A Dichotomy on Constrained Topological Sorting

Antoine Amarilli$^1$ and Charles Paperman$^2$

1 LTCI, Télécom ParisTech, Université Paris-Saclay
antoine.amarilli@telecom-paristech.fr
2 Université Paris-Diderot, IMJ-PRG
charles.paperman@gmail.com

Abstract

We introduce the constrained topological sorting problem (CTS-problem): given a target language $L$ and a directed acyclic graph (DAG) $G$ with labeled vertices, determine if $G$ has a topological sort which forms a word that belongs to $L$. This natural problem applies to several settings, including scheduling with costs or verifying concurrent programs. It also generalizes the shuffle problem of formal language theory, which asks if a list of input strings has an interleaving that achieves a target string. We accordingly call constrained shuffle problem (CSh-problem) the restriction of our CTS-problem where the input DAG consists of disjoint strings.

We study the complexity of the CTS-problem and CSh-problem for regular target languages: for each fixed regular language $L$, we call $\text{CTS}(L)$ and $\text{CSh}(L)$ the corresponding problems, where the input is the DAG. Our goal is to characterize the regular languages for which these problems are tractable. We show that both problems are tractable (in NL) for unions of monomials, a useful language class for pattern matching, as well as some other cases. We extend this for the CSh-problem to unions of district group monomials. We also show NP-hardness for some other languages such as $(ab)^*$. These results lead to a dichotomy for a different problem phrasing, when the target is specified as a semiautomaton to enforce some closure assumptions, and where the semiautomaton is assumed to be counter-free. In this case, both problems are in NL if the transition monoid of the semiautomaton is in some class, and NP-hard otherwise. Without the counter-freeness assumption, we can extend the dichotomy to a partial result for the CSh-problem. Our proofs use a variety of tools ranging from complexity theory, combinatorics, algebraic automata theory, Ramsey’s theorem, as well as a custom reduction and other new techniques.

1 Introduction

Many scheduling or ordering problems amount to computing a topological sort of a directed acyclic graph (DAG), namely, find a totally ordered sequence of the vertices which is compatible with the edge relation: when we enumerate a vertex, all its predecessors must have been enumerated first.

However, in some settings, we wish to compute topological sorts satisfying additional constraints that cannot be expressed as DAG edges. In this paper, we formalize this problem as follows: the vertices of the DAG are labeled with some symbols from a finite alphabet, and we want to find a topological sort which falls into a specific regular language. We call this the CTS-problem. For instance, if we fix the language $K = ab^*c$, and consider the example DAGs of Figure 1, then DAGs (a) and (b) have a topological sort that falls in $K$.

The CTS-problem relates to many applications. For instance, many scheduling applications use a dependency graph of tasks, where we may need to impose some more 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 regular target 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 the
a Dichotomy on Constrained Topological Sorting

CTS-problem with a target language describing the behavior that we wish to detect. Our initial motivation for the CTS-problem comes from earlier work by the first author in uncertain data management applications [4]. We thus believe that the CTS-problem is useful, and natural, but we are not aware of previous studies of this problem. We can equivalently phrase the CTS-problem in terms of partial orders: seeing the labeled DAG as a labeled partial order $\prec$, we ask if some linear extension (total order compatible with $\prec$) achieves a word in the target language $K$. This is the problem phrasing that we use in [4], but we are also not aware of existing work under this terminology. The CTS-problem has received more attention under a restricted phrasing called the shuffle problem: the input DAG consists of a union of chains (aka line graphs, seen as strings), so the CTS-problem asks if these strings have some interleaving that falls in the target language $K$. We call constrained shuffle problem (CSh) the special case of the CTS-problem where the input DAGs obey this restriction. In Figure 1 (b) and (c) are shuffle instances. The shuffle operation has been studied in the context of concurrent programming languages [21, 24], computational biology [20], and formal languages [10, 7, 27]. In the latter context, the CSh-problem was shown to be NP-complete [22, 35, 19] when the target language $K$ is given as input and consists of just one string: we will build upon these results in our work.

Our goal in this paper is to study the complexity of the CTS-problem and CSh-problem, as a function of the target regular language $K$. To study more precisely the complexity of each language $K$, we assume that $K$ is fixed, and study complexity as a function of the input labeled DAG (on the alphabet of $K$): we call CTS($K$) the CTS-problem for the fixed language $K$, and CSh($K$) for the CSh-problem. This problem phrasing is inspired by the line of work on constraint satisfaction problems (CSPs) [11], which studies the complexity of homomorphism problems when the “constraints” (right-hand-side of the homomorphism) are fixed. However, the CTS-problem does not seem easy to rephrase in the CSP context: finding a topological sort subject to a regular language constraint seems hard to express in terms of homomorphisms, or even in terms of extensions such as temporal CSPs [5, 6]. This motivates our independent study of the CTS-problem, and the central question of this work: for which regular languages $K$ are the problems CTS($K$) or CSh($K$) tractable?

