. Given a UNARY-3-PARTITION instance (E, B), we create a CSh instance I by writing each integer n as the string $a^n b^n$, and we choose the target language K in $\mathcal{K}$ to be $(a^B b^B)^*$, which is clearly a PTIME reduction. Clearly, if (E, B) can be partitioned in triples summing to B, then we can define a topological sort of I by enumerating, for each triple, the B copies of the a's in that triple, and then the b's, achieving a word of K. Conversely, any topological sort achieving a word of K must start by enumerating B copies of a's followed by the same number of b's, and the only way to free sufficiently many b's is to enumerate completely the initial a segments of some strings: we know that the number of such strings is exactly 3 by our assumption that $B/4 < e < B/2$ for all $e \in E$. Hence, by applying this argument repeatedly, a topological sort of I achieving K must define a solution to (E, B), completing the proof of the reduction. ◀

B.3 Proof of Theorem 3.5: Shuffle Reduction

▶ **Theorem 3.5.** *For any regular language family $\mathcal{K}$ and language K', if $\mathcal{K}$ shuffle-reduces to K' then we can reduce in PTIME from* $\mathrm{CTS}[\mathcal{K}]$ *to* $\mathrm{CTS}[K']$*, and from* $\mathrm{CSh}[\mathcal{K}]$ *to* $\mathrm{CSh}[K']$*.*

Of course, note that this result also applies to languages and not just to language families, because we can always take $\mathcal{K}$ to be a singleton language family containing only one single language.

Proof. We show the result for the CSh-problem; the result for the CTS-problem is shown in exactly the same way. Fix the family $\mathcal{K}$ and language K'. Let K be the input language of $\mathcal{K}$, and let I be an input instance of the CSh-problem for K. Let (f_n^K) be the filter sequence for K and K'. Letting $n := |I|$, let us call I' the instance of the CSh-problem for K' that contains I and a separate string labeled with f_n^K: by our computability hypothesis on (f_n^K), this is computable in PTIME. We now argue that I' is a positive instance to the CSh-problem for K' iff I is a positive instance to the CSh-problem for K. Indeed, assuming that there is a word v of K achieved by a topological sort σ of I, we have $|v| = n$ by definition, so by definition of (f_n^K) we have $(v \sqcup\!\sqcup f_n^K) \cap K' \neq \emptyset$. Hence, let v' be an element of this set. It is in $v \sqcup\!\sqcup f_n^K$, so it can be obtained as a topological sort of I' by shuffling f_n^K with σ, and it is in K' so it witnesses that I' is a positive instance to the CSh-problem for K'.

Conversely, if there is a topological sort σ' of I' achieving a word $v' \in K'$, then σ' defines a topological sort σ of I achieving a word v such that $v' \in v \sqcup\!\sqcup f_n$. As we have $|v| = n$ by definition, and v' witnesses that $v \sqcup\!\sqcup f_n$ is non-empty, we must have $v \in K$, so that σ witnesses that I is a positive instance to the CSh-problem for K. This establishes correctness, and concludes the proof. ◀

Note that, for simplicity, we have shown Theorem 3.5 for PTIME reductions. This is because we only use the shuffle reduction in this paper to prove NP-hardness results. However, Theorem 3.5 result can also be shown for NL reductions if we further assume that the filter sequences can be computed in logspace, i.e., the function mapping the unary representation of n to the word f_n is computable by a logspace transducer.

B.4 Proof of Theorem 3.6: Hardness of $(ab)^*$

▶ **Theorem 3.6.** *The problem* $\mathrm{CSh}[(ab)^*]$ *(hence* $\mathrm{CTS}[(ab)^*]$*) is NP-hard.*

Proof. Let $\mathcal{K}$ be the family of regular languages defined in Lemma 3.2. We define a filter sequence (f_n^B) for each such language $K_B = (a^B b^B)^*$ for $B \in \mathbb{N}$. We first explain how to define the filter sequence for word lengths of the form $2Bn$, for which there are words in K_B having the specified length. For such lengths, we define $f_{2Bn}^B := (b^B a^B ab)^n$. For other word lengths $n' \in \mathbb{N}$, for which there are no words in K_B, we define $f_{n'}^B := a^{n'+1}$: this ensures that we can never obtain a word of K_B by interleaving n' symbols with $f_{n'}^B$, which is correct. The filter sequence is clearly computable in PTIME. So we only have to show that, for all $n \in \mathbb{N}$, the word f_{2Bn}^B is a filter sequence for word length $2Bn$.

To do so, fix $n \in \mathbb{N}$, and consider a word v of length $2Bn$ in K_B. For the forward direction, if v is $(a^B b^B)^n$ which is the only word of K_B of length $2Bn$, then we can interleave v with f_{2Bn}^B to form a word of $(ab)^*$ by inserting the letters in bold: $((\mathbf{a}b)^B(a\mathbf{b})^B ab)^n$.

Conversely, for the backward direction, we are forced to insert at least these letters. More precisely, considering an interleaving of f_{2Bn}^B with a word v that achieves a word w of $(ab)^*$, we know that, in each of the n occurrences of $b^B a^B ab$ in v, each of the B first b's must be preceded by an a in w (so B insertions of a), then each of the B occurrences of a must be followed by a b in w (so B insertions of b). As we have $|v| = 2Bn$, we must perform these insertions in this order, and as they do not overlap, this completely specifies v: so we know that $(v \sqcup\!\sqcup f^B_{2Bn}) \cap (ab)^*$ is non-empty iff $v = (a^B b^B)^n$. This shows that f^B_{2Bn} is indeed a filter sequence, which establishes that $\mathrm{CSh}[(ab)^*]$ is NP-hard thanks to Theorem 3.5 and Lemma 3.2. ◀

B.5 Hardness Proofs for Other Languages

▶ **Proposition 3.7.** *Let $u \in A^*$ such that $|u|_a > 0$ and $|u|_b > 0$ for $a \neq b$ in A. Then $\mathrm{CSh}[u^*]$ (hence $\mathrm{CTS}[u^*]$) is NP-hard.*

Proof. Fix $u \in A^*$ and the two witnessing letters a and b. We first make a straightforward preliminary observation: for any word w of u^* and factor z of w such that $|z| = |u|$, we must have $|z|_a = |u|_a$ and $|z|_b = |u|_b$. Indeed, when running w through the obvious deterministic finite automaton for u^*, we know that, while z is read, the total number of a-transitions and b-transitions will be $|u|_a$ and $|u|_b$.

We now write $u = xy$ such that the last letter of x is different from the first letter of y; by assumption on u, this is always possible. We can now write $u^* = \epsilon + x(u')^*y$, where $u' := yx$; this ensures that the first and last letters of u' are different.

We now show that $(ab)^*$ shuffle-reduces to u^*, by constructing a filter sequence (f_n). To this end, we let u'_{-a} be a word obtained by removing some a in u', and u'_{-b} be defined likewise. Now, to define the filter sequence, we first deal with odd numbers as in the proof of Proposition 3.8, by defining f_{2n+1} for $n \in \mathbb{N}$ as something that can never be in u^* even when inserting $2n+1$ arbitrary symbols, e.g., $f_{2n+1} := a^{(2n+1)\times|u|+1}$, which is clearly computable in PTIME.

For even numbers, we define $f_{2n} := x(u'u'_{-a}u'u'_{-b}u')^n y$ for $n \in \mathbb{N}$: this is clearly computable in PTIME. We show that this is a filter sequence by picking $n \in \mathbb{N}$ and letting v be a word such that $|v| = 2n$. If $v = (ab)^n$, we can clearly interleave v and f_{2n} to obtain a word of u^* by inserting each a of v in u'_{-a} and each b of v in u'_{-b}. Conversely, for an interleaving of any word with f_{2n} to yield a word of u^*, we know that we must at least insert one a in or around each u'_{-a}, and one b in or around each u'_{-b}. Specifically, consider a word v and consider a candidate interleaving $w \in (v \sqcup\!\sqcup f_{2n}) \cap u^*$ and assume by contradiction that there is a factor u'_{-a} of f_{2n} such that w did not insert any a from v within this factor or adjacently to this factor (the case of u'_{-b}-factors is symmetric). Now, consider the factor w' of w that contains this factor u'_{-a}, the neighboring letter from the beginning or end of u' where we take one such letter which is not a (which is always possible by hypothesis on u'), and all inserted elements (which by hypothesis are all b's). The number of a's in the factor w' is $|u'_{-a}|_a$, which is $|u|_a - 1$, but $|w'| \geq |u|$, so, by our preliminary observation, this is impossible because we had assumed that $w \in u^*$. Hence, indeed, we must insert one a in u'_{-a} or adjacently to it, and likewise for the u'_{-b}: these insertions are distinct, and they use up all letters of v, so for $f_{2n} \sqcup\!\sqcup v$ to intersect u^* nontrivially, the only possibility is that $v = (ab)^n$. This shows that (f_n) is indeed a filter sequence, and allows us to conclude by Theorem 3.5 and Theorem 3.6. ◀

▶ **Proposition 3.8.** *Let $L := (aa + bb)^*$. The problem $\mathrm{CSh}[L]$ (hence $\mathrm{CTS}[L]$) is NP-hard.*

Proof. We show a shuffle reduction from $K := (ab)^*$ to $K' := (aa + bb)^*$, which concludes by Theorem 3.5 and Theorem 3.6. We first define the filter sequence for even values of n, and show correctness for them; then we explain how to handle the case of odd n.

For all even $n \in \mathbb{N}$ we set $f_n := (ab)^n$, which is clearly computable in PTIME. Let us now show correctness. For the forward direction, it is clear that for every even $n \in \mathbb{N}$, the only word of length n of $(ab)^*$ is $(ab)^{n/2}$ and we can interleave it with f_n to form $(aabb)^{n/2}$.

For the backwards direction, fix $n \in \mathbb{N}$, take $v \in A^*$ such that $|v| = n$, and assume that $(v \sqcup\!\sqcup f_n) \cap K'$ contains some word w. For each $1 \leq i \leq n$, consider the position where the i-th letter of f_n occurs in w, and call φ_i the maximal factor of w which contains the i-th letter of f_n and consists only of occurrences of the same letter (i.e., is of the form a^* or b^*). By definition, these factors must occur in w in the order $\varphi_1 \leq \cdots \leq \varphi_n$. Now, as any two consecutive letters of f_n are different, we know that the φ_i are disjoint (so we have $\varphi_1 < \cdots < \varphi_n$) and that there is only one letter in each φ_i that was taken from f_n, namely, the i-th letter of f_n: the others were inserted from v. Further, by definition of K', the φ_i must all be of even length. This means that f_1 contains at least one inserted a, that f_2 contains at least one inserted b, etc. As we have $|v| = n$, this completely specifies v, specifically as n is even we must have $v = (ab)^{n/2}$. This is a word of $(ab)^*$, which concludes the backward direction and establishes correctness for even n.

There remains to define the filter sequence for odd numbers, i.e., $2n + 1$ with $n \in \mathbb{N}$. As there are no words of odd length in $(ab)^*$, it suffices to define f_{2n+1} to be something that can never be in K' even when inserting n arbitrary symbols. For instance, we can take $f_{2n+1} := (ab)^{2n+2}$, which has the required property by a variant of the proof for the backward direction above. This concludes the proof of the proposition. ◀

C Proofs for Section 4 (Tractability Results)

C.1 Additional Explanations About $(ab)^* + A^*aaA^*$

Let $A := \{a, b\}$. We first substantiate a claim made in the main text, namely:

▶ **Claim C.1.** *The regular language $K = (ab)^* + A^*aaA^*$ cannot be expressed as a union of monomials.*

We have already mentioned that it is decidable to check if a given (regular) language can be expressed as a union of monomials. We explain how this process can be applied to K to prove the claim:

Proof. It is shown in Theorem 8.7 of [PW97] that a regular language K can be expressed as a union of monomials (equivalently called "languages of level 3/2" in the statement of that result) if and only if the ordered syntactic monoid of K satisfies the *profinite* identity:

$$\text{For all } x, y \in A^* \text{ having same content, } x^\omega \geq x^\omega y x^\omega \tag{1}$$

where "x and y having the same content" means that, for each letter $a \in A$, we have $|x|_a > 0$ iff $|y|_a > 0$, and where ω denotes the idempotent power in the free profinite monoid (see [PW97] for precise definitions).

This can be rephrased in more elementary terms using the notion of *syntactic order* $\leq_K$ induced by K, which can be thought of as an ordered version of the Myhill-Nerode congruence. Formally, the order $\leq_K$ is defined as follows: for all $x, y \in A^*$, we have $x \leq_K y$ iff for all $u, v \in A^*$, $uyv \in K$ implies $uxv \in K$. Equation 1 can then equivalently be rephrased to the following condition: for all words $x, y \in A^*$ with same content, and for all integers n such that $x^n \leq_K x^{2n} \leq_K x^n$, we have $x^n \geq_K x^n y x^n$.

For our choice of language K, we can show that this rephrased condition does not hold, by taking $x := ab$ and $y := bab$ and $n := 1$. Indeed, we have $(ab)^1 \leq_K (ab)^2 \leq_K (ab)^1$, but the right-hand-side of the implication is wrong: we have $x^1 = ab$ in K, so we can take $u = v = \epsilon$ in the definition of the syntactic order, however we then have $x^1 y x^1 = abbabab$ which is not in K, so we have shown that $x^n \not\geq_K x^n y x^n$. ◀

C.2 Proof of Theorem 4.3: Tractability for Monomials

▶ **Theorem 4.3.** *For any monomial language K, the problem* CTS$[K]$ *is in NL.*

Proof. Let K be $A_1^* a_1 A_2^* a_2 \cdots A_n^* a_n A_{n+1}^*$. First, we can guess in NL the vertices $v_1, \ldots, v_n$ of $G = (V, E, \lambda)$ to which the $a_1, \ldots, a_n$ are associated, and verify that indeed we have $\lambda(v_i) = a_i$ for all $1 \leq i \leq n$. Hence, up to making such a guess and relabeling the vertices, we can assume without loss of generality what we call the *fresh pivot* condition on the input A-DAG: for each a_i in our target language, there is exactly one v_i in the input instance such that $\lambda(v_i) = a_i$.

We now prove by induction on n that, for any monomial $K = A_1^* a_1 \cdots A_n^* a_n A_{n+1}^*$, given an input A-DAG satisfying the fresh pivot condition, we can decide in NL whether A has a topological sort satisfying K.

The base case of $n = 0$ is trivial because K is of the form A_1^*: we simply check if all element labels are in A_1^*. For the induction step on $n+1$, let $K = A_1^* a_1 A_2^* \cdots a_{n+1} A_{n+2}^*$ and $K' = A_1^* a_1 A_2^* \cdots a_n A_{n+1}^*$. Let $G = (V, E, \lambda)$ be the input A-DAG satisfying the fresh pivot condition, and let $v_1, \ldots, v_{n+1}$ be the uniquely defined vertices matched to $a_1, \ldots, a_{n+1}$. We define the sub-A-DAG G' to be the restriction of G on the following vertex set V':

- the ancestors of the $v_1, \ldots, v_n$, including $v_1, \ldots, v_n$;
- the ancestors of v_{n+1} except v_{n+1} itself;
- for each w incomparable to v_{n+1} such that $\lambda(w) \notin A_{n+2}$, the ancestors of w (including itself).