Of course, these problems can sometimes be quite easy, e.g., when $K$ is a commutative language like $a^*$ that does not depend on the order of letters. Further, as regular languages have PTIME membership testing, both problems are always in NP: we can simply guess a permutation of the DAG vertices, and check that it satisfies the order constraints and realizes a word in $K$. Further, both problems can be made tractable under some assumptions on the input DAG, e.g., CSh can be solved with dynamic programming when the number of input strings is constant, which was already observed in [4]. However, when allowing arbitrary labeled DAGs as input, it seems tricky to determine the languages for which the complexity of these problems is below NP.

We show in this work that the tractability boundary for these problems is surprisingly
challenging to chart out, and does not seem to map to well-known classes of regular languages. Specifically, the contributions of this work are as follows:

- We introduce the constrained topological sort (CTS) and constrained shuffle (CSh) problems, as natural abstractions of problems in formal language theory, parallel programming, and data management. We undertake the first study of their complexity for regular languages.
- We show that both problems are tractable (in NL) for languages $K$ that are unions of monomial languages, i.e., languages of the form $A_1^*a_1 \cdots A_n^*a_n$, with the $a_i$ being letters and the $A_i$ being subalphabets. These languages allow us in particular to search for a specific pattern, e.g., $A^*uA^*$ for a fixed word $u$.
- We show that the CSh-problem is tractable for group languages and unions of district group monomials, generalizing the above. Group languages are those where every letter acts bijectively, i.e., their underlying monoid is a group. Our results imply that, for any fixed finite group $H$, the following is in NL: 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 $g$ in the group $H$. The proof of this result is our main technical contribution.
- We show that the CSh-problem is NP-hard for the language $(ab)^*$ and several others. While [35] easily implies that CSh is NP-hard for some languages, some effort is required to derive this specifically for $(ab)^*$. Our proof relies on a general technique to show how our problems for some languages can reduce to the same problem for another language.
- In a restricted setting, and under suitable closure assumptions, we show a dichotomy theorem: the CTS- and CSh-problems are either in NL or NP-complete. This result applies to counter-free semiautomata and to a multiletter phrasing of the problem: this amounts to restricting to so-called aperiodic languages, and imposing closure under intersection, inverse morphism, complement, and left and right quotients. The dichotomy is effective: given a semiautomaton, we can decide whether the problems are in NL or NP-complete; further, the criteria for CTS and CSh in this context turns out to be the same.

2 Problem Statement and Main Results

We define formally the two problems that we study. We start with the constrained topological sort problem. We fix a finite alphabet $A$, and call $A^*$ the set of all finite words on $A$: we denote $\epsilon \in A^*$ the empty word. A labeled DAG on 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 $\lambda : V \rightarrow A$ is a function giving a label in $A$ to each vertex in $V$. A topological sort of an $A$-DAG is a bijective function $\sigma$ from $\{1, \ldots, n\}$ to $V$ such that, for all $(u, v) \in E$, we have $\sigma^{-1}(u) < \sigma^{-1}(v)$. We say that $\sigma$ achieves the word $\lambda(\sigma(1)) \cdots \lambda(\sigma(n)) \in A^*$; we abuse notation and write this word $\lambda(\sigma)$. The constrained topological sort problem (or CTS-problem) for a fixed language $K \subseteq A^*$ is written $\text{CTS}(K)$ and defined as follows: given an $A$-DAG $G$, determine whether there exists a topological sort $\sigma$ of $G$ such that $\lambda(\sigma) \in K$. We often abuse notation further and identify the topological sort $\sigma$ with the word $\lambda(\sigma)$ that it achieves. We also talk of $\sigma$ achieving $K$ to mean that it achieves a word in $K$. The language $K$ is generally infinite, and described, e.g., by a regular expression.

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

We now define the constrained shuffle problem. Given two words \( u, v \in A^* \), the shuffle \cite{4} of \( u \) and \( v \), written \( u \shuffle v \), is the set of words that can be obtained by interleaving \( u \) and \( v \). Formally, a word \( w \in A^* \) belongs to \( u \shuffle v \) iff there exists a partition \( P \cup Q \) of \( \{1, \ldots, |w|\} \) such that \( w_P = u \) and \( w_Q = v \), where \( w_P \) denotes the sub-word of \( w \) obtained by keeping the letters at positions in \( P \), and likewise for \( w_Q \). The shuffle of a tuple of words \( U \), written \( \shuffle(U) \), is defined by induction on \( |U| \) as follows: we set \( \shuffle(\{\} \) := \( \{\} \), \( \shuffle(u) := \{u\} \), and \( \shuffle(u_1, \ldots, u_n, u_{n+1}) := \bigcup_{v \in \shuffle(u_1, \ldots, u_n)} v \shuffle u_{n+1} \).

The constrained shuffle problem (or CSh-problem) for a fixed language \( K \) on \( A \), written \( \text{CSh}(K) \), is defined as follows: given a tuple of words \( U \), determine whether \( K \cap \shuffle(U) \) is nonempty.

The CSh-problem is a special case of the CTS-problem, because any tuple \( U \) of words of \( A^* \) can be coded as an \( A \)-DAG \( G_U \), by coding each \( u \in U \) as a line graph \( v_1 \rightarrow \cdots \rightarrow v_{|u|} \) with \( \lambda(v_i) = u_i \) for all \( 1 \leq i \leq |u| \). In this case, the set of words achieved by topological sorts of \( G_U \) is clearly equal to the set \( \shuffle(U) \). Thus, we will equivalently see inputs to the CSh-problem as tuples of words (called strings in this context) and as \( A \)-DAGs which are unions of line graphs.

In this work, we study the complexity of the problems \( \text{CTS}(K) \) and \( \text{CSh}(K) \) depending on the language \( K \). One important class of languages where these problems are easy are the commutative languages, which we define shortly. Writing the alphabet \( A \) as \( a_1, \ldots, a_k \) in some fixed order, recall that 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 commutative closure \( \text{CCI}(K) \) of a language \( K \) is then \( \text{PI}^{-1}(\text{PI}(K)) \), and \( K \) is commutative if it is equal to its commutative closure. For instance, \( \text{CCI}((ab)^*) = \{w \in A^* \mid |w|_a = |w|_b\} \); note that it is not a regular language.

In the case where \( K \) is a commutative language, \( \text{CTS}(K) \) and \( \text{CSh}(K) \) simply reduce to the word problem for \( K \), by taking any topological sort of the input. In other words, the problems only depend on the Parikh image \( \text{PI}(G) \) of the input \( A \)-DAG \( G = (V, E, \lambda) \), defined as \( (|G|_{a_1}, \ldots, |G|_{a_k}) \), each \( |G|_{a_i} \) being \( |\{v \in V \mid \lambda(v) = a_i\}| \). Hence, we will mostly focus on non-commutative languages in the sequel.

We first state some initial complexity bounds. We can always solve our problems for the language \( K \) 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 \( \text{CTS}(K) \) and \( \text{CSh}(K) \) are in NP\(^K \).

In particular, the problems are in NP when the language \( K \) is regular, as the word problem for regular languages is in PTIME.

We can refine this complexity analysis to make it dependent on some parameters of the input instance, specifically, the width of the input DAG, as we will now define. Given two vertices \( u \neq v \) of a DAG \( G \), we write \( u \prec \prec v \) if there is a (possibly empty) directed path from \( u \) to \( v \), and call \( v \) a descendant of \( u \) and \( u \) an ancestor of \( v \). If neither \( u \prec \prec v \) nor \( v \prec \prec u \) hold, we say that \( u \) and \( v \) are incomparable. 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. We can then show the following for regular languages, essentially following a dynamic algorithm:

**Proposition 2.3.** For any language \( K \), the problem \( \text{CTS}(K) \) 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 \( \text{CSh}(K) \).
Proof sketch. We use Dilworth’s theorem [8] which 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 perform a logspace algorithm following such a partition to guess an accepting path of a (fixed) automaton for $K$. This result follows Theorem 57 of [4], but there the complexity was stated as PTIME; our result is stronger, and requires additional work, namely, showing how to compute implicitly in logspace a chain partition which is minimal in a certain sense.

Other results are given in [4] about structural restrictions of the input DAG, so we do not focus on such assumptions in this paper. Instead, we study the complexity of CTS($K$) and CSh($K$) on arbitrary inputs, depending on the fixed language $K$. We believe that regular languages can be classified depending on the complexity of these problems, and make the following dichotomy conjecture:

Conjecture 2.4. For every regular language $K$, the problem CTS($K$) is either in NL or NP-complete. Likewise, the problem CSh($K$) is either in NL or NP-complete.

We have not been able to prove this conjecture in its full generality, or to completely characterize the class of tractable languages. This paper presents the results that we have obtained towards this end.

Paper structure. We first show hardness results (for both problems) for $(ab)^*$ and other regular languages, building on the results of [35] and extending them via a custom reduction technique (Section 3). We then show the tractability (for both problems) of unions of monomial languages (Section 4), as well as some other unrelated languages. These results allow us to give a partial answer to the above conjecture, with a dichotomy result under a different problem definition: we assume a family of input languages described as a semiautomaton, which enforces some closure properties, and assume that the semiautomaton is counter-free (Section 5). We last study whether one can lift the counter-free assumption in the case of the constrained shuffle problem, and show a generalization of our dichotomy result which implies tractability for group languages (Section 6). We conclude in Section 7. In the interest of brevity, the complete proofs of most results are deferred to the appendix.

3 Hardness Results

We start by presenting intractable cases of our two problems. As we pointed out in the introduction, some hardness results are already known in formal language theory for the shuffle problem, which is related to our CSh-problem. The shuffle problem asks, given a word $w$ and a tuple $U$ of words, whether $w \in \omega(U)$. This problem is known to be NP-hard already on the alphabet $\{a, b\}$ [35].