We now claim the following:

Claim. G is a positive instance to K iff all descendants z of v_{n+1} are such that $\lambda(z) \in A_{n+2}$ and G' is a positive instance to K'.

Note that G' is always computable in NL, and the condition on the descendants of v_{n+1} can be checked in co-NL, hence in NL by the Immerman-Szelepcsényi theorem [Imm88, Sze88]. Hence, once this claim is proved, we have an NL algorithm for CTS$[K]$ by running the NL algorithm on the descendants of v_{n+1} and running the algorithm given by the induction hypothesis on G', which has been implicitly computed in NL.

What remains is to prove the claim. For the backward direction, if the condition of the claim is respected, then we build the topological sort σ of G satisfying K by concatenating the topological sort σ' of G' satisfying K' which exists by assumption, the vertex v_{n+1} which achieves a_{n+1}, and any topological sort of $G \setminus (G' \cup \{v_{n+1}\})$. We must argue that this a topological sort. Indeed, observe first that the condition of the claim and the fresh pivot condition ensures that no descendant of v_{n+1} has a label in $a_1, \ldots, a_n$, i.e,. v_{n+1} is not an ancestor of any v_i; in particular v_{n+1} is not in V'. However, by definition of G', all ancestors of v_{n+1} are in V'. So we know that we can indeed concatenate σ', v_{n+1}, and a topological sort of the remaining elements of G, and the result σ is indeed a topological sort of G. We now argue that σ achieves K: this is because σ' achieves K', v_{n+1} achieves a_{n+1}, and by assumption all remaining vertices are either descendants of v_{n+1} so their label is in A_{n+2}, or they are incomparable to v_{n+1} so their label must be in A_{n+2} (they would be in G' otherwise). Thus, σ is a topological sort of G that achieves K, establishing the backward implication.

For the forward direction, consider a topological sort σ of G that achieves K. Thanks to the fresh pivot condition, we know that v_{n+1} is matched to a_{n+1}. Let U be the elements enumerated before v_{n+1} in σ, and let σ' be the topological sort induced by σ on U: we know that σ' satisfies K'. We now claim that $V' \subseteq U$. Indeed, first, by the fresh pivot condition, σ' must enumerate a_i for all $1 \leq i \leq n$, so $v_1, \ldots, v_n$ and their ancestors must be in V'. Second, as σ enumerates v_{n+1} just after σ', we know that σ' must enumerate all ancestors of v_{n+1} except v_{n+1} itself. Third, assuming by way of contradiction that V' does not contain an ancestor of a vertex w incomparable to v_{n+1} such that $\lambda(w) \notin A_{n+2}$, we would have that V' does not contain w either, and as w is incomparable to v_{n+1} it is different from v_{n+1} so w must be enumerated after v_{n+1} by σ, but $\lambda(w) \notin A_{n+2}$, which is impossible because we are matching elements to A^*_{n+2} after v_{n+1}. So indeed $V' \subseteq U$. Further, as V' contains the $v_1, \ldots, v_n$, we know that the topological sort σ'' of V' defined as the restriction of σ' to V' also achieves K': intuitively, given a topological sort that achieves K', we can remove any elements except those matched to the a_i and the result still achieves K'. So σ'' witnesses that G' is a positive instance to K'. Now, as σ must enumerate all descendants z of v_{n+1} after v_{n+1} which achieves a_{n+1}, we know that they must be such that $\lambda(z) \in A_{n+2}$, so we have shown the condition and established the forward implication.

We have shown our claim, which concludes the proof of Theorem 4.3. ◀

C.3 Proof of Proposition 4.5: Tractability Based on Width

▶ **Proposition 4.5.** *Let $A := \{a, b\}$, let K' be a regular language, let $i \in \mathbb{N}$, and let $K := K' + A^*(a^i + b^i)A^*$. The problem* CTS$[K]$ *(hence* CSh$[K]$*) is in NL.*

To show this result, we will need several preliminary definitions. Recall from the main text that an *antichain* is a set $S \subseteq V$ of vertices which are pairwise incomparable, and the *width* of a DAG is the size of its largest antichain. The main claim is then the following:

▶ **Proposition C.2.** *For any regular language K, the problem* CTS$[K]$ *can be solved in space $O(k \log n)$, where k is the width of the input DAG and n is its total size. The same bound holds for* CSh$[K]$ *where k is the number of input strings.*

We note that a similar task was already known to be in PTIME by [ABDS17, Theorem 17], but showing the space bound given here will introduce several additional technicalities. From Proposition C.2, it is easy to show Proposition 4.5:

Proof of Proposition 4.5. We follow the proof sketch: we test in NL if the input DAG contains an antichain of size $2i$. If it does, as explained in the sketch, the DAG is a positive instance. So the only remaining case is when the input DAG has width $\leq 2i$, so we can conclude by Proposition C.2. ◀

So all that remains is to show Proposition C.2, which we do in the rest of Appendix C.3. The high-level idea of the proof is to use Dilworth's theorem [Dil50], which essentially shows that the *width* of any DAG G is equal to the minimal cardinality of a *chain partition* of G, i.e., a partition of G into disjoint chains, where we may additionally have arbitrary edges between the chains. We will then perform a logspace algorithm following such a partition to guess an accepting path of a (fixed) automaton for K.

We present the complete proof of Proposition C.2 in the rest of Appendix C.3. We first define formally the notion of chain partition. Let $G = (V, E)$ be a DAG, and let (V, E') be its transitive closure. A *chain partition* of G is a partition $V_1 \sqcup \cdots \sqcup V_n$ of V, such that, for all $1 \leq i \leq n$, the restriction of E' to $V_i \times V_i$ is the transitive closure of a directed path graph: equivalently, for each pair of vertices $v \neq v'$ of V_i, either v has a directed path to v' in E or v' has a directed path to v in E. We call each of the V_i a *chain* of G. Note that, in addition to the edges between vertices of the chain, there may also be arbitrary edges in G between V_i and V_j for $i \neq j$. The *width* of a chain partition is the number of chains that it contains. The following is then known from partial order theory:

▶ **Theorem C.3 [Dil50].** *For any DAG G, the width of G is k iff there exists a chain partition of width k of G.*

However, to show our desired space bound, we need to look closely into the complexity of computing a chain partition. This task is known to be in PTIME [Ful55] but we are unaware of an existing proof to show that it can be done in NL. Because of this, we must give a custom scheme to compute implicitly a specific chain partition that meets our logspace requirements. One difficulty will be to ensure that, as we compute the chain partition implicitly in NL, we are always looking at the same chain partition each time we recompute it implicitly (i.e., we are not looking at some random chain partition that was nondeterministically chosen for this implicit computation). To fix a canonical choice of chain partition, we look at the *minimal* one in an order that we will define.

We will see a width-k chain partition as a labeling function χ from V to $\{1, \ldots, k\}$ such that, letting $V_i := \{v \in V \mid \chi(v) = i\}$, then $V_1 \sqcup \cdots \sqcup V_k$ is indeed a chain partition. Given a DAG (V, E), the vertices of V are integers, each of them represented in binary by a sequence of size $\log |n|$, and we let $<$ denote the corresponding total order relation on V. We can then talk about the topological sort σ, equivalently seen as a total order $<_\sigma$, which is minimal according to the lexicographic order defined by $<$: namely, σ is constructed by picking, at each step, the smallest possible vertex according to $<$ which can be picked (i.e., it has not been picked yet, but all its ancestors have): we write the vertices of V in the order of $<_\sigma$ as $v_1 < \cdots < v_{|V|}$. We then lift the total order $<_\sigma$ on V to a total order relation on chain partitions: we write each chain partition as the word $\chi(v_1) \cdots \chi(v_{|V|})$, and $<_\sigma$ defines an order on the chain partitions given by the lexicographic order on words of $\{1, \ldots, k\}^{|V|}$. Now, we can talk about the chain partition χ_0 which is *minimal* according to this total order relation $<_\sigma$ on chain partitions. We will explain how this minimal chain partition can be computed implicitly in logspace. Again, the reason why we are concerned about minimality is simply to ensure that, when using the implicitly-computed chain partition within our logspace algorithm for CTS$[K]$, then the chain partition that we follow is well-defined, i.e., it is the same over all calls to the implicit nondeterministic logspace chain partition oracle. The specific definition of minimality that we use does not matter much.

We now describe the specific implicit representation that we want for the minimal chain partition χ_0. We want to show that we can evaluate efficiently two functions: one function next, which takes as input a vertex $v \in V$ and returns the *next vertex* of the chain of v in χ_0, and one function first, which takes as input a chain number $1 \leq i \leq k$ and returns the *first vertex* of i in χ_0. Formally, $\mathsf{next}(v)$ for $v \in V$ is defined as follows: letting $c := \chi_0(v) \in \{1, \ldots, k\}$ be the chain to which v belongs in χ_0, return the vertex $\mathsf{next}(v) \in V$ which is the successor of v on chain c in χ_0, if any, or $\top$ if v is the last vertex of chain c. More formally, $\mathsf{next}(v)$ is the vertex of V such that $\chi_0(\mathsf{next}(v)) = c$, the edge $v \to \mathsf{next}(v)$ is in E', and there is no $z \in V$ such that $\chi_0(z) = c$ and the edges $v \to z$ and $z \to \mathsf{next}(v)$ are in E'. As for the function first, for any chain number $1 \leq c \leq k$, we let $\mathsf{first}(c)$ be the first element of the chain c in χ_0, that is, we have $\chi_0(\mathsf{first}(c)) = c$ and there is no $z \in V$ such that $\chi_0(z) = c$ and the edge $z \to \mathsf{first}(c)$ is in E'. We can now claim:

► **Lemma C.4.** *For any input to the functions* next *and* first*, we can evaluate them in space* $O(k \log n)$.

We will do two things in the sequel: prove this lemma, and use it to prove Proposition C.2. To do this, we need to define the notion of a *configuration*, which will be useful in our algorithms on chain partitions. A *configuration* is a k-tuple $X = (v_1, \ldots, v_k)$, where each v_i is either an element of V or $\perp$. Intuitively, X describes the lowest element of each chain, with $\perp$ indicating that no element has been assigned to this chain so far; when we consider a configuration X in an algorithm, we assume that the *ancestors* of X, meaning all vertices w such that for some v_i the edge $w \to v_i$ is in E', have already been assigned to a chain in some fashion. We say that a configuration X is *continuable* if there exists a chain partition χ which is consistent with X, meaning that $\chi(v_i) = i$ for all $1 \leq i \leq k$ such that $v_i \neq \perp$. One useful lemma will be the following:

► **Lemma C.5.** *There is an algorithm to decide, given a configuration X, whether X is continuable, in space $O(k \log n)$.*

We will first show how to use this lemma to prove Lemma C.4. We will then explain how to prove Lemma C.5. Last, we will prove Proposition C.2 from Lemma C.4.

We start by proving Lemma C.4. The intuition is that we can use the continuation check of Lemma C.5 as a way to compute implicitly the minimal chain partition, by considering all vertices in the minimal topological sort $<_\sigma$, and assigning each vertex to the smallest possible chain such that the resulting configuration is continuable. Formally, we show:

Proof of Lemma C.4. We maintain a configuration $X = (v_1, \ldots, v_k)$, initially $(\perp, \ldots, \perp)$, and extend it *deterministically* at each step using the (nondeterministic) oracle for continuation checking described in Lemma C.5. Specifically, at each step of the algorithm, we call S the set of vertices which are ancestors of elements in X, and we consider the vertex v which is as small as possible according to $<_\sigma$ and which is not in S but all its strict ancestors are in S: we can find this vertex in NL. Now, for each $1 \leq i \leq k$ such that the edge $v_i \to v$ is in E' or $v_i = \perp$, we check whether the configuration X_i obtained by replacing v_i by v is continuable. We pick the smallest i such that it is, and continue the algorithm with X_i: specifically, we guess a suitable i, and guess in co-NL that there is no $i' < i$ which is suitable: this is still in NL overall, thanks to the Immerman-Szelepcsényi theorem [Imm88, Sze88]. At the end of the process, we have memorized the successor of the vertex of interest on its chain (i.e., the input to next), or the first vertex of the chain of interest (i.e., the input to first), and we return this.

We will soon explain why the algorithm does not get stuck, in the sense that, for each vertex v that we consider, there is a choice of i for which the conditions are respected. However, notice first that, if the algorithm does not get stuck, then the algorithm considers all vertices of V exactly once, following the order $<_\sigma$ of the minimal topological sort. Indeed, at each step, the set S contains all vertices that have been seen so far: the only thing to notice is that, whenever we remove a vertex z from the configuration, we replace it by a vertex z' such that all of its ancestors are in S and z is an ancestor of z', so that the new value of S becomes $S \cup \{z\}$. This ensures that we are indeed picking at each step the next vertex that $<_\sigma$ has picked.

We now explain why the algorithm does not get stuck, which we show by induction. Initially, the configuration is $(\perp, \ldots, \perp)$, and this configuration is continuable, as we know by Dilworth's theorem (Theorem C.3). Now, at each step of the algorithm, the current configuration X is continuable by induction hypothesis, because it was chosen to be continuable at the previous step of the algorithm. Now, as $X = (v_1, \dots v_k)$ is continuable, letting χ be a witnessing chain partition, letting v be the next vertex that we consider, we know that, if $v_{\chi(v)} = \perp$, then we can take $i = \chi(v)$. If $v_{\chi(v)} \neq \perp$, then $v_{\chi(v)}$ must be an ancestor of v in chain i, and by the condition on the ancestors of v, we know that v must be the first descendant of $v_{\chi(v)}$ on the chain, justifying that the edge $v_{\chi(v)} \to v$ must exist in E'. Hence, χ witnesses that the algorithm does not get stuck.

Last, we argue that the values computed by the algorithm are correct. To do so, we show by induction that all choices performed by the algorithm actually follow χ_0, in the sense that, at each step of the algorithm, the current configuration is consistent with χ_0, and, for each vertex v that we consider, we take $i := \chi_0(v)$. We do this by mutual induction on these two claims. The base case is trivial because $(\perp, \dots, \perp)$ is of course consistent by χ_0. Now, assuming consistency of the configuration, as χ_0 is defined to be minimal following $<_\sigma$, by minimality of the vertex v picked by both $<_\sigma$ and the algorithm, we know that $\chi_0(v)$ is the minimal value such that the resulting configuration is continuable. Indeed, if it were not, then by taking a smaller continuable value, and taking any witnessing continuation afterwards, we would obtain a chain partition which would be smaller in the lexicographic order, contradicting the minimality of χ_0. So we have shown that our algorithm actually computes next and first following χ_0, proving the result. ◀

We now come back to the proof of Lemma C.5:

Proof of Lemma C.5. The proof follows similar ideas as in Lemma C.4: we have a current configuration, we consider the vertices following a topological order, and we try to assign them to a chain, updating the configuration. The only difference is that, instead of assigning the minimal chain number following a continuation check, we simply nondeterministically guess a chain to which we assign them. When the nondeterministic guesses succeed, we can show exactly as in Lemma C.4 (but without worrying about minimality) that these guesses witness the existence of a chain partition which is consistent with the input configuration X, so that X is indeed continuable; and conversely, whenever such a chain partition exist, these is a sequence of nondeterministic guesses which make the algorithm succeed. ◀

Thanks to Lemma C.4, we now know that we can implicitly compute the minimal chain partition within the prescribed time bounds. We are now ready to prove Proposition C.2:

Proof of Proposition C.2. We fix an automaton $\mathcal{A}$ for the regular language K: remember that, as K is fixed, we can compute $\mathcal{A}$ in constant time, and the size of its state set Q and transition relation $\delta \subseteq Q \times A \times Q$ is constant.

Our state at any stage of the algorithm will consist of a *configuration*. Remember that this is a k-tuple $X = (v_1, \dots, v_k)$ such that each v_i is either $\perp$ or an element of V, which intuitively codes the lowest element for each chain, or $\perp$ if no element of the chain has been seen so far: initially the configuration is $(\perp, \dots, \perp)$. The state also contains one state $q \in Q$ of the automaton, which is initially some initial state, chosen nondeterministically.

At each stage of the algorithm, we nondeterministically guess one chain $1 \leq i \leq k$ to extend. We then replace the current configuration X with the new configuration X_i defined as follows: if $v_i = \perp$, then we replace v_i in X_i by $v'_i := \mathsf{first}(v_i)$; if $v_i \neq \perp$, then we replace v_i in X_i by $v'_i := \mathsf{next}(v_i)$ if it is different from $\top$; otherwise we cannot choose this value of i. We also cannot choose a value of i when the v'_i that we have defined cannot be enumerated yet, i.e., if it is not the case that all strict ancestors of v'_i are in X or are ancestors of vertices in X. Once we have made an appropriate choice for i, we also replace the current state q with some element q' such that $(q, \lambda(v'_i), q') \in \delta$, nondeterministically chosen. Intuitively, this means that the automaton processes the letter which is the label of the new element v_i' which is read along the chain i.

The algorithm concludes when we can no longer perform a step, meaning that $v_i \neq \bot$ and $\mathsf{next}(v_i) = \top$ for each $1 \leq i \leq k$. Then, the algorithm accepts if the current state q is final.

It is clear that, whenever the algorithm succeeds, then the sequence of guesses witnesses the existence of a topological sort of G, obtained following the vertices that are chosen at each step: the definition of the steps that we perform ensure that this sequence indeed respects the edge relation of G, for similar reasons as in the proof of Lemma C.4. Conversely, whenever there is a witnessing topological sort, then we can decompose it along the minimal chain partition χ_0 defined earlier. Specifically, the sequence of vertices given by this topological sort can be expressed as a sequence of operations where we enumerate the first vertex of a chain, or enumerate the next vertex of a chain from the preceding one. The definition of the algorithm ensures that these steps can be mimicked by a sequence of nondeterministic guesses (in particular, following these guesses, the algorithm does not "get stuck" and can always pick the right v_i' at each step), and likewise the accepting path in the automaton can be mimicked by nondeterministic choices of the states in the transition relation. This establishes the correctness of the algorithm, and concludes the proof. ◀

C.4 Proof of Proposition 4.6: Other Tractable Case

▶ **Proposition 4.6.** *Let $A := \{a, b\}$ and $K := (aa + b)^*$. The problem* $\mathrm{CSh}[K]$ *is in NL.*

Proof. We can first check in NL whether the total number of a-elements is even; if not, clearly there is no suitable topological sort, so in the sequel we assume that it is.

Note that, if any string consists only of b's, then we can clearly enumerate these b's first, and the result is equisatisfiable; so without loss of generality we can always remove any input string that consists only of b's as soon as they appear, so we never consider such strings. Now, if there are less than 3 input strings, then we can conclude in NL by Proposition C.2, so we assume that there are at least 3 strings in the input instance (which contain some a by the assumption that we just made).

Given an input instance I to the CSh-problem for K, we call a *block* a maximal contiguous sub-sequence of a-labeled elements in a string, and call it an *even* or *odd* block depending on the number of such elements. The *a-weight* of a string is its total number of a-labeled elements, and the *a-alternation* of a string is its total number of odd a-blocks.

We claim that I does *not* have a topological sort satisfying K if and only if there is a string whose a-alternation is greater than the sum of the a-weights of all other strings. This condition can clearly be checked in NL: compute the maximal a-alternation of a string, and compute the a-weight of the other strings and compare. Hence, all that remains is to show this condition.

The easy direction is the backward one. If there is a string C whose a-alternation is greater than the sum of the a-weights of all other strings, we know that any topological sort satisfying K must enumerate one element of every odd block of C together with an a-element of another string of C: indeed, when enumerating two a-labeled elements from C, they must be in the same block because of the b-elements between blocks, so this cannot change the parity of a block of C. Hence, under our assumption, a topological sort would have to enumerate more a-elements in the other strings than their total a-weight, which is impossible; this concludes the backward direction.

To show the forward direction, we show the contrapositive: if, for each string C, the a-alternation of C is no greater than the total a-weight of the other strings (which we call assumption (*)), then there exist a suitable topological sort.

We first make a simplifying observation. Given an instance I, for any choice of two contiguous a-elements in a string of I, we let I' be the result of removing these two elements. If I' has a suitable topological sort, then so does I, because we can just mimic the topological sort on I and enumerate the two adjacent a-elements when they become available. Hence, to show that there is a suitable topological sort, we can always decide to remove any two contiguous a's in a block (even at a stage where they are not available). We call this a *simplification*. Note, however, that we cannot apply this simplification blindly, as the converse implication to the above does not hold in general (consider $\{ababa, aaa\}$ vs $\{ababa, a\}$).

We will define a second assumption (**), and show two things: (i.) given any input instance satisfying (*) with an even number of a's and with at least 3 strings (containing some a), we can rewrite it through simplifications (and removal of strings containing only b) to an instance satisfying (**), and (ii.) that given an instance satisfying (**) and our preliminary assumptions, we can build a suitable topological sort. Condition (**) says: for each string C, the a-*weight* of C is no greater than the total a-weight of the other strings. (Notice the difference with (*).)

We first show (ii.): under our preliminary assumptions on I, any instance satisfying (**) has a suitable topological sort. We do so by describing a greedy algorithm which enumerates elements in a way that achieves a suitable topological sort. Namely:

1. If we can enumerate a b-element, then enumerate it.
2. Otherwise, pick the two strings whose non-enumerated elements have largest a-weight and enumerate one a from each of these two strings.

If this algorithm does not get stuck, then it clearly constructs a topological sort satisfying K. Now, the only way for this algorithm to get stuck is if there is only one string left, but this would violate (**). Hence, it suffices to show that the algorithm preserves assumption (**). Clearly step 1 preserves it, so we focus on step 2. By assumption (**) there are at least two strings left: if there are exactly two strings left, then condition (**) is preserved as the a-weight of both strings is decreased. Assume now that there are at least three strings left before applying step 2, and let C, C', C'' be the strings with the largest a-weight (in terms of unenumerated elements) and let $n \geq n' \geq n''$ be their respective a-weights. After step 2, the a-weights are $n-1$, $n'-1$, and n''. It is clear that, as condition (**) held of C and C' before step 2, then the condition still holds, as the a-weight of each of these two strings and the total a-weight of the other strings has been decremented, then condition (**) still holds of these strings. We must show that it holds of the other strings, and clearly it suffices to focus on C'', which has the largest a-weight in terms of non-enumerated elements. There are three cases, depending on the relationship of n'' to n.

- If $n'' < n-1$, then as (**) is still satisfied for C after the step and the a-weight of C'' is still smaller than C after the step, then (**) is satisfied for C'' too.
- If $n'' = n-1$, then after performing the step, C and C'' have same a-weight, and it is obvious that if condition (**) holds of a string C then it holds of a string with the exact same a-weight (as the a-weight of the two strings is the same, and so is the a-weight of the other strings).
- If $n'' = n$, then we have $n'' = n' = n$. Now, the only problematic case would be if, after performing the step, n'' were strictly greater than the a-weight of all other strings, in particular, we would have $n'' > (n-1) + (n'-1)$. But substituting in this inequality we get $n > 2n-2$, hence $n < 2$. Hence, the only bad situation is when all strings have a-weight at most 1, but then, remembering that the number of a's was initially even and

clearly remains even throughout the enumeration, we have at least 2 strings left after the step in this case that all have a-weight exactly 1, so condition (**) is always respected.

Hence, we have shown that, on any input instance satisfying condition (**) in addition to our preliminary requirements, the above algorithm succeeds and produces a suitable topological sort.

The only thing left to show is (i.): given an instance satisfying (*) and our preliminary requirements, in particular that of having at least 3 strings containing an a-element, then we can rewrite it using simplifications to an instance satisfying (**). To do so, let us observe that, for any string with a-alternation n and a-weight m, we can clearly perform simplifications to rewrite it to a string of a-weight p for any value $n \leq p \leq m$ of the same parity as m (or of n, as m and n have same parity). So let us simplify the string C with the greatest a-alternation to make its a-weight equal to its a-alternation n, and let us rewrite all strings in the following way: if the string has a-weight $\leq n+1$, we do not change it; otherwise we simplify it to n or $n+1$ depending on the parity of its a-weight. Let us show that the result of this transformation satisfies assumption (**). Consider a string C' and show the condition. If $C' = C$, then C has a-weight n, and thanks to condition (*) we know that the sum of a-weights are greater than n, because the only case where we have reduced the a-weight of another string C'' than C was to bring it down to n or $n+1$, in which case C'' suffices to witness that (**) is satisfied for C. If C' is different from C, then its greatest possible a-weight is $n+1$ by construction, however, we know that C achieves a-weight n, and thanks to the assumption that we have at least 3 strings containing a's, we know that there is another string containing some a, hence (**) holds for C'. This establishes that (**) now holds after the simplifications, which concludes the proof. ◀

D Proofs for Section 5 (A Coarser Dichotomy Theorem)

D.1 Proofs of Proposition 5.1: Closure Counterexamples

▶ **Proposition 5.1.** *We have the following counterexamples to closure:*

- ***Quotient.*** *There exists a word $u \in A^*$ and a regular language K such that* $\mathrm{CSh}[K]$ *is in NL but* $\mathrm{CSh}\left[u^{-1}K\right]$ *is NP-hard.*
- ***Intersection.*** *There exists two regular languages K_1 and K_2 such that* $\mathrm{CTS}[K_1]$ *and* $\mathrm{CTS}[K_2]$ *are both in PTIME but* $\mathrm{CSh}[K_1 \cap K_2]$ *is NP-hard*
- ***Complement.*** *There exists a regular language K such that* $\mathrm{CTS}[K]$ *is in NL, but* $\mathrm{CSh}[A^* \setminus K]$ *is NP-hard.*
- ***Inverse of morphism.*** *There exists a regular language K and morphism φ such that* $\mathrm{CTS}[K]$ *is in NL but* $\mathrm{CSh}\left[\varphi^{-1}(K)\right]$ *is NP-hard.*

First, we show that tractable languages for CSh are not closed under left quotient. Recall that the *left quotient* of a language K by a word $u \in A^*$ is the language $u^{-1}K := \{v \in A^* \mid uv \in K\}$; right quotients are defined analogously. We only consider left quotients, but of course the same result holds for right quotients because both our problems are symmetric under the reverse operator:

▶ **Proposition D.1.** *There exists a word $u \in A^*$ and a regular language K such that* $\mathrm{CSh}[K]$ *is tractable but $u^{-1}K = (ab)^*$, so that* $\mathrm{CSh}\left[u^{-1}K\right]$ *is NP-hard by Theorem 3.6.*

Proof. Take $A := \{a, b\}$ and $K := bA^* + aaA^* + (ab)^*$. Take $u := ab$. It is clear that $u^{-1}K = (ab)^*$. However, $\mathrm{CSh}[K]$ is tractable by the following reasoning. Consider an input instance to $\mathrm{CSh}[K]$. If there is a string that starts with b, then we can clearly always construct a topological sort achieving bA^*. Hence, we can assume that all strings start with a. If there are two strings or more, by taking their first letters, we can clearly always construct a topological sort achieving aaA^*. Hence, we can assume that there is only one string, and we can clearly check in NL whether the only possible topological sort achieves K. ◀

However, we point out that the tractable languages for the CTS-problem *are* closed under quotient:

▶ **Proposition D.2.** *For any word $u \in A^*$ and regular language K, there is an logspace reduction from* $\mathrm{CTS}\left[u^{-1}K\right]$ *to* $\mathrm{CTS}[K]$.

Thus, for the language $K := b^*A^* + aaA^* + (ab)^*$ used in the proof of Proposition 5.1, we know that $\mathrm{CTS}[K]$ is NP-hard but $\mathrm{CSh}[K]$ is in NL: hence, K separates the problems CSh and CTS.

Proof of Proposition D.2. Fix $u \in A^*$ and K. Given an A-DAG G, to solve $\mathrm{CTS}\left[u^{-1}K\right]$ on G, construct the DAG G' obtained by adding a directed path of elements whose label is u and adding edges from each element of the directed path to all elements of G. It is obvious that there is a topological sort of G' achieving K iff there is a topological sort of G achieving $u^{-1}K$, which concludes the proof. ◀

Second, we illustrate that tractable languages are not closed under the intersection operator, for both problems:

▶ **Proposition D.3.** *There exists two regular languages K_1 and K_2 such that* $\mathrm{CTS}[K_1]$ *and* $\mathrm{CTS}[K_2]$ *are both in PTIME, but $K_1 \cap K_2 = (ab)^*$, so that* $\mathrm{CSh}[K_1 \cap K_2]$ *is NP-hard by Theorem 3.6.*

Note that we do not show that $\mathrm{CTS}[K_1]$ and $\mathrm{CTS}[K_2]$ are in NL, although we conjecture that this should hold.

Proof of Proposition D.3. We fix $A := \{a, b\}$ and take $K_1 = (ab)^*(\epsilon + bA^*)$ and $K_2 = (ab)^*(\epsilon + aaA^*)$. It is clear that $K_1 \cap K_2 = (ab)^*$, so we only need to show that $\mathrm{CTS}[K_1]$ and $\mathrm{CTS}[K_2]$ are tractable. Now, observe that $a^{-1}K_1b^{-1} = (ba)^*(\epsilon + bbA^*)$, which is the result of swapping the symbols a and b in K_2. Hence, if we establish that $\mathrm{CTS}[K_1]$ is in PTIME, then by Proposition D.2, as PTIME-membership is clearly preserved by renaming the symbols, we have also shown that $\mathrm{CTS}[K_2]$ is in PTIME. So we focus on K_1.

We will show a greedy algorithm in PTIME to solve $\mathrm{CTS}[K_1]$, and explain why it succeeds. The algorithm has two states:

- State α (the initial state), where:
  - being out of symbols means that we have succeeded, i.e., we have constructed a topological sort in $(ab)^*$;
  - enumerating an a allows us to move to state β;
  - enumerating a b allows us to "win", i.e., that we can continue the topological sort in any way and remain in K_1.
- State β, where:
  - being out of symbols means that we have failed, i.e., the word that we have formed is of the form $(ab)^*a$ and not in K_1;
  - enumerating an a is forbidden;
  - enumerating a b allows us to move back to state α.