In this section, we build upon this result to show the hardness of CSh($K$), hence CTS($K$), for some regular languages $K$. Note that, from the NP-hardness of the shuffle problem above [35], we easily deduce NP-hardness of CSh($K$) for some specific choices of $K$, e.g., for $K_0 := (a_1 a_2 + b_1 b_2)^*$. Indeed, we can reduce a shuffle instance $(w, U)$ to the instance $I := w_1 \cup U_2$ for CSh($K_0$), where $w_1$ is $w$ but adding the subscript 1 to all letters, and $U_2$ is defined analogously. A topological sort of $I$ achieving $K_0$ must then enumerate identical letters alternatively from $w$ and $U$, and
conversely if \( w \in \mathcal{U}(U) \) then we can build a topological sort of \( I \) achieving \( K_0 \) in the same way, showing correctness of the reduction.

We will refine this approach to show the NP-hardness of CSh for \((ab)^*\) and other simple languages, by building on the hardness results of [35] and on custom tools. We first recall some initial hardness results from [35]. Second, we introduce our general notion of shuffle reduction, that allows us to lift these hardness results to other languages. We then use these tools to prove our hardness results, for \((ab)^*\) and for other languages.

**Hardness for an initial language family.** To bootstrap the hardness results on the shuffle problem (on input words) to our CSh-problem (on fixed languages), we define the intermediate notion of the CSh-problem on language families. A language family \( K \) is simply an infinite family of languages, and the CSh-problem for \( K \), written \( \text{CSh}(K) \), asks, given a language \( K' \in K \) and a set of strings \( U \), whether \( K' \cap \mathcal{U}(U) \) is nonempty. This means that the shuffle problem on input strings is definitionally equivalent to \( \text{CSh}(\mathcal{S}) \), where \( \mathcal{S} \) denotes the class of singleton languages (consisting of only one word). Hence, \( \text{CSh}(\mathcal{S}) \) is NP-hard; but a closer look at [35] yields the following:

**Lemma 3.1 ([35], Lemma 3.2).** Let \( K \) be the family \( \{(a^ib)^* \mid i \in \mathbb{N}_{>0}\} \). Then \( \text{CSh}(K) \) is NP-hard.

**Proof sketch.** We summarize the proof of [35]. The reduction is from UNARY-3-PARTITION: given a tuple \( E \) of \( 3m \) numbers written in unary and a bound \( B \in \mathbb{N}_{>0} \), decide whether the numbers can be partitioned into \( m \) triples, with each triple summing to \( m \). This problem is NP-hard [14].

Given a UNARY-3-PARTITION instance \((E, B)\), we create a CSh instance \( I \) by writing each number \( n \) as the string \( a^n b^n \), and we choose the target language \( K \) in \( K \) to be \((a^Bb^B)^*\), 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 can ensure that the number of such strings is exactly \( 3 \) by imposing some bounds on the values of the UNARY-3-PARTITION instance (see [35]). 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.

**Shuffle reduction.** Having shown the hardness of an initial language family, we now describe the reduction technique that we will use to establish hardness of \((ab)^*\) and other languages. We present the reduction for regular languages, although it applies to more general languages.

The intuition for the technique is as follows: we wish to reduce from \( \text{CTS}(K) \) to \( \text{CTS}(K') \) (or from \( \text{CSh}(K) \) to \( \text{CSh}(K') \)) for two regular languages \( K \) and \( K' \) on alphabet \( A \). To do so, given an input \( A \)-DAG \( G \), we build an \( A \)-DAG \( G' \) formed of \( G \) plus an additional chain labeled by a word \( w \). Thus, any topological sort \( \sigma' \) of \( G' \) must be the interleaving of \( w \) and of a topological sort \( \sigma \) of \( G \). Now, if we require that \( \sigma' \) achieves \( K' \), the presence of \( w \) can impose specific conditions on \( \sigma \). Intuitively, if \( w \) is very long but is "far away" from \( K' \), then \( \sigma' \) must "repair" \( w \) to a word of \( K' \) by inserting symbols from \( I \), so \( \sigma \) may be forced to enumerate \( G \) following \( K \). This intuition is illustrated on Figure 2 to achieve a word of \((ab)^*\) on the complete DAG \( G' \), a
topological sort must enumerate elements from $G$ to insert them at the appropriate positions in $w$, achieving a word of $(ba)^*b$. We call filter sequence a family of words such as $w$ that allow us to reduce any CTS($K$)-instance to CTS($K'$), and formalize it as follows:

**Definition 3.2 (Filter sequence).** Let $K$ and $K'$ be regular 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 the commutative closure $CCl(K)$ of $K$ such that $|v| = n$, we have $v \in K$ iff $(v \sqcup f_n) \cap K' \neq \emptyset$.