We accordingly design the algorithm as follows:

- In state β:
  - if there is an available b, enumerate any of them and move to state α;
  - otherwise fail.
- In state α:
  - if there is an available b, enumerate it and succeed;
  - otherwise, if there is an available a such that, when enumerating this a, there is an available b (call this a *profitable* a), then enumerate any one of these a's and move to state β;
  - otherwise, if there are no symbols left, succeed;
  - otherwise fail.

If the algorithm succeeds, then it clearly builds a suitable topological sort, hence we have to argue for the other direction: if there is a suitable topological sort then the algorithm will find it. To do so, we must justify that the choices made by the algorithm are without loss of generality, i.e., any suitable topological sort can be modified to follow the rules of the algorithm, so as to witness that the algorithm succeeds.

Let us thus consider a witnessing topological sort σ, and consider the first point at which σ disagrees with the actions of the algorithm, and change σ to continue like the algorithm did and still achieve K: we can then repeat the argument until σ is exactly what the algorithm does, which allows us to conclude that the algorithm succeeds. When the algorithm did the choice that disagrees with σ, either it was in state α or in state β; note that if the algorithm had already decided that it had succeeded, then there is nothing left to show as indeed the topological sort is suitable no matter how it continues.

If the algorithm was in state β, as σ is suitable, there must be an available b. If there is only one available b, then the algorithm and the topological sort cannot disagree, so the only thing to consider is the case where the algorithm picks one b-labeled element v and σ picks another v'. However, in this case, we can modify σ to pick v' and then pick v (which is available), and this ensures that σ succeeds immediately, so it is still suitable. So we have modified modify σ to do like the algorithm does (and succeed immediately).

If the algorithm was in state α, if there is an available b, then we can always modify σ to take it and succeed. Likewise, if there is no available symbol, then σ and the algorithm are both finished and both succeed. Hence, the only possible disagreement is if σ picks a different a than what the algorithm chose, of if σ picked an unprofitable a whereas the algorithm failed. However, note that, as σ is a suitable topological sort, it cannot pick an unprofitable a, as it will necessarily be stuck afterwards (only a's, if anything, will be available, and we will be in state β), so the second case is impossible by our assumption that σ is suitable. So the only case to consider is the first case, and we will explain how to modify σ to pick immediately the profitable a that the algorithm enumerates (call it v), followed by the b that the algorithm enumerates after v (call it w).

To do this, consider the later moment at which σ enumerates v. It is possible that, when σ enumerates v, it has already succeeded (meaning, there were two contiguous b's after an a earlier in σ), but in this case there is no constraint on σ and we can simply move v and w in σ to enumerate them at the moment the algorithm does, and σ is still suitable. If σ has not already succeeded when it enumerates v, then either σ enumerates w just after v (subcase 1), or it does not (subcase 2). If it does (subcase 1), then we can modify σ by moving v and w to the beginning: σ is still a topological sort after this change (indeed we can enumerate v and w because the algorithm does it, and for the other elements there is nothing to show), and σ is still suitable (we have added an ab-factor at the beginning, and removed an ab-factor in what follows but this has no effect on the fact that σ realizes K_1).

Now, if σ does not enumerate w immediately after v (subcase 2), then let w' be the element that σ enumerates instead; it must be b-labeled (by our assumption that σ has not already succeeded when it enumerates v). But we know from what the algorithm does that σ can only enumerate w after having enumerated v, and not before it has enumerated v, so σ must enumerate w somewhere after w'. We modify σ to enumerate w instead of w' and enumerate w immediately after: this is still a topological sort as we just explained, and σ is still suitable (no matter what happens afterwards)because now it enumerates two consecutive b's (namely, w and w') immediately after an a (namely, v). We are now back to subcase 1, because σ now enumerates w just after v, so we can conclude as in that subcase. This concludes the correctness proof.

Note that the algorithm described here is not in NL; we conjecture that $\mathrm{CTS}[K_1]$ is in NL, but we do not know how this can be shown. ◀

Third, we show that tractable languages are not closed under complement:

▶ **Proposition D.4.** *There exists a regular language K such that $\mathrm{CTS}[K]$ is in NL, but $A^* \setminus K = (ab)^*$, so that $\mathrm{CSh}[A^* \setminus K]$ is NP-hard by Theorem 3.6.*

Proof. Take $K = bA^* \cup A^*a \cup A^*aaA^* \cup A^*bbA^*$. As K is a union of monomials, we know by Theorem 4.3 that $\mathrm{CTS}[K]$ is in NL, however by construction we have $A^* \setminus K = (ab)^*$. ◀

Fourth, we show that tractable languages are not closed under *inverse morphisms*. Recall that a *morphism* from alphabet B to alphabet A is a function $\varphi : B^* \to A^*$ such that $\varphi(uv) = \varphi(u)\varphi(v)$ for all $u, v \in B^*$; note that a morphism is completely defined by the image of each letter of B. The *inverse image* of a language K over alphabet A by a morphism φ is the language over alphabet B defined by $\varphi^{-1}(K) := \{v \in B^* \mid \varphi(v) \in K\}$. We show:

▶ **Proposition D.5.** *There exists a regular language K and morphism φ such that $\mathrm{CTS}[K]$ is in NL, but $\varphi^{-1}(K) = (ab)^*$, so that $\mathrm{CSh}\left[\varphi^{-1}(K)\right]$ is NP-hard by Theorem 3.6.*

Proof. We take $A := \{a, b\}$ and $K := (ab)^* + A^*(a^3 + b^3)A^*$, as in Proposition 4.5. We know by this proposition that $\mathrm{CTS}[K]$ is in NL. However, let $\varphi : A^* \to A^*$ be defined by $\varphi(a) := aba$ and $\varphi(b) := bab$. We then have $\varphi^{-1}(K) = (ab)^*$, for which the CSh-problem is NP-hard by Theorem 3.6. Indeed, no word in the image of φ has three consecutive a's or three consecutive b's, so $\varphi^{-1}(K) = \varphi^{-1}((ab)^*)$, and this is clearly equal to $(ab)^*$. ◀

D.2 Proof of Theorem 5.2: Coarser Dichotomy Theorem

Recall from the main text the definition of the *transition monoid* $T(S)$ of a semiautomaton S. We call *transition morphism* the morphism $\eta : A^* \to T(S)$ defined by $\eta(u) = f_u$ for all $u \in A^*$: by construction, this morphism is surjective. Recall that our counter-free assumption on S is equivalent to requiring that $T(S)$ is an *aperiodic monoid*: formally, it for all $x \in T(S)$, we have $x^\omega = x^{\omega+1}$, where $\omega \in \mathbb{N}$ is the *idempotent power* of M, i.e., the least integer $\omega \in \mathbb{N}$ such that for every element x in M, we have $x^\omega = x^{2\omega}$.

Our characterization of tractable semiautomata in Theorem 5.2 is based on the class **DA** of monoids [TT02], which is a subset of **A**. A monoid M is in **DA** iff it satisfies the equation $(xy)^\omega x(xy)^\omega = (xy)^\omega$ for all $x, y \in M$, where ω again refers to the idempotent power of M; this implies in particular that M is aperiodic. Our dichotomy result relies on the following characterization of **DA**:

▶ **Theorem D.6 ([TT02], Theorem 5 and Theorem 11).** *Let K be a regular language of A^*. The following conditions are equivalent:*

- *K is an union of* unambiguous *monomials, i.e., of monomials $K = A_1^* a_1 \cdots A_n^* a_n A_{n+1}^*$ such that every word $u \in K$ has a unique decomposition $u_1 a_1 \cdots u_n a_n u_{n+1}$ where $u_i \in A_i^*$ for all $1 \leq i \leq n+1$.*
- *There exists a monoid M in **DA** and a morphism $\varphi : A^* \to M$ such that K is* recognized *by M, meaning that $K = \varphi^{-1}(P)$ for some subset $P \subseteq M$.*

We will also rely on a characterization of monoids that are *not* in **DA**:

▶ **Proposition D.7 ([TT01], Lemma 10).** *An aperiodic monoid M is not in **DA** iff there exists a morphism $\theta : \{a, b\}^* \to M$ and $P \subseteq M$ such that $\theta^{-1}(P)$ is either $(ab)^*$ or $(ab + b)^*$.*

Proof. This result follows from [TT01], Lemma 10, but the latter result is presented in slightly different terminology. Specifically, that result states that an aperiodic monoid is not in **DA** iff it is *divided* by two monoids BA_2 and U, that are respectively the syntactic monoid of $(ab)^*$ and $(ab + b)^*$ (up to relabeling the symbols of Figure 2 of [TT01]). A monoid N *divides* another monoid M iff there exists a submonoid K of M such that N is a quotient of K. Our lemma follows from this result thanks to the well-known fact that a language K is recognized by a monoid M iff its syntactic monoid divides M: see [Str94, Theorem V.1.3]. ◀

We are now ready to state and prove our dichotomy theorem:

▶ **Theorem 5.2.** *Let S be a counter-free semiautomaton. Then the multi-letter CSh-problem and CTS-problem for S are either both in NL, or both NP-complete. The dichotomy is effective: given S, it is PSPACE-complete to decide which case applies.*

Proof. Fix the input semiautomaton S. We wish to show that the multi-letter CTS-problem is tractable for S iff the transition monoid $T(S)$ of S is in **DA**. We call $\mathrm{SL}(K)$ the set of possible languages that can be defined from S depending on the input instance, namely, depending on the set $\{(i_1, F_1), \ldots, (i_k, F_k)\}$ of pairs of initial and final states. For one direction we prove that: (a) if $T(S)$ is in **DA**, then for any language K in $\mathrm{SL}(S)$, the multi-letter CTS-problem for K is in NL. For the converse direction we prove that: (b) if $T(S)$ is not in **DA**, then there exists a language K in $\mathrm{SL}(S)$ whose multi-letter CSh-problem is NP-complete, so we can show NP-hardness by restricting to input instances that use this language.

Proof of (a). Assume that $M := T(S)$ is in **DA**. We denote by $\eta : A^* \to M$ the transition morphism of S and by $\psi : M^* \to M$ the morphism on words over the alphabet M defined by $\psi(m) := m$ for all $m \in M$. Intuitively, applying ψ to a sequence of elements of M simply evaluates the sequence in M.

Let $I = (G, (i_1, F_1), \ldots, (i_k, F_k))$ be an instance of the semiautomaton CTS-problem and let K_j be the language recognized by the automaton (Q, A, δ, i_j, F_j) for all $1 \leq j \leq k$. We must determine whether $G = (V, E, \lambda)$ has a topological sort in $K := \bigcap_j K_j$. We will reduce this to our original definition of the CTS-problem for regular languages, with a language that we know to be in NL. Specifically, we will work on the alphabet M of the transition monoid, and the language that we will use is $K' := \{u \in M^* \mid \psi(u) = \eta(K)\}$. In other words, $K' = \psi^{-1}(\eta(K))$, so K' is recognized by M which is a monoid in **DA**: by Theorem D.6, we know that K' is a union of monomials.

Our goal is then to reduce to $\mathrm{CTS}[K']$. Formally, we construct from the A^*-DAG $G = (V, E, \lambda)$ the M-DAG $G' = (V, E, \lambda')$ where we define $\lambda'(v) := \eta(\lambda(v))$ for all $v \in V$. Intuitively, we have relabeled the multi-letter labels of G to single-letter labels in M. We claim that I is a positive instance to the CTS-problem for S iff G' is a positive instance to

CTS[K']. This will allow us to conclude, because, by Theorem 4.3 and Corollary 4.2, we know that CTS[K'] is in NL.

To show the equivalence, we will show that for any topological sort σ of (G, V), the word $\lambda(\sigma)$ achieved by σ in G is in K iff the word $\lambda'(\sigma)$ achieved by σ in G' is in K'. In other words, letting $w_1 \cdots w_n := \lambda(\sigma)$, we must show that $w_1 \cdots w_n \in K$ iff $\eta(w_1) \cdots \eta(w_n) \in K'$. The forward direction is immediate by applying the morphism η. For the backward direction, we have $\psi(\eta(w_1) \cdots \eta(w_n))) \in \eta(K)$, and the left-hand-side is $\eta(w_1) \cdots \eta(w_n)$, which is $\eta(w_1 \cdots w_k)$ because η is a morphism, so applying η^{-1} concludes. We have shown the equivalence, so we can reduce in NL to CTS[K'] with K' a union of monomials, which establishes NL-membership.

Proof of (b). Assume that $T(S)$ is not in **DA**. Remember that $T(S)$ is still aperiodic because S is counter-free. Hence, we can apply Proposition D.7: there exists a morphism $\theta : \{a, b\}^* \to M$, a set $P \subseteq M$, and a regular language $H \in \{(ab)^*, (ab + b)^*\}$ such that $\theta^{-1}(P) = H$. Our goal is to use θ and P to define a set of pairs of initial and final states of S so that the CSh-problem for S with these states reduces in logspace to the corresponding problem for H. To do this, let $x := \theta(a)$ and $y := \theta(b)$. As these are elements of the transition monoid, we can pick $u, v \in A^*$ such that $f_u = x$ and $f_v = y$, which we will use to define our reduction.

Let $G = (V, E, \lambda)$ be an instance of the CSh-problem for H. Let us build $G' = (V, E, \lambda')$ where we define $\lambda'(w) := \theta(\lambda(w))$ for all $w \in V$. For each function $f \in P$, let us define an instance I_f of the semiautomaton CSh-problem of S by $I_f = (G', (q_1, \{f(q_1)\}), \ldots, (q_n, \{f(q_n)\}))$ where $(q_i)_{i=1,\ldots,n}$ is an arbitrary enumeration of Q, the set of states of S. Note that a word $z \in A^*$ is accepted by S for the choice of initial and final states in I_f iff $f_z = f$ in M. This construction is in NL. Let us show that G is a positive instance to CSh[H] iff one of the I_f is a positive instance to the semiautomaton CSh-problem of S, which shows that our reduction is correct (but note that this is not a many-one reduction).

For the forward direction, assume that we have a topological sort σ of (V, E) achieving a word $z := \lambda(\sigma)$ of H, and let us consider the word $\lambda'(\sigma) = \theta(z_1) \cdots \theta(z_n) = \theta(z_1 \cdots z_n)$ because θ is a morphism. As $z \in H$ and $\theta(H) = P$, we know that $f := \theta(z_1 \cdots z_n)$ is in P. Hence, consider the instance I_f. We know that $f_z = f$ by definition, hence σ witnesses that I_f has a suitable topological sort.

For the backward direction, assume that there is $f \in P$ such that we have a solution of I_f. This means that there is a topological sort σ of (V, E) such that the word $z := \lambda'(\sigma)$ achieved by σ in G' is such that $f_z = f$. Now, we know that $\theta^{-1}(f) \subseteq H$. Hence, the word $\lambda(\sigma)$ achieved by σ in G is in H, so G is a positive instance to CSh[H], which establishes the desired equivalence.