In Figure 2, $w$ is a possible choice for $f_5$ in a filter sequence for $(ba)^*b$ and $(ab)^*$: it ensures that, when interleaving $w$ with any DAG $G$ with $|G|_a = 2$ and $|G|_b = 3$, then a topological sort $\sigma$ of $G$ achieves $K$ iff some interleaving $\sigma'$ of $\sigma$ with $w$ achieves $K'$. We can now define the shuffle reduction:

**Definition 3.3 (Shuffle reduction).** We say that a regular language $K$ reduces to a regular language $K'$, and write $K \leq_{shuf} K'$, if there is a filter sequence $(f_n)$ for $K$ and $K'$ such that the function $n \mapsto f_n$ is computable by a logspace transducer. We say that a family $\mathcal{K}$ of regular languages reduces to $K'$ if each $K$ does, and there is a logspace transducer that computes the function $K, n \mapsto f_n$, which maps a language $K$ of $\mathcal{K}$ and an integer $n$ to the $n$-th element of a filter sequence for $K$ and $K'$.

The definition of a filter sequence ensures that the shuffle reduction implies a logspace reduction for the CTS-problem. As we just add a disjoint chain in the reduction, the same applies to the CSh-problem. Formally, we have:

**Theorem 3.4.** For any family $\mathcal{K}$ of regular languages and regular language $K'$, if $\mathcal{K} \leq_{shuf} K'$ then there is a logspace reduction from CTS($K$) to CTS($K'$), and from CSh($\mathcal{K}$) to CSh($K'$).

**Hardness for $(ab)^*$.** We now apply our reduction technique to show the hardness of the language $(ab)^*$ from that of the language family in Lemma 3.1. This will be instrumental to our dichotomy in Section 5.

**Theorem 3.5.** The problem CSh($(ab)^*$) (hence CTS($(ab)^*$)) is NP-hard.

**Proof.** Let $\mathcal{K}$ be the family of regular languages defined in Lemma 3.1. For each such language $K_B = (aBb)^*$, noting that the lengths of words in $K_B$ are of the form $2Bn$, we define the filter sequence as $f_{2Bn}^B := (b^B a^B b)^n$ for each $n \in \mathbb{N}$. To show that each $(f_n^B)$ is indeed a filter sequence for $K_B$ and $(ab)^*$, fix $n \in \mathbb{N}$, and consider a word $v \in CCl(K_B)$ such that $|v| = 2Bn$: necessarily we have $|v|_a = |v|_b = Bn$. Now, it is clear that, when interleaving any word with $f_{2Bn}^B$ to form a word of $(ab)^*$, we can achieve a word of $(ab)^*$ by inserting the letters in bold: $((ab)^B (ab)^B)^n$. Conversely, we are forced to insert at least these letters, so we must insert $Bn$ times $a$ and $Bn$ times $b$ in $f_{2Bn}^B$ to achieve a word of $(ab)^*$: as we have $|v|_a = |v|_b = Bn$,
we know that \((v \sqcup f_{2Bn}^B) \cap (ab)^*\) is non-empty if we can do these exact insertions in this order, i.e., if \(v = (aBbB)^n\). This shows that \(f_{2Bn}^B\) is indeed a filter sequence, which establishes that \(\text{CSh}((ab)^*)\) is NP-hard thanks to Theorem 3.4.

**Hardness for other languages.** From the hardness of \((ab)^*\), using our notion of reduction, we can 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 characters:

▶ **Proposition 3.6.** Let \(u \in A^*\) such that \(|u|_a > 0\) and \(|u|_b > 0\) for \(a \neq b\) in \(A\). Then \(\text{CSh}(u^*)\) (hence \(\text{CTS}(u^*)\)) is NP-hard.

**Proof sketch.** We use the reduction of Theorem 3.4 and let the filter sequence be \(f_{2n} := (uu_{-a}uu_{-b}u)^n\), where \(u_{-a}\) (resp. \(u_{-b}\)) are \(u\) but with one occurrence of \(a\) (resp. \(b\)) removed. If a word \(v\) with \(|v|_a = |v|_b = n\) is such that \(f_{2n} \sqcup v\) intersects \(u^*\) nontrivially, then \(v\) must intuitively insert one \(a\) in each \(u_{-a}\) and one \(b\) in each \(u_{-b}\). This can be formalized by ensuring that \(u\) has been rotated so that its first and last letters are different. We can then reason on factors of the interleaving of length \(|u|\) centered on the \(u_{-a}\) and \(u_{-b}\), plus the first or last character of \(u\), to argue that they cannot have the right number of \(a\)’s or \(b\)’s for a factor of \(u^*\) of this length unless some additional symbol was inserted in them.

We can also use our reduction technique to show hardness for other languages not of this form, e.g., \((aa + bb)^*\):

▶ **Proposition 3.7.** Let \(L := (aa + bb)^*\). The problem \(\text{CSh}(L)\) (hence \(\text{CTS}(L)\)) is NP-hard.