We have thus shown a reduction from CSh[H] to the multi-letter CSh-problem for the semiautomaton S. We can then conclude that the latter problem is NP-hard, because CSh[H] is NP-hard: either $H = (ab)^*$ and this follows from Theorem 3.6, or $H = (ab + b)^*$, in which case we conclude from Proposition 3.9. ◀

E Proofs for Section 6 (Lifting the Counter-Free Assumption for CSh)

E.1 Proof of Theorem 6.1: Coarser Dichotomy Theorem for CSh

We first explain how Theorem 6.1 follows from Theorem 6.2, before dealing with the much more difficult task of proving Theorem 6.2. The overall scheme is like in Section 5: show that monoids in **DO** can be reduced to tractable languages (specifically, to district group monomials), and show that monoids not in **DS** capture an intractable language. For the upper bound, we use the following result, which is the counterpart of Theorem D.6 but for **DO** rather than **DA**:

▶ **Theorem E.1 ([TT05], Theorem 1).** *Let K be a regular language of A^*. The following conditions are equivalent:*

- *K is an union of* unambiguous *district group monomials, i.e., of district group monomials $K = K_1 a_1 \cdots K_n a_n K_{n+1}$ such that every word $u \in K$ has a unique decomposition $u_1 a_1 \cdots u_n a_n u_{n+1}$ where $u_i \in K_i$ for all $1 \leq i \leq n+1$.*
- *There exists a monoid M in **DO** and a morphism $\eta : A^* \to M$ such that K is* recognized *by M, meaning that $K = \eta^{-1}(P)$ for some subset $P \subseteq M$.*

For the lower bound, we use the following folklore result, which extends Proposition D.7 to the non-aperiodic case:

▶ **Proposition E.2 ([Alm94], Exercise 8.1.6).** *A monoid M is not in **DS** iff there exists a morphism $\theta : \{a,b\}^* \to M$ and $P \subseteq M$ such that $\theta^{-1}(P)$ is either $(ab)^*$ or $(ab+b)^*$.*

From these two results, we can prove Theorem 6.1 exactly like we proved Theorem 5.2 in the previous section, using Theorem 6.2 instead of Theorem 4.3. The hard work that remains is to prove Theorem 6.2.

E.2 High-Level Presentation of the Proof of Theorem 6.2

This appendix gives a high-level view of the proof of Theorem 6.2. For most of the proof, we focus on the case of *group languages*: the case of group district monomials is only presented at the very end, in Appendix E.6. The CSh-problem for group languages can essentially be stated directly in terms of the underlying group: we fix a finite group H and a target element g, our instance to the CSh-problem is a tuple I of strings over H, and we want to test if there is an interleaving of I which evaluates to g according to the group operation. So we see $A := H$ as the alphabet of I.

As explained in the proof sketch, given the CSh-instance I, we will split the letters of A between *rare* letters A_{rare} and *frequent* letters A_{freq}, which we call a *rare–frequent partition*. This will ensure that the rare letters A_{rare} only occur in constantly many input strings (called the *rare* strings), and the frequent letters A_{freq} occur in sufficiently many different input strings (called the *frequent* strings).

For the frequent letters, the key idea is that we can pick many occurrences of each frequent letter in different strings, and obtain an *antichain* C (subset of pairwise incomparable elements), which contains many occurrences of each frequent letter of A_{freq}. Now, in a topological sort, we can enumerate all elements of C contiguously, following any permutation on C. Intuitively, as C contains many occurrences of each frequent letter, this should give us the freedom to create many different elements in the subgroup of H generated by A_{freq}. We cannot obtain *all* elements of this subgroup, because the number of occurrences of each group element is fixed by that of C. To formalize this intuition, the notion of *Parikh image* is helpful:

▶ **Definition E.3.** Write the alphabet A as $a_1, \ldots, a_k$ in some fixed order. The *Parikh image* of a word $w \in A^*$ is $\text{PI}(w) := (|w|_{a_1}, \ldots, |w|_{a_k}) \in \mathbb{N}^k$, where $|w|_a$ for $a \in A$ denotes the number of occurrences of a in w. The *Parikh image* of a language K is then the set $\text{PI}(K)$ of the Parikh images of the words that K contains: for instance, $\text{PI}((ab)^*) = \{(i,i) \mid i \in \mathbb{N}\}$.

The *Parikh image* $\mathrm{PI}(G)$ of an A-DAG $G = (V, E, \lambda)$ is $(|G|_{a_1}, \ldots, |G|_{a_k})$, with each $|G|_{a_i}$ being $|\{v \in V \mid \lambda(v) = a_i\}|$. The *Parikh image* $\mathrm{PI}(I)$ of a CSh-instance I is defined in the same way, seeing I as a DAG formed of disjoint paths.

As it will turn out, the Parikh image is the only constraint on what we can generate using such an antichain C. We formalize this intuition in the *antichain lemma* (Lemma E.4): we show that, for any finite group, if we have enough copies of each element, we can permute them to realize any element of the group, up to "commutative constraints". Thanks to this, the CSh-problem simply reduces to a test on the Parikh image $\mathrm{PI}(I)$ of the instance, under our initial assumption.

We must now explain how to handle the rare letters A_{rare}. We can simply look at the constant number of strings that contain a letter of A_{rare}, and handle these strings with an approach based on dynamic programming, as in the proof of Proposition C.2. So it seems like the problem is solved: apply dynamic programming to the rare strings, and use the antichain lemma to argue that the frequent strings can generate any letter of the subgroup spanned by A_{freq}, up to the commutative constraints. However, one difficulty remains: in a topological sort of the rare strings, we can insert elements from the frequent strings at any point in the dynamic algorithm, and the rare strings may be arbitrarily long; yet the frequent strings cannot create arbitrarily many copies of each group element, because we must use a constant bound when splitting H into A_{rare} and A_{freq}. We address this by proving a result called the *insertion lemma* (Lemma E.5), which intuitively says that a constant number of insertions always suffice. This is the result whose proof uses Ramsey's theorem. Thanks to the insertion lemma, it suffices to allow constantly many insertions of frequent elements when performing the NL algorithm on the rare strings, which allows us to conclude.

We give some more detail by stating the antichain lemma and insertion lemma as standalone results (and defer their complete proof to the next sections of the appendix, i.e., Appendices E.3 and E.4). We then formalize the rare–frequent partition and sketch the remainder of the proof of Theorem 6.2 (the details about the reminder of the proof are given in Appendix E.5).

Antichain lemma. Let G be an A-DAG over some alphabet A, let C be an antichain of G, and let $n \in \mathbb{N}$. We call C an *n-rich antichain* if each letter of A appears at least n times in C. The *antichain lemma* intuitively shows that when G has a rich antichain, then it suffices to look at commutative information of G, namely, its Parikh image, to decide whether it has a topological sort that achieves a group element. In fact, the claim applies to any constant-length sequence of group elements, following our needs for the insertion lemma later. Formally:

▶ **Lemma E.4 (Antichain lemma).** *Let H be a finite group and $\mu : A^* \to H$ be a surjective morphism. For any integer $k > 0$, there exists an integer n_k such that, for any A-DAG $G = (V, E, \lambda)$ with an n_k-rich antichain, for any elements $g_1, \ldots, g_k$ of H, if $\mathrm{PI}(G) \in \mathrm{PI}(\mu^{-1}(g_1 \cdots g_k))$ then there is a topological sort σ of G decomposable as $\sigma = \sigma_1 \cdots \sigma_k$ such that $\mu(\lambda(\sigma_i)) = g_i$ for each $i \in \{1, \ldots, k\}$.*

Note that this result is not specific to the CSh-problem, and applies to arbitrary DAGs. We now sketch its proof here; the complete proof is given in Appendix E.3:

Proof sketch. We capture the "commutative information" contained in the Parikh image of the rich antichain as an element in a commutative monoid N constructed from the commutative closure of H. The elements that we can hope to reach with the antichain are then the images of this element of N by a so-called *relational morphism* [Eil74] written $\tau : N \to \mathcal{P}(H)$. Intuitively, for $n \in N$ capturing some "commutative information", $\tau(n)$ are the elements of H which correspond to this information. We then study the elements of N that use sufficiently many copies of each generator of N, called the *fully recurrent* elements, and show that their images by τ all have the same cardinality. In other words, all antichains that are sufficiently rich can achieve the same number of elements of H. This allows us to conclude, because making the antichain richer always allows us to reach more elements, so an antichain which is richer than this threshold always achieves the maximal possible number of elements. ◀

Insertion lemma. We now turn to the *insertion lemma*, which allows us to show that we only need to insert group elements at a constant number of places. More precisely, when we achieve a group element by interleaving two sequences, we can always interleave them differently so that there are constantly many insertions and still achieve the same element.

▶ **Lemma E.5 (Insertion lemma).** *Let H be a finite group and $\mu : A^* \to H$ be a surjective morphism. There exists a constant $B \in \mathbb{N}$ such that, for any $n \in \mathbb{N}$, for any n-tuple $w_1, \ldots, w_n$ of words of A^* and $(n+1)$-tuple $w'_0, \ldots, w'_n$ of words of A^*, letting $u = w'_0 w_1 w'_1 w_2 w'_2 \cdots w_n w'_n$, there exists a set $J \subseteq \{0, \ldots, n\}$ of cardinality at most B such that, letting w''_j for $0 \leq j \leq n$ be w'_j if $j \in J$ and the empty word otherwise, letting $v = w''_0 w_1 w''_1 \cdots w_n w''_n$, we have $\mu(u) = \mu(v)$ and $\mu(w'_0 \cdots w'_n) = \mu(w''_0 \cdots w''_n)$.*

We give a sketch of the result; the complete proof is presented in Appendix E.4:

Proof sketch. We reason on the complete graph of positions of the word u, coloring each edge by three group elements derived from the corresponding factor: the group element achieved when performing the insertions (from u), the group element achieved when we do not perform them (from v), and the group element achieved by the insertions on their own (from the w'_i). We then use Ramsey's theorem to extract a monochromatic triangle in this graph: we show that, in the factor spanned by this triangle, there is no difference between performing the insertions and not performing them. We can repeat this argument as long as the word has sufficiently many letters, so we reach a constant bound B which comes from Ramsey's theorem. ◀

Putting the proof together. We are now ready to explain at a high level the rest of the proof of Theorem 6.2 in the case of group languages. Let K be a group language on the alphabet $A = \{a_1, \ldots, a_k\}$. We let $\mu : A^* \to H$ be the syntactic morphism of K, where H is a finite group generated by the $\mu(a_i)$. We consider an instance $I = (S_1, \ldots, S_n)$ to the CSh-problem, where each S_i is a string of vertices labeled with letters of the alphabet A. Let B be the bound whose existence is shown in Lemma E.5, and, using Lemma E.4 for the value $k := B$, let R be the value of n_k given by this lemma. We will decompose I following a *rare–frequent partition*, which we now define:

▶ **Definition E.6.** A *rare–frequent* partition of I consists of a partition of A into *rare* letters A_{rare} and *frequent* letters A_{freq}, and a partition of the strings into *rare* strings S_{rare} and *frequent* strings S_{freq}, where all vertices of S_{freq} are labeled with letters of A_{freq}, and where S_{freq}, when seen as an subinstance of I over the alphabet A_{freq}, contains an R-rich antichain.

Note that, in a partition, rare strings may still contain arbitrarily many frequent letters, and rare letters may still occur a unbounded number of times overall in I, as they can occur arbitrarily many times in each rare string. We can then show the following:

▶ **Lemma E.7.** *For any fixed alphabet A of size k, given an input CSh-instance $I = (S_1, \ldots, S_n)$, we can compute a rare–frequent partition of I in NL, represented as the partition $A_{\text{freq}} \sqcup A_{\text{rare}}$ of A and the set of rare strings S_{rare}, such that $|S_{\text{rare}}| \leq R \cdot k^2$.*

Proof. We first argue for the existence of a suitable rare–frequent partition by giving a naive algorithm to construct it, and then justify that we can do it in NL instead.

The naive algorithm initializes $A_{\text{rare}} = \emptyset$, $A_{\text{freq}} = A$, $S_{\text{rare}} = \emptyset$, $S_{\text{freq}} = S$, and does the following until convergence: if a letter $a \in A_{\text{freq}}$ occurs in less than $R \cdot k$ strings of S_{freq}, then remove a from A_{freq}, add a to A_{rare}, remove the $\leq R \cdot k$ strings that contain a from S_{freq}, and add them to S_{rare}. As we perform the move operation at most once for each letter, it is immediate that the algorithm terminates, and that at the end there are at most $R \cdot k^2$ rare strings: now the definition of the algorithm clearly ensures that S_{freq} cannot contain any letter of A_{rare} and that each letter of A_{freq} occurs in at least $R \cdot k$ different strings of S_{freq}. By picking R strings of S_{freq} for each letter of A_{freq} in a way that does not overlap, we see that S_{freq} contains an R-rich antichain for the alphabet A_{freq}. Hence, a suitable rare–frequent partition exists.

To construct the rare–frequent partition in NL, simply guess the partition $A_{\text{rare}} \sqcup A_{\text{freq}}$ of A, guess the set S_{rare} of rare strings of size $\leq R \cdot k^2$ (which is constant), guess R occurrences for each letter of A_{freq}, check that they are all in different strings and that they are not in strings of S_{rare}, and check that the strings which are not in S_{rare} contain only frequent letters. ◀

Hence, we assume that we have computed in NL a rare–frequent partition of I, given by A_{rare}, A_{freq}, S_{rare}, and (implicitly) S_{freq}. We write H_{freq} for the subgroup of H equal to $\mu(A^*_{\text{freq}})$, i.e., the subgroup spanned by A_{freq}. We can now sketch the remainder of the proof of Theorem 6.2:

Proof sketch. Our goal is to determine whether I has some topological sort in K. We relabel all elements of I with their image in H by μ, and equivalently test whether I has a topological sort achieving a target group element $g \in H$. We do so by an NL algorithm: we perform the analogue of Proposition C.2 on the rare strings S_{rare}, with some insertions of a constant number of elements from H_{freq} which respect the constraints on the Parikh image (again formalized via the notion of relational morphisms). To show correctness, we rely on the antichain lemma (Lemma E.4) to argue that any such pattern of insertions can indeed be performed using S_{freq}, thanks to the rich antichain that it contains. To show completeness, we rely on the insertion lemma (Lemma E.5) to argue that any topological sort achieving an element of H can indeed be rewritten to an equivalent one where we only perform constantly many insertions. ◀

In the rest of the appendix, we first prove the antichain lemma in Appendix E.3, and then prove the insertion lemma in Appendix E.4. We then complete our presentation of the proof of Theorem 6.2 for group languages in Appendix E.5, using the two lemmas and some of the notions introduced in Appendices E.3 and E.4. Last, we extend the proof to district group monomials in Appendix E.6.

E.3 Proof of Lemma E.4: Antichain Lemma

To prove the antichain lemma, let us fix the finite group H and morphism μ. Recall the definition of the Parikh image (Definition E.3), and let us define the *commutative closure* $\mathrm{CCl}(K)$ of a regular language K as $\mathrm{PI}^{-1}(\mathrm{PI}(K))$, where PI denotes the Parikh image

(Definition E.3). Remark that, for any element $g \in H$, the inverse image $\mu^{-1}(g)$ is a group language. Relying on some more standard notions from algebraic language theory, we will say that a language K is *recognized* by the morphism μ if there exists $P \subseteq H$ such that $K = \mu^{-1}(P)$. We will also talk about the *syntactic monoid* of K, which is the transition monoid of the minimal automaton which recognizes K.

We will use the following result on the group languages defined as $\mu^{-1}(g)$ for $g \in H$:

▶ **Lemma E.8 ([GGP08], Theorem 3.1).** *The commutative closure of a group language is regular.*

Remark that this result does not hold for the commutative closure of arbitrary regular languages (e.g., $(ab)^*$), and that the commutative closure of a group language is not necessarily a group language (see [GGP08] for a counterexample). Let us accordingly define a finite monoid N, and let $\mathrm{Com}_\mu : A^* \to N$ be a surjective morphism such that, for each $g \in H$, the morphism Com_μ recognizes $\mathrm{CCl}(\mu^{-1}(g))$. We can construct N, for instance, by taking the direct product of the syntactic monoids recognizing the commutative closure of each $\mu^{-1}(g)$, using Lemma E.8. Further, thanks to commutativity, we can choose N to be a finite commutative monoid. Let ω be a positive *idempotent power* of N, that is, a value $\omega \in \mathbb{N} \setminus \{0\}$ such that we have $p^{2\omega} = p^\omega$ for every $p \in N$. (Such an idempotent power exists: indeed, for every p in N, there exists k such that $p^k = p^{2k}$, and we can take ω to be the least common multiple of the idempotent powers of all elements of N.)

To characterize the "commutative information" of elements of H, we will study the connection between H and the commutative monoid N. We will do so using relational morphisms. A *relational morphism* [Eil74] between two monoids M and M' is a map from M to the powerset $\mathcal{P}(M')$ of M', such that for all $m \in M$ we have $\tau(m) \neq \emptyset$, and for all $m, m' \in M$, we have $\tau(m) \cdot \tau(m') \subseteq \tau(mm')$, where we extend the product operator of M' to the powerset monoid of M' in the expected way, that is, $S \cdot S' = \{g \cdot g' \mid g \in S, g' \in S'\}$. For any surjective morphism $\eta : A^* \to M$ and morphism $\mu : A^* \to M'$, the map $m \mapsto \mu(\eta^{-1}(m))$ is a relational morphism. We write $\tau : M \overset{\triangleleft}{\to} M'$ if τ is a relational morphism between M and M'.

We can now introduce the crucial notion of *fully recurrent* elements for our purposes, which will formalize the connection to rich antichains. An element p of a commutative monoid N is said to be *fully recurrent* if there exists a generator S of N and *positive* integers $r_1, \ldots, r_n$ such that $p = s_1^{r_1} \cdots s_n^{r_n}$, where $n = |S|$, and $r_i \geq \omega$ for all $1 \leq i \leq n$.

The notion of fully recurrent elements is motivated by the following lemma:

▶ **Lemma E.9.** *Let $\tau : N \overset{\triangleleft}{\to} H$ be any relational morphism from a commutative monoid to a finite group. For any fully recurrent elements p and q of N, the sets $\tau(p)$ and $\tau(q)$ have the same size.*

Proof. We will show the result using the following claim (*): for any fully recurrent element r, we have $|\tau(r)| = |\tau(r^i)|$ for any $i \geq 1$. This suffices to conclude the lemma, because for any fully recurrent elements p and q, we have $p^\omega = q^\omega$. Indeed, writing $p = s_1^{r_1} \cdots s_n^{r_n}$, we have $p^\omega = (s_1^\omega)^{r_1} \cdots (s_n^\omega)^{r_n} = s_1^\omega \cdots s_n^\omega$, and similarly for q. This allows us to conclude from (*) because we have $|\tau(p)| = |\tau(p^\omega)| = |\tau(q^\omega)| = |\tau(q)|$.

So we simply show claim (*). Let r be a fully recurrent element, and let us study the sequence (x_i) defined by $x_i := |\tau(r^i)|$ for all $i \geq 1$. We must show that the sequence (x_i) is constant. We do this in two parts: (i) we show that it is nondecreasing, and (ii) we show that there are arbitrary large $b \in \mathbb{N}$ such that $x_b = x_1$. Parts (i) and (ii) clearly imply that the sequence is constant, which establishes (*).

For part (i), we show that $|\tau(r^i)| \leq |\tau(r^{i+1})|$ for all $i \geq 1$. By definition of relational morphisms, we have $\tau(r^i)\tau(r) \subseteq \tau(r^{i+1})$. Now, remembering that the empty set is not in the image of a relational morphism, pick any $x \in \tau(r)$. We know that $\tau(r^i) \cdot \{x\} \subseteq \tau(r^i)\tau(r)$. Now, as $x \in H$ and H is a group, we know that H acts bijectively on any subset of H, in particular $\tau(r)$, hence $|\tau(r^i)| = |\tau(r^i) \cdot \{x\}| \leq |\tau(r^i)\tau(r)| \leq |\tau(r^{i+1})|$. This shows part (i).

We now show part (ii). To do so, let us show first that $r^{\omega+1} = r$. Indeed, write $r = s_1^{r_1} \cdots s_n^{r_n}$, and we simply conclude using the fact that $s_i^{r_i+\omega} = s_i^{r_i-\omega}(s_i^{\omega})^2 = s_i^{r_i-\omega}s_i^{\omega} = s_i^{r_i}$. This implies that we have $r^{j\omega+1} = (r^{\omega})^j r = r^{\omega}r = r$, for any $j \geq 0$. As $\omega \geq 1$, there are arbitrarily large values of $j\omega$, so this concludes part (ii) and we have established claim (*), which finishes the proof. ◀

We are now ready to show the antichain lemma (Lemma E.4). Recall its statement:

Lemma E.4: Let H be a finite group and $\mu : A^* \to H$ be a surjective morphism. For any integer $k > 0$, there exists an integer n_k such that, for any A-DAG $G = (V, E, \lambda)$ with an n_k-rich antichain, for any elements $g_1, \ldots, g_k$ of H, if $\mathrm{PI}(G) \in \mathrm{PI}(\mu^{-1}(g_1 \cdots g_k))$ then there is a topological sort σ of G decomposable as $\sigma = \sigma_1 \cdots \sigma_k$ such that $\mu(\lambda(\sigma_i)) = g_i$ for each $i \in \{1, \ldots, k\}$.

Proof of Lemma E.4. Fix the finite group H, and let $\mu : A^* \to H$ be the surjective morphism. We fix $\gamma = \max_{g \in H} \min_{u \in \mu^{-1}(g)} |u|$: this value is well-defined because μ is surjective, and is finite because H is finite. Let $\mathrm{Com}_\mu : A^* \to N$ be the surjective morphism defined as before, where N is a commutative monoid, and let ω be the idempotent power of N. Finally, let $\tau : N \overset{\triangleleft}{\to} H$ be the relational morphism defined by $\tau(x) = \mu(\mathrm{Com}_\mu^{-1}(x))$. Observe that the Parikh image assumption on the input A-DAG G and on the $g_1, \ldots, g_k$ in the statement of the lemma is equivalent to $\mathrm{Com}_\mu(G) \in \mathrm{Com}_\mu(\mu^{-1}(g_1 \cdots g_k))$. Indeed, the forward implication is immediate, and the converse holds because Com_μ recognizes $\mathrm{CCl}(\mu^{-1}(g_1 \cdots g_k))$, so the rephrased condition implies that $\mathrm{CCl}(G) \in \mathrm{CCl}(\mu^{-1}(g_1 \cdots g_k))$, which clearly implies the original condition. Further, by composing with τ and simplifying using the definition of τ, the condition rephrases to $g_1 \cdots g_k \in \tau(\mathrm{Com}_\mu(G))$. We will use this equivalent rephrased condition throughout the proof.

Let us now show the result by induction on $k > 0$. For every k, we will choose $n_k := \omega + (k-1)\gamma$. Let us first show the base case for $k = 1$ and $n_k = \omega$. Let $G = (V, E, \lambda)$ be the input A-DAG to the CTS-problem, and let us study the set $T = \{\mu(\lambda(\sigma)) \mid \sigma \text{ is a topological sort of } G\}$. Remembering that all topological sorts of G have the same Parikh image, namely, $\mathrm{PI}(G)$, we know from the commutativity of N that all topological sorts of G have the same image by Com_μ, namely, $\mathrm{Com}_\mu(G)$. Hence, T is included in $\tau(\mathrm{Com}_\mu(G))$. Our goal is to show that, when G has a ω-rich antichain, we have $T = \tau(\mathrm{Com}_\mu(G))$. Indeed, in this case, we know that, for any g_1 such that $\mathrm{PI}(G) \in \mathrm{PI}(\mu^{-1}(g_1))$, we have $g_1 \in \tau(\mathrm{Com}_\mu(G))$ as we explained above, so $g_1 \in T$ and there is a topological sort $\sigma := \sigma_1$ of G such that $\mu(\lambda(\sigma_1)) = g_1$. So all that remains to show for the base case is that $T = \tau(\mathrm{Com}_\mu(G))$.

Let C be a ω-rich antichain of G. For simplicity, let us make C *maximal*: whenever some vertex x of G is not in C but is incomparable to all vertices of C, we add it to C. We choose the vertices arbitrarily. At the end of the process, C is still an antichain, and it is still ω-rich. Further, we can partition G as $G^- \sqcup C \sqcup G^+$, where G^- contains all vertices having a directed path of positive length to a vertex of C, and G^+ contains all vertices having a directed path of positive length from a vertex of C. To see why this is a partition, observe that it covers G because any counterexample vertex x would contradict the maximality of C. Further, C is disjoint from G^+, and from G^-, because it is an antichain, and G^+ and G^-

are disjoint: any element in $G^+ \cap G^-$ would witness by transitivity a path from an element of C to an element of C, contradicting the fact that C is an antichain.

Let σ^- and σ^+ be arbitrary topological sorts of G^- and G^+ respectively. Our chosen partition ensures that we can build a topological sort of G as $\sigma^-, \sigma, \sigma^+$ where σ is a topological sort of C. Hence, $T' := \mu(\sigma^-) \cdot \tau(\mathrm{Com}_\mu(C)) \cdot \mu(\sigma^+)$ is a subset of T, so $|T'| \leq |T|$. Let us now write $s_a := \mathrm{Com}_\mu(a)$ for each letter $a \in A$. We can write $\mathrm{Com}_\mu(C) = \Pi_{a \in A} s_a^{i_a}$, where i_a is the number of vertices labeled by a in C. As C is ω-rich, we have $i_a \geq \omega$. Thus, $\mathrm{Com}_\mu(C)$ is fully recurrent by definition. Now, it is clear that $\mathrm{Com}_\mu(G)$ is also fully recurrent, because G is ω-rich also. Thus, by Lemma E.9, we have $|\tau(\mathrm{Com}_\mu(C))| = |\tau(\mathrm{Com}_\mu(G))|$. Now, we know that $\mu(\sigma^-)$ (resp. $\mu(\sigma^+)$) act bijectively on the left (resp. right) of H, so we also have $|\tau(\mathrm{Com}_\mu(C))| = |T'|$. We have thus shown that $|\tau(\mathrm{Com}_\mu(G))| = |T'| \leq |T|$. As $T \subseteq \tau(\mathrm{Com}_\mu(G))$, we deduce that $T = \tau(\mathrm{Com}_\mu(G))$. As we have argued, this concludes the proof of the base case $k = 1$.

We now prove the inductive step. Assume the property holds for $k > 0$. Let G be an instance of the CTS-problem that has a n_{k+1}-rich antichain: as in the base case we expand it to a maximal such antichain, denote it by C, partition G as $G^- \sqcup C \sqcup G^+$, and let σ^- and σ^+ be arbitrary topological sorts of G^- and G^+ respectively. Let us choose elements $g_1, \ldots, g_{k+1}$ of H such that $g_1 \cdots g_{k+1} \in \tau(\mathrm{Com}_\mu(G))$: remember that this implies that $g_1 \cdots g_{k+1} \in \tau(\mathrm{Com}_\mu(G))$.

Now, let us consider $g' := g_{k+1} \cdot \mu(\sigma^+)^{-1}$. Let $u_{g'} \in A^*$ be a word that realises the minimum in the definition of γ, and let $C_{g'}$ be a subset of C whose elements are labeled with the letters of $u_{g'}$. As C is n_{k+1}-rich, we can find such a subset, and further $C \setminus C_{g'}$ is still a $((k-1)\gamma + \omega)$-rich antichain, i.e., an n_k-rich antichain. Further, the definition of $C_{g'}$ ensures that it has a topological sort σ' that realizes the word $u_{g'}$, so that $\mu(\sigma') = g'$. By composing it with σ^+, we can then construct $\sigma'\sigma^+$, which is a topological sort of $G'' \sqcup C_{g'} \sqcup G^+$ such that $\mu(\lambda(\sigma'\sigma^+)) = g_{k+1}$.

We now wish to apply the induction hypothesis for $g_1, \ldots, g_k$ on the subinstance $G' := G^- \sqcup (C \setminus C_{g'})$, which still has a n_k-rich antichain. To do so, we must check that $\mathrm{PI}(G') \in \mathrm{PI}(\mu^{-1}(g_1 \cdots g_k))$, which as we argued is equivalent to $g_1 \cdots g_k \in \tau(\mathrm{Com}_\mu(G'))$. As G is the disjoint union of G' and G'', we have $\mathrm{Com}_\mu(G) = \mathrm{Com}_\mu(G')\mathrm{Com}_\mu(G'')$, so by composing by τ and applying the definition of a relational morphism we have:

$$\tau(\mathrm{Com}_\mu(G)) \subseteq \tau(\mathrm{Com}_\mu(G')\mathrm{Com}_\mu(G''))$$

Now, as both G and G' contain an antichain which is at least ω-rich, we know that $\mathrm{Com}_\mu(G)$ and $\mathrm{Com}_\mu(G)$ are fully recurrent. By applying Lemma E.9 again, we know that $|\tau(\mathrm{Com}_\mu(G))| = |\tau(\mathrm{Com}_\mu(G')|$. Remember now that $\sigma'\sigma^+$ is a topological sort of G'' such that $\mu(\lambda(\sigma'\sigma^+)) = g_{k+1}$. Hence, $g_{k+1} \in \tau(\mathrm{Com}_\mu(G''))$. Now, as g_{k+1} acts bijectively on $\tau(\mathrm{Com}_\mu(G'))$ in the group H, we deduce that $\tau(\mathrm{Com}_\mu(G)) = \tau(\mathrm{Com}_\mu(G'))g_{k+1}$. Now, since we have $g_1 \cdots g_{k+1} \in \tau(\mathrm{Com}_\mu(G))$ by hypothesis, we deduce that indeed $g_1 \cdots g_k \in \tau(\mathrm{Com}_\mu(G'))$, so we can apply the induction hypothesis.

Hence, we do so and obtain a topological sort $\sigma_1, \ldots, \sigma_k$ of G' such that $\mu(\lambda(\sigma_i)) = g_i$ for each $i \in \{1, \ldots, k\}$. Now, letting $\sigma_{k+1} := \sigma'\sigma^+$, it is clear that $\sigma_1, \ldots, \sigma_k, \sigma_{k+1}$ is a topological sort of G, and we have $\mu(\lambda(\sigma'\sigma^+)) = g_{k+1}$, so we have shown the induction hypothesis. This concludes the proof. ◀

E.4 Proof of Lemma E.5: Insertion Lemma

We now prove the insertion lemma (Lemma E.5). Recall its statement:

Lemma E.5: Let H be a finite group and $\mu : A^* \to H$ be a surjective morphism. There exists a constant $B \in \mathbb{N}$ such that, for any $n \in \mathbb{N}$, for any n-tuple $w_1, \ldots, w_n$ of words of A^* and $(n+1)$-tuple $w'_0, \ldots, w'_n$ of words of A^*, letting $u = w'_0 w_1 w'_1 w_2 w'_2 \cdots w_n w'_n$, there exists a set $J \subseteq \{0, \ldots, n\}$ of cardinality at most B such that, letting w''_j for $0 \leq j \leq n$ be w'_j if $j \in J$ and the empty word otherwise, letting $v = w''_0 w_1 w''_1 \cdots w_n w''_n$, we have $\mu(u) = \mu(v)$ and $\mu(w'_0 \cdots w'_n) = \mu(w''_0 \cdots w''_n)$.