**Proof sketch.** We use again the reduction of Theorem 3.4 with the filter sequence \(f_n = (ab)^n\). If a word \(v\) with \(|v|_a = |v|_b = n\) is such that \(v \sqcup f_n\) intersects \((aa + bb)^*\) nontrivially, it must intuitively insert \(a\)’s and \(b\)’s in \(f_n\) alternatively, so be in \((ab)^*\).

The proof straightforwardly extends to show NP-hardness of the CSh-problem for \((a^i + b^j)^*\) whenever \(i, j \geq 2\). One last result, which is pretty easy but will be useful for our dichotomy, is that the hardness of some languages can also be shown by considerations on the Parikh image of an instance. In particular:

▶ **Proposition 3.8.** The problem \(\text{CSh}((ab + b)^*)\) (hence, \(\text{CTS}((ab + b)^*)\)) is NP-hard.

**Proof.** Simply reduce \(\text{CSh}((ab)^*)\) to \(\text{CSh}((ab + b)^*)\) in PTIME: given an instance \(I\), check if \(\text{PI}(I) \in \text{PI}((ab)^*)\) and fail if not. Otherwise, then \(I\) achieves a word of \((ab + b)^*\) iff it achieves one of \((ab)^*\): intuitively, we do not have enough \(b\)’s to enumerate one without an \(a\).

We believe that our reduction technique extends to many other languages, though we are unable to characterize which ones. In particular, we believe that the following could be shown via our reduction, generalizing all the above hardness results except Proposition 3.7:

▶ **Conjecture 3.9.** 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 \(\text{CSh}(F^*)\) is NP-hard.

### 4 Tractability Results for the DAG Problem

Having shown hardness for several languages, we now present our tractability results. These results will allow us to claim our dichotomy theorem in the next section.
Closure under union. The first important observation about tractability is that tractable languages are closed under the union operator, in the following sense (remember the definition of our problems for language families in Section 3):

Lemma 4.1. For any finite family of languages $K$, letting $K' = \bigcup K$, there is a logspace reduction from $CTS(K')$ to $CTS(K)$, and likewise from $CSh(K')$ to $CSh(K)$.

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

This immediately implies:

Corollary 4.2. For any finite family of languages $K$, if $CTS(K')$ is in NL for each $K' \in K$, then so is $CTS(\bigcup K)$. The same is true of the CSh-problem.

Clearly, tractability for our problems is also preserved under the reverse operator, i.e., reversing the order of words in a language, as can be seen by reversing the direction of edges in the input DAG. However, there are many other operations to which tractability for our problems does not extend; see Section 5. Closure under union is nevertheless useful and will be used often in the sequel.

Monomials. The main family of languages for which we can show that the CTS-problem is tractable are the monomial languages (and unions of such languages, by the above). Having fixed the alphabet $A$, a monomial language is a language of the form $A_1^*a_1A_2^*a_2\cdots a_nA_{n+1}^*$ 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, the language $A^*uA^*$ is a monomial language for every word $u \in A^*$. 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 $\Sigma_2[\prec]$, that is, first-order formulas with quantifier prefix $\exists^*\forall^*$. Furthermore, it is decidable to check if a given regular language is in this class: see [26]. We accordingly show:

Theorem 4.3. For any monomial language $K = A_1^*a_1A_2^*a_2\cdots A_n^*a_nA_{n+1}^*$, the CTS-problem for $K$ is in NL.

Proof sketch. We can first guess in NL the vertices $v_1, \ldots, v_{n+1}$ to which the $a_1, \ldots, a_{n+1}$ are mapped, so all that remains is to find, for each such guess, whether we can match the remaining vertices should be matched to the $A_i$. We proceed by induction on $n$. The base case of $n = 0$ is trivial because $A_i^*$ is commutative. For the induction step, we check that the descendants of $v_{n+1}$ are all in $A_{n+1}^*$, and then we compute the set $S$ of vertices that must be enumerated before $v_{n+1}$: 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 in $A_1^*a_1\cdots A_{n-1}^*a_{n-1}A_n^*$.

Tractability based on width. We can see that unions of monomial languages are not the only tractable languages. For instance, we can show the following result, in an entirely different way:

---

1 An implementation of this method is available online: see [http://paperman.cadilhac.name/pairs/](http://paperman.cadilhac.name/pairs/)
Proposition 4.4. Let $A := \{a, b\}$ and $K := (ab)^* + A^*aaA^*$. The problem $\text{CTS}(K)$ (hence $\text{CSh}(K')$) 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).

Proof. Let $G$ be an input $A$-DAG. We first check in NL if $G$ contains two vertices $v_1 \neq v_2$ such that $\lambda(v_1) = \lambda(v_2) = a$ and $v_1$ and $v_2$ are incomparable, i.e., neither $v_1 \leadsto v_2$ nor $v_2 \leadsto v_1$. If yes, we conclude that $G$ is a positive instance, as we can clearly find a topological sort where $v_1$ and $v_2$ are enumerated contiguously, achieving $K$.