Proof of Lemma E.5. Fix the alphabet A, the morphism μ, and the group H. By Ramsey's theorem, there exists a constant B such that, for any complete graph Γ whose edges are labeled with triples of elements of H, if Γ has at least B vertices, then it contains a monochromatic triangle, that is, three vertices v_1, v_2, v_3 such that the edges $\{v_1, v_2\}$, $\{v_2, v_3\}$, and $\{v_1, v_3\}$ are labeled by the same triple of elements of H.

Let us now show the rest of the claim by strong induction on $n \in \mathbb{N}$. The base case of the induction is when $n < B$, and in this case there is nothing to show: we can simply take $J = \{0, \ldots, n\}$ which achieves the cardinality bound, and then we have $u = v$ so clearly $\mu(u) = \mu(v)$.

Let us now show the induction step. We take an arbitrary $n \in \mathbb{N}$ with $n \geq B$, assume that the result is true for all smaller n, and show the result for n. Fix the words w_i and w'_i. Now, let us construct the complete graph Γ with n vertices $v_1, \ldots, v_n$ and with edges colored by triples of elements of H in the following way: the edge between v_i and v_j for $i < j$ is colored with the triple $(g_{i,j}, g'_{i,j}, g''_{i,j})$, where we define $g_{i,j} := \mu(w_i \ldots w_{j-1}))$, $g'_{i,j} := \mu(w_i w'_i \cdots w_{j-1} w'_{j-1})$, and $g''_{i,j} := \mu(w'_i \cdots w'_{j-1})$.

Now, by Ramsey's theorem, as Γ has more than B vertices, it has a monochromatic triangle. This implies that there are $1 \leq l < m < r \leq n$ such that $g_{l,m} = g_{m,r} = g_{l,r}$, and $g'_{l,m} = g'_{m,r} = g'_{l,r}$. Now, as by definition we have $g_{l,r} = g_{l,m} g_{l,r}$, this means that we have $g_{l,r} = g^2_{l,r}$, and as H is a group we can simplify and deduce that $g_{l,r} = e$, the neutral element of H. We deduce in the same way that $g'_{l,r} = e$. Hence, we have shown $g_{l,r} = g'_{l,r}$, which means that (*): $\mu(w_l w'_l \cdots w_{r-1} w'_{r-1}) = \mu(w_l \cdots w_{r-1})$. Further, we deduce in the same way that (**) $g''_{l,r} = e$.

We will now conclude using the induction hypothesis. Let $n' = n - (r - l)$, and consider the n'-tuple $w_1, \ldots, w_{l-1}, (w_l \cdots w_{r-1}), w_r, \cdots, w_n$ of words of A^*, and the $(n' + 1)$-tuple $w'_0, \ldots, w'_{l-1}, w'_r, \ldots, w'_n$. Using the induction hypothesis for n', we deduce the existence of $J' \subseteq \{0, \ldots, n'\}$ of cardinality at most B such that, defining w'''_j for all $0 \leq j \leq n'$ as the empty word if $j \notin J$, as w_j if $j \in J'$ and $j < l$, and as $w_{j+(r-l)}$ if $j \in J$ and $j \geq l$, letting

$$
\begin{aligned}
u' :=& w'_0 w_1 w'_1 \cdots w_{l-1} w'_{l-1} (w_l \cdots w_{r-1}) w_r w'_r \cdots w_n w'_n \\
v' :=& w'''_0 w_1 w''_1 \cdots w_{l-1} w'''_{l-1} (w_l \cdots w_{r-1}) w_r w''_r \cdots w_n w''_n
\end{aligned}
$$

we have $\mu(u') = \mu(v')$, and we have (***) $\mu(w'_0 \cdots w'_{l-1} w'_r \cdots w'_n) = \mu(w'''_0 \cdots w'''_n)$. Let us accordingly define $J \subseteq \{0, \ldots, n\}$ by $\{j \mid j \in J, j < l\} \sqcup \{j + (r - l) \mid j \in J, j \geq l\}$, which satisfies the cardinality bound. Let us show that $\mu(u) = \mu(v)$ and $\mu(w_0 \cdots w_n) = \mu(w'''_0 \cdots w'''_n)$ with v and the w''_i defined from this choice of J. From the equality (*), we know that we can replace $(w_l \cdots w_{r-1})$ by $(w_l w'_l \cdots w_{r-1} w'_{r-1})$ in u' without changing its image by μ, so we have $\mu(u) = \mu(u')$. Second, from the fact that J does not contain any element in $\{l, \ldots, r - 1\}$, we know that w''_j is empty for all $j \in \{l, \ldots, r - 1\}$, so we have $w_l \cdots w_{r-1} = w_l w''_l \cdots w_{r-1} w''_{r-1})$; further, from this and our definition of J, we observe that $v = v'$, hence $\mu(v) = \mu(v')$. We thus deduce that $\mu(u) = \mu(v)$. Last, we can use (**) to insert in (***) the product $w''_l \cdots w''_{r-1}$, to establish the second required equality. This concludes the proof. ◀

E.5 Proof of Theorem 6.2 for the Case of Group Languages

We give the complete proof of Theorem 6.2 for the case of group languages.

Let K be a group language on the alphabet $A = \{a_1, \dots, a_k\}$, let $\mu : A^* \to H$ be the syntactic morphism of K, where H is a finite group generated by the $\mu(a_i)$. Consider an instance $I = (S_1, \dots, S_n)$ to the CSh-problem, where each S_i is a directed path of vertices labeled with letters of the alphabet A. Recall from Appendix E.2 that B is the bound whose existence is shown in Lemma E.5, and, using Lemma E.4 for the value $k := B$, R is the value of n_k given by this lemma. Recall the definition of a *rare–frequent* partition of I (Definition E.6) from Appendix E.2, and recall that we have used Lemma E.7 to compute in NL a rare–frequent partition of I, given by A_{rare}, A_{freq}, S_{rare}, and (implicitly) S_{freq}. We write H_{freq} for the subgroup of H equal to $\mu(A^*_{\text{freq}})$, i.e., the subgroup spanned by A_{freq}.

Our goal is to determine whether I has some topological sort in K. This is the case iff it has a topological sort mapped to an accepting element of H by μ, so we can equivalently test, for each accepting element of H, whether there is a topological sort that achieves it. Hence, let g be the target element. Recall that the commutative closure of the language $\mu^{-1}(g)$ is a regular language by Lemma E.8, and is obviously commutative. Further recall the morphism $\text{Com}_\mu : A^* \to N$ from Section E.3, where N is a commutative monoid that recognises the inverse image of all elements of H, in particular g. Recall also the relational morphism $\tau : N \overset{\triangleleft}{\to} H$ defined by $\tau(x) = \mu(\text{Com}_\mu^{-1}(x))$.

We will state a condition, called (*), and construct an NL algorithm to check (*). We will then show that (*) holds iff I has a topological sort that achieves g. Condition (*) is: there exists a topological sort ρ of S_{rare} which can be decomposed as $\rho_1 \cdots \rho_n$, and a sequence $g_0, \dots, g_n$ of elements of H_{freq}, such that:

1. $g_0\mu(\lambda(\rho_1))g_1 \cdots \mu(\lambda(\rho_n))g_n = g$;
2. $g_0 \cdots g_n \in \tau(\text{Com}_\mu(S_{\text{freq}}))$;
3. $n < B$.

To test this condition (*), we simply nondeterministically guess a sequence S' of elements of H_{freq} of size at most B (i.e., a constant) such that the concatenation of its elements is in $\tau(\text{Com}_\mu(S_{\text{freq}}))$, add S' to S_{rare}, and check whether the resulting CSh instance has a topological sort using the NL algorithm of Proposition C.2 (because its number of strings is at most $R \cdot k^2 + 1$, which is constant): the language to test is $\mu^{-1}(g)$ on the modified alphabet where the elements of S' carry labels in H_{freq} and stand for themselves; note that this clearly yields a group language.

All that remains to show is that condition (*) is equivalent to the existence of a topological sort of I that achieves g. For the forward direction, assume that condition (*) holds. Recall that we have defined $R := n_B$. Focus on S_{freq}, which has an R-rich antichain for A_{freq}, and observe that $g_0 \cdots g_n \in \tau(\text{Com}_\mu(S_{\text{freq}})$, which is the equivalent rephrasing of the condition $\text{PI}(S_{\text{freq}}) \in \text{PI}(\mu^{-1}(g_0 \cdots g_n))$, as argued at the beginning of the proof. Using the antichain lemma (Lemma E.4), we know that there is a topological sort $\sigma = \sigma_0 \cdots \sigma_n$ of S_{freq} such that $\mu(\lambda(\sigma_i)) = g_i$ for each $i \in \{0, \dots, n\}$. Now, considering the topological sort $\rho_1, \dots, \rho_n$ of S_{rare} given by condition (*), it is clear that $\sigma_0\rho_1\sigma_1 \cdots \rho_n\sigma_n$ is a topological sort of I, built by interleaving S_{rare} and S_{freq}; and furthermore $\mu(\lambda(\sigma_0\rho_1\sigma_1 \cdots \rho_n\sigma_n)) = \mu(\lambda(\sigma_0))\mu(\lambda(\rho_1))\mu(\lambda(\sigma_1)) \cdots \mu(\lambda(\rho_n))\mu(\lambda(\sigma_n))$, which by (*) is equal to g, concluding the forward direction of the correctness proof.

We now show the backward direction. Assume that there is a topological sort σ' of I achieving g, i.e., $\mu(\sigma') = g$. We can decompose it as an interleaving of S_{rare} and S_{freq}, which we write $\sigma_0\rho'_1\sigma_1\cdots\rho'_{n'}\sigma_{n'}$, with $\rho'_1\cdots\rho'_{n'}$ being a topological sort of S_{rare}, and $\sigma_0\cdots\sigma_{n'}$ being a topological sort of S_{freq} (in particular, we have $\mu(\lambda(\sigma_0\cdots\sigma_{n'}))\in\tau(\text{Com}_\mu(S_{\text{freq}}))$, which we call condition (#2′)). We now use the insertion lemma (Lemma E.5) to argue that there exists a set $w_0,\ldots,w_{n'}$ of words of A^*, with $w_i=\lambda(\sigma_i)$ for at most B values of i and being the empty word otherwise, such that $\mu(w_0\lambda(\rho'_1)w_1\cdots\lambda(\rho'_{n'})w_{n'})=\mu(\sigma')=g$, and (#2″) $\mu(\lambda(\sigma_0\cdots\sigma_{n'}))=\mu(w_0\cdots w_n)$. We now collapse the ρ'_i which are contiguous, calling the result $\rho_1,\ldots,\rho_n$, where we have (#3) $n<B$, and write g_i the μ-image of the i-th w_i which is non-empty: this image is in H_{freq} because the strings in S_{freq} are only labeled with letters in A_{freq}. This gives us a topological sort $\rho_1,\ldots,\rho_n$ of S_{rare}, and a sequence $g_0,\ldots,g_n$ of elements of H_{freq}, such that (#1) $g_0\mu(\lambda(\rho_1))g_1\cdots\mu(\lambda(\rho_n))g_n=g$. By (#1), (#2′) combined with (#2″), and (#3), we have satisfied condition (*). This concludes the backward direction, and establishes the equivalence proof. Hence, we have shown Theorem 6.2 in the case of group languages.

E.6 Proof of Theorem 6.2 for the Case of District Group Monomials

We now show the complete proof of Theorem 6.2 by adapting the proof of Appendix E.5 from the case of group languages to that of district group monomials. We write $K=K_0a_1K_1\cdots a_mK_m$, where each a_i is a letter of the alphabet (they are not necessarily distinct), and each K_i is a group language on some subset A_i of the alphabet. We fix as before the instance $I=(S_1,\ldots,S_n)$ of the CSh-problem. A K-*slicing* of the instance I is an $(m+1)$-tuple of instances $I_0,\ldots,I_m$, with each I_j being a n-tuple $(S^j_1,\ldots,S^j_n)$ of strings, and an m-tuple of instances $I'_1,\ldots,I'_m$, with each I'_j being a n-tuple $((S')^j_1,\ldots,(S')^j_n)$ as before, with the stipulation that, for each $1\leq j\leq m$, all $(S')^j_i$ are empty except one which is a singleton whose only element is labeled a_j; and that, for each $1\leq i\leq n$, the concatenation $S^0_i(S')^1_iS^i_i\cdots(S')^m_iS^m_i$ is equal to S_i. In other words, a slicing is a partition of each string of I in a way that respects the a_i.

Intuitively, we would like to guess a slicing, check the I'_i in the obvious way, and apply the previous result to the I_j for odd j, corresponding to the group languages K_j. Unfortunately, while guessing the even I_j is immediate, we cannot afford to guess the entire slicing in NL. For this reason, we need a more elaborate approach.

We will follow the previous proof and introduce a notion of rare–frequent partition, generalised to slicings. As before, we let B be the bound whose existence is shown in Lemma E.5, use Lemma E.4 with $k:=B$ to obtain n_k, and let $R:=n_k$. Given a slicing $I_0\ldots I_m$ and $I'_1\ldots I'_m$, a *rare–frequent partition* of the slicing consists of one partition A^j_{rare}, A^j_{freq} for all $1\leq j\leq m$, and one *global* partition of the strings $S_1,\ldots,S_n$ into rare strings S_{rare} and frequent strings S_{freq} (again, the frequent strings are not explicitly represented). We require that (i) for every string S of S_{freq}, considering its slices $S^0,\ldots,S^m$, for each $1\leq j\leq m$, the slice S^j contains only letters of A^j_{freq}; that (ii) for every $1\leq j\leq m$, the S^j for S in S_{freq}, when seen as a subinstance of I over the alphabet A^j_{freq}, contains an R-rich antichain; and that (iii) for every $1\leq j\leq m$, the one non-empty string of I'_j is in S_{rare}.

We can show as before that, for any slicing, we can compute a rare–frequent partition. In fact we will only need to show that it exists, as the problem in guessing the slicing prevents us from guessing it anyway.