If there are no two such vertices, then the restriction of the order induced by $G$ on the vertices labeled by $a$ must be a total order. This hypothesis makes it easy to check in NL whether $G$ has a topological sort achieving $K$. Indeed, it is then easy to check whether two of the $a$-labeled vertices can be enumerated contiguously, i.e., there are two vertices $v_1, v_2$ such that $\lambda(v_1) = \lambda(v_2) = a$ and $v_1 \rightarrow v_2$ but there is no vertex $w \notin \{v_1, v_2\}$ such that $v_1 \leadsto w \leadsto v_2$; if yes, we succeed. Otherwise, we know that we cannot achieve $A^*aaA^*$, so it suffices to test whether we can achieve $(ab)^*$. As we know that there is one $b$-element that must be enumerated between each pair of consecutive $a$-elements on the chain, this is the case iff there are no additional $b$-elements except exactly one, which we must be able to enumerate after the last $a$. This is clearly testable in NL, which concludes.

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 can generalize this observation for the constrained shuffle problem on the alphabet $\{a, b\}$: if we assume that the number of strings containing $a$ is bounded by a constant, then the CSh-problem for any regular language is clearly in NL by a dynamic programming argument like in Proposition 2.3. Hence, for instance, the languages $(ab)^* + A^*a^iA^*$ for any $i \in \mathbb{N}$ are tractable for the CSh-problem. This observation, suitably generalized, will be important to handle the case of groups in Section 6. However, outside of the specific case of the proposition above, we do not know whether this observation generalizes to the CTS-problem.

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

Proposition 4.5. Let $A := \{a, b\}$ and $K := (aa + b)^*$. The problem $\text{CSh}(K)$ is in NL.

We do not know the complexity of this language for the CTS-problem, 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 Dichotomy Theorem

In the two previous sections, we have established some tractability and intractability results about the constrained topological sort and constrained shuffle problems. We have shown several different types of techniques to establish these results, but we have not proposed any general theory of what makes a language tractable or intractable. In this section, however, we will show that our results already suffice to show a dichotomy that characterizes the tractable and intractable regular languages, up to changing the problem phrasing and making some additional assumptions. First, we will restrict to so-called counter-free languages, an assumption that we will partially lift in Section 6 for the CSh-problem. Second, we will rephrase our problems in a way that impose closure under some operations. Indeed, unlike the union operator (see Lemma 4.1), the tractable languages for our problems are not closed under intersection, inverse morphisms, complement, and left and right quotients, as we will illustrate. This makes it harder to characterize the individual behavior of all languages, whereas enforcing closure under these operations makes it possible to leverage existing algebraic tools from automata theory.

In this section, we first illustrate why tractable languages are not closed under these operations. We then rephrase our problem in different terms to impose closure, and state our dichotomy theorem.

**Closure counterexamples.** We first illustrate that tractable languages for the constrained shuffle problem are not closed under quotients. 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. As our problem is symmetric under reverse, it suffices to consider left quotients:

- **Proposition 5.1.** There exists a word $u \in A^*$ and a regular language $K$ such that $\text{CSh}(K)$ is tractable but $u^{-1}K = (ab)^*$, so that $\text{CSh}(u^{-1}K)$ is NP-hard by Theorem 3.5.

  **Proof sketch.** Taking $K := b^*A^* + aaA^* + (ab)^*$, we know that $(ab)^{-1}K = (ab)^*$, yet any shuffle instance with more than one string satisfies $K$, so $\text{CSh}(K)$ is in NL.

However, it is easy to notice that CTS-tractable languages are closed under quotients:

- **Proposition 5.2.** For any word $u \in A^*$ and regular language $K$, there is an logspace reduction from $\text{CTS}(u^{-1}K)$ to $\text{CTS}(K)$.

  **Proof sketch.** We can simply add an $u$-labeled chain in the DAG with edges to all other elements to show the logspace reduction. Note that this would not work for the CSh-problem.

Incidentally, from the two previous propositions, letting $K$ be the language of the proof of Proposition 5.1, we know that $\text{CTS}(K)$ is NP-hard but $\text{CSh}(K)$ is in NL: this is our first example of a language that separates the two problems. Second, we illustrate that tractable languages are not closed under the intersection operator, for both problems:

- **Proposition 5.3.** There exists two regular languages $K_1$ and $K_2$ such that $\text{CTS}(K_1)$ and $\text{CTS}(K_2)$ are both in PTIME, but $K_1 \cap K_2 = (ab)^*$, so that $\text{CSh}(K_1 \cap K_2)$ is NP-hard by Theorem 3.5.