▶ **Lemma E.10.** *For any slicing $I_0,\ldots,I_m$, $I'_1,\ldots,I'_m$, there exists a rare–frequent partition such that $|S_{\text{rare}}|\leq m\cdot R\cdot k^2$.*

Proof. We apply Lemma E.7 to each I_j for $1\leq j\leq m$ to obtain one rare–frequent partition for it, written $A^j_{\text{rare}}\sqcup A^j_{\text{freq}}=A_j$ and $S^j_{\text{rare}}\sqcup S^j_{\text{freq}}=I_j$, except that we take $m\times(R+2)$

instead of m. Now, the only thing that remains is to justify that we can take the set of rare strings to be global instead of local, and to satisfy condition (iii). We simply then take S_{rare} to be the union of the strings S of I such that S^j is in S^j_{rare} for some $0 \leq j \leq m$, plus the strings that are non-empty in some I'_j. We take S_{freq} to be the complement. This ensures that condition (iii) is respected by construction. Now, it is clear that condition (i) is respected, as, for each slice, the frequent strings to consider are a subset of the one given by the previous condition. Now, condition (ii) is respected because it was respected initially for the richness threshold of $m \times (R+2)$, and we have only removed at most $m \times (R+1)$ frequent strings in the modification: $((m+1)-1) \times R$ for the other slices of the form I_j, and m for the slices of the form I'_j. Hence, we can deduce an R-rich antichain by looking at any preexisting $(m \times (R+2))$-rich antichain. ◀

While we cannot guess the slices, let us guess partitions $A_j = A^j_{\text{rare}} \sqcup A^j_{\text{freq}}$ for $0 \leq j \leq m$ and the set S_{rare} of (globally) rare strings of size at most $R \cdot k^2$. Let us further guess the slices S'_j for $1 \leq j \leq m$, i.e., we guess elements in I with suitable order and labels. As the number of rare strings is constant and m is constant, we guess, for each string of S_{rare}, the m points at which the slices end, i.e., we guess a slice but restricted to the rare strings. As for the frequent strings, we will not guess the slices globally, as there is generally a non-constant number of frequent strings. However, we will guess the "sequence of insertions" to be performed using the frequent antichains for each slice, i.e., the analogue to the sequence $g_0, \ldots, g_n$ in condition (*) in the previous proof. Formally, we guess a sequence $g^j_0, \ldots, g^j_{n_j}$ for all $0 \leq j \leq m$, with each g^j_i being an element of H^j_{freq}, the subgroup of H_j spanned by A^j_{freq}. Last, we also guess an element $\gamma_0, \ldots, \gamma_m$ of $H_0 \times \cdots \times H_m$ to describe the accepting elements of the H_i achieved in each slice.

Intuitively, we will now do two things: first, verify that our guesses are consistent (except for the choice of the γ_i); second, reduce the problem to a simpler problem by replacing all strings of S_{freq} with an additional string labeled directly with elements of the groups H_i of the group languages K_i, as in the previous proof.

First, to verify that our guesses are consistent, we check the rare strings. On these strings, it is straightforward to verify that the sub-alphabet for each slice is respected. Further, for the slices I'_j, the verification is immediate. Now, for the frequent strings, we go over them in succession. We maintain a state that stores, for each slice of the form I_j for $0 \leq j \leq m$, how many occurrences of each letter of A we have seen in the slice j, and in how many different strings are these occurrences. Initially, each letter occurs 0 times. Now, when processing a frequent string S which is in S_{freq} (i.e., not in S_{rare}), we guess a slicing of S, count the number of occurrences of each letter in each slice and add it to our counter of occurrences, and add one to the counter of strings for the symbols that did occur. At the end, we check that the value of our counters satisfies some conditions, which will witness the existence of a suitable slicing of the frequent strings. Specifically, we verify:

- For each $0 \leq j \leq m$, for each $a \in A \setminus A^j_{\text{freq}}$, that our choice of slicing does not contain any occurrence of a in the restriction of the slice I_j to S_{freq}.
- For each $0 \leq j \leq m$, for each $a \in A^j_{\text{freq}}$, that our choice of slicing ensures that there are at least R different strings that contain an occurrence of a in the restriction of slice I_j to S_{freq}, witnessing that it has an R-rich antichain for the alphabet A^j_{freq}.
- For each $0 \leq j \leq m$, letting w be the word containing all letters of the restriction of slice S_j to S_{freq} with the correct number of occurrences, that $g^j_0 \cdots g^j_{n_j} \in \tau_j(\text{Com}_{\mu_j}(w))$, intuitively checking that we have the right commutative image.

Second, we check the following condition (**), inspired from condition (*) in the previous proof: for all $0 \leq j \leq m$, there exist a topological sort $\rho_0^j \cdots \rho_{n_j}^j$ of the slice S_{rare}^j of S_{rare} whose concatenation, interleaved with the singleton elements of the I'_j, is a topological sort of S_{rare}, and $g_0^j \lambda(\rho_0^j) \cdots g_{n_j-1}^j \lambda(\rho_{n_j-1}^j) g^j n_j = \gamma_j$. This can be decided in NL by adapting the algorithm of Proposition C.2 as previously, running it on each slice with one additional string.

Overall, our algorithm succeeds iff there is a guess of γ_i, of S_{rare} (at most Rk^2 of them), partitions $A_{\text{freq}}^j \sqcup A_{\text{rare}}^j$, and sequences $g_0^j, \ldots, g_{n_j}^j$, such that the verification stage succeeds, and condition (**) holds.

We have described our NL algorithm. We now argue that it works as intended. There are two directions: the forward direction is to show that if the algorithm succeeds then there is a suitable topological sort of I, and the backward direction is to show the converse.

For the forward direction, assume that the algorithm succeeds. We deduce the existence of a set S_{rare} of rare strings (whose slices are written S_{rare}^j), and frequent strings S_{freq} (with the same convention for slices), partitions $A_{\text{freq}}^j \sqcup A_{\text{rare}}^j$, a slicing $I_0, \ldots, I_m$ and $I'_1, \ldots, I'_m$, a topological sort of S_{freq} constituting of topological sorts $\rho_0^j \cdots \rho_{n_j}^j$ of each S_{rare}^j for $0 \leq j \leq m$ interleaved with the singleton elements of the I'_j for $1 \leq j \leq m$, sequences $g_0^j, \ldots, g_{n_j}^j$ of elements of H_j for $0 \leq j \leq m$, and an element $\gamma_0, \ldots, \gamma_m$ of $H_0 \times \cdots \times H_m$, such that:

- For all $0 \leq j \leq m$, the element γ_j is accepting in H_j.
- For all $0 \leq j \leq m$, for all $S \in S_{\text{freq}}$, the slice S^j contains only letters from A_{freq}^j, and contains an R-rich antichain on the sub-alphabet A_{freq}^j.
- For all $0 \leq j \leq m$, for all $S \in S_{\text{rare}}$, the slice S^j contains only letters from A_j.
- For all $0 \leq j \leq m$, letting S_{freq}^j be the slice of S_{freq} defined in the expected way, we have $g_0^j \cdots g_{n_j}^j \in \tau_j(\text{Com}_{\mu_j}(S_{\text{freq}}^j))$.
- (#) For all $0 \leq j \leq m$, we have $g_0^j \lambda(\rho_0^j) \cdots g_{n_j-1}^j \lambda(\rho_{n_j-1}^j) g_{n_j}^j = \gamma_j$

We claim that we can deduce from this the existence of a witnessing topological sort. To do this, as before, we will use Lemma E.4 in the S_{freq}^j for all $0 \leq j \leq m$. From our definition of R, as $n_j < B$, as S_{freq}^j contains an n_k-rich antichain (seen as an instance on the sub-alphabet A_{freq}^j), as $g_1^j \cdots g_{n_j}^j \in \tau_j(\text{Com}_{\mu_j}(S_{\text{freq}}^j))$, there is a topological sort $\sigma_1^j \ldots \sigma_{n_j}^j$ of S_{freq}^j such that $\mu_j(\lambda(\sigma_i^j)) = g_i^j$ for each $0 \leq j \leq m$ and $1 \leq i \leq n_j$. This allows us to deduce our witnessing topological sort of I, consisting of a topological sort of each slice I_j of I achieving γ_j, interleaved with the trivial topological sorts of the I'_j that achieve the required a_j: the topological sort of I_j is formed of the guessed topological sort $\rho_0^j \cdots \rho_{n_j}^j$ of S_{rare}^j interleaved with the topological sort $\sigma_1^j, \ldots, \sigma_{n_j}^j$ of S_{freq}^j, each v_i^j achieving g_i^j, so that the topological sort of I_j indeed achieves γ_j by point (#).

We now show the backward direction. We show that if there is a suitable topological sort, then the algorithm succeeds. The witnessing topological sort must define a slicing of I such that each I_j for $0 \leq j \leq m$ has a topological sort achieving an element γ_j which is accepting for H_j. We now use Lemma E.10 to argue that there exists a rare–frequent partition consisting of a partition $S_{\text{rare}} \sqcup S_{\text{freq}}$ of the strings, and $A_{\text{rare}}^j \sqcup A_{\text{freq}}^j$ of the alphabets A_j, such that $|S_{\text{rare}}| \leq m \cdot R \cdot k^2$. In each slice, the witnessing topological sort must consist of a topological sort of the S_{rare}^j interleaved with topological sorts of the S_{freq}^j. As in the previous proof, we now use Lemma E.5 to argue that we can assume that there are at most n_j such insertions, without changing the μ_j-image of the result or the μ_j-image of the inserted elements. Now, we define the $g_1, \ldots, g_{n_j}^j$ as the μ_j-images of these insertions. We now consider the run of the algorithm where we guess the right rare–frequent partition, the right slices in the rare strings, the right topological sort of the rare strings.

We first check that the verification phase of the algorithm does not fail. This is the case: the first condition is by definition of a witnessing topological sort (for $A \backslash A_j$) and of a rare–frequent partition (for $A_j \backslash A^j_{\text{rare}}$); the second condition is by definition of a rare–frequent partition; the third condition is by definition of $g_1, \ldots, g^j_{n_j}$ being achieved as a topological sort of S^j_{rare}. We next explain why the second phase works, by explaining why condition (**) is satisfied. This can be seen by considering when the insertions of the S^j_{freq} are performed in the S^j_{rare}: we perform the same additions with the additional string. Hence, this run of the algorithm succeeds. This concludes the backwards direction of the correctness proof, so our NL algorithm is correct. This concludes the proof of Theorem 6.2.

E.7 Proof of Proposition 6.3: Example in DS \ DO

We show the side result on the language in **DS** \ **DO**. Note that the fact that this language is indeed in **DS** and not in **DO** can be simply checked from the equations that define **DS** and **DO**, as can be performed, e.g., using [Pap18].

▶ **Proposition 6.3.** *Let* $K = (a^+b^+a^+b^+)^*$*. Then* $\mathrm{CSh}[K]$ *is in NL.*

Proof. Consider an input instance I to the CSh-problem for K. Observe first that, if I has no string whose first element is a, then clearly no topological sort of I achieves K. Likewise, if I has no string whose last element is b, then clearly no topological sort of I achieves K. We can check these two conditions in NL and fail if one of them does not hold, so in the sequel we assume that I has a string whose first element is a and a string whose last element is b.

Recall that a 3-rich antichain for A in I is an antichain containing at least 3 elements labeled by a and 3 elements labeled by b. We show that if I contains a 3-rich antichain then it is necessarily a positive instance to $\mathrm{CSh}[K]$. Of course, note that we can easily test in NL if such a 3-rich antichain exists.

To show the claim, let C'' be such an antichain, and C' be a subset of C'' containing exactly three occurrences of each letter; it is still an antichain. We now define C as a subset of C' containing exactly two occurrences of each letter, and ensuring that there is an a-labeled element v_a which is the first element of a string and is not in a string of C, and likewise there is a b-labeled element v_b which is the last element of a string and is not in a string of C: we can ensure this because we can choose which a-labeled element and which b-labeled element we remove from C' to construct C.

Now, consider a topological sort σ_1 of I formed by concatenating v_a, a topological sort σ_- of the ancestors of elements of C and of the elements incomparable to C except v_a and v_b, a topological sort σ'_1 of C achieving the word $aabb$, a topological sort σ_+ of the successors of C, and v_b. The word w_1 achieved by σ_1 starts with a and ends with b, so it must be of the form $(a^+b^+)^*$. Let n_1 be the number of repetitions of a^+b^+ in w_1. Now, consider the topological sort σ_2 obtained by concatenating v_a, σ_-, σ'_2, σ_+, and v_b, where σ'_2 is a topological sort of C achieving the word $abab$. Again, the word w_2 achieved by σ_2 must be of the form $(a^+b^+)^*$: let n_2 be the number of repetitions of a^+b^+ in w_2. We claim that $n_2 = n_1 + 1$. Indeed, consider the subfactor a^+b^+ that contains σ'_1 in σ_1. In σ_2, the other subfactors are unchanged, and this subfactor is split into two subfactors, one ending at the first b of σ'_2, the other one starting at the second a of σ'_2. So indeed $n_2 = n_1 + 1$. Hence, one of n_1, n_2 is even, and the corresponding σ_i witnesses that I is a positive instance to $\mathrm{CSh}[K]$.

Hence, it suffices to handle the case where I has no 3-rich antichain. This implies that there is one symbol $\alpha \in A$ which occurs in at most two strings S and S', which means that the other strings $S_1, \ldots, S_m$ only contain elements labeled with the other symbol $\beta \neq \alpha$ of A. Now, it is easy to see that we obtain exactly the same topological sorts by merging together the $S_1, \ldots, S_m$ to one string S'' of elements labeled β whose length is $\sum_i |S_i|$. Hence, we can reduce the problem in NL to the instance $\{S, S', S''\}$. As it has three strings, we can conclude in NL using Proposition C.2. Hence, we have indeed shown that CSh[K] is in NL. ◀

References for the Appendix

ABDS17 Antoine Amarilli, M. Lamine Ba, Daniel Deutch, and Pierre Senellart. Possible and certain answers for queries over order-incomplete data. In *Proc. TIME*, 2017.

Alm94 J. Almeida. *Finite Semigroups and Universal Algebra*. Series in algebra. World Scientific, 1994.

Dil50 Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 1950.

Eil74 Samuel Eilenberg. *Automata, languages, and machines. Vol. A*. Academic Press, 1974. Pure and Applied Mathematics, Vol. 58.

Ful55 D. R. Fulkerson. Note on Dilworth's decomposition theorem for partially ordered sets. In *Proc. Amer. Math. Soc*, 1955.

GGP08 Antonio Cano Gómez, Giovanna Guaiana, and Jean-Eric Pin. When does partial commutative closure preserve regularity? In *Proc. ICALP*, 2008.

GJ75 Michael R. Garey and David S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM J. Comput*, 1975.

Imm88 Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput*, 17(5), 1988.

Pap18 Charles Paperman. Semigroup online, 2018. `https://www.paperman.name/semigroup/`.

PW97 Jean-Éric Pin and Pascal Weil. Polynominal closure and unambiguous product. *TCS*, 30(4), 1997.

Str94 Howard Straubing. *Finite automata, formal logic, and circuit complexity*. Birkhauser Verlag, 1994.

Sze88 Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3), 1988.

TT01 Pascal Tesson and Denis Thérien. The computing power of programs over finite monoids. *J. Autom. Lang. Comb.*, 7(2), 2001.

TT02 Pascal Tesson and Denis Thérien. Diamonds are forever: the variety DA. *Semigroups, algorithms, automata and languages*, 1, 2002.

TT05 Pascal Tesson and Denis Thérien. Complete classifications for the communication complexity of regular languages. *TCS*, 38(2), 2005.

WH84 Manfred K. Warmuth and David Haussler. On the complexity of iterated shuffle. *JCSS*, 28(3), 1984.