  **Proof sketch.** We take $K_1 := (ab)^*(\epsilon + bA^*)$ and $K_2 := (ab)^*(\epsilon + aaA^*)$. We have $K_1 \cap K_2 = (ab)^*$, but each of these languages can be shown to be tractable by an ad-hoc greedy algorithm that runs in PTIME. Note that we do not show that $\text{CTS}(K_1)$ and $\text{CTS}(K_2)$ are in NL, although we conjecture that this should hold.
Third, we show that tractable languages are not closed under complement:

**Proposition 5.4.** There exists a regular language $K$ such that $CTS(K)$ is in NL, but $A^* \setminus K = (ab)^*$, so that $CSh(A^* \setminus K)$ is NP-hard by Theorem 3.5.

**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 $CTS(K)$ is in NL, however by construction we have $A^* \setminus K = (ab)^*$. ◀

Fourth, we can see that tractable languages are not closed under inverse morphisms. 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 $\varphi$ is the language over alphabet $B$ defined by $\varphi^{-1}(K) := \{ v \in B^* \mid \varphi(v) \in K \}$. We show:

**Proposition 5.5.** There exists a regular language $K$ and morphism $\varphi$ such that $CTS(K)$ is in NL, but $\varphi^{-1}(K) = (ab)^*$, so that $CSh(\varphi^{-1}(K))$ is NP-hard by Theorem 3.5.

**Proof.** We take $A := \{a,b\}$ and $K := (ab)^* + A^*aaA^*$, as in Proposition 4.4. We know by this proposition that $CTS(K)$ is in NL. However, let $\varphi : A^* \to A^*$ be defined by $\varphi(a) := aba$ and $\varphi(b) := b$. We then have $\varphi^{-1}(K) = (ab)^*$. Indeed, no word in the image of $\varphi$ has two consecutive $a$’s, so $\varphi^{-1}(K) = \varphi^{-1}((ab)^*)$, which is clearly equal to $(ab)^*$. ◀

**Redefining the problem.** We have explained why the constrained shuffle and constrained topological sort problems are not closed under usual operations. To achieve a dichotomy result in a restricted setting, we rephrase the definition of these problems in a way which enforces closure under the operations that we have seen so far. This simplifies our study substantially: when we enforce closure by these operations, and some regular language $K$ has an NP-hard quotient, or complement, etc., then $K$ is NP-hard when classified in this sense. The point of studying this simpler problem is that we can then rely on the tools of algebraic automata theory to achieve a dichotomy.

We will define these problems using *semiautomata*, which are a special case of deterministic state machines [17]. A *semiautomaton* is an automaton where initial and finite states are not specified. Formally, it is a tuple $(Q, A, \delta)$ where $Q$ is the set of states, $A$ is the alphabet, and $\delta : Q \times A \to A$ is the transition function. A regular language $K$ is computed by a semiautomaton $(Q, A, \delta)$ if there exists a choice of initial and finite states $q_0 \in Q$ and $F \subseteq Q$ such that the automaton $(Q, A, \delta, q_0, F)$ computes $K$. We will phrase the CTS- and CSh-problems using fixed semiautomata, where the problem input also specifies a set of initial-final state specifications: this enforces closure under quotients (by choosing the initial and final states), complement (by toggling the final states), and intersection (by imposing a logical AND over each initial-final specification in the set).

To finally close under inverse morphisms, we will study the *multi-letter CTS-problem*: we assume that the input instance is given as a $(V, E, \lambda)$ called an $A^*$-DAG, where $(V, E)$ is a DAG and $\lambda$ is a labeling function from $V$ to $A^*$ (i.e., we label each vertex with a word of $A^*$). As before, a topological sort $\sigma$ of $G$ achieves a word $\lambda(\sigma) \in A^*$ obtained by concatenating the $\lambda$-images of the vertices of $G$ in the order of $\sigma$. Hence, the vertices in the multiletter phrasing are labeled by “atomic” words, in the sense that the content of these words cannot be interleaved with anything else.

We can now define formally the new problem that we study, namely, the *multi-letter CTS-problem* for a semiautomaton $S = (Q, A, \delta)$. The input contains an $A^*$-DAG and a set
\[ \{ (i_0, F_0), \ldots , (i_k, F_k) \} \] where \( i_j \in Q \) and \( F_j \subseteq Q \) for all \( 0 \leq j \leq k \), specifying the choices of initial and final states. The input is accepted if there exists a topological sort \( \sigma \) of \( G \) such that for all \( 0 \leq j \leq k \), the word \( \lambda (\sigma) \) is accepted by the automaton \((Q, A, \delta, i_j, F_j)\).

In this rephrased problem setting, our dichotomy in this section only applies counter-free semiautomata, which amounts to a restriction on the languages covered by our dichotomy result. We say that 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 \( \delta(q, u) = q \).

**Dichotomy theorem.** We can now state our dichotomy theorem:

> **Theorem 5.6.** 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 common tractability criterion for CTS and CSh intuitively asks whether \( S \) captures \((ab)^*\) or a closely related language. Formally, as we will explain, it asks if the transition monoid of \( S \) is in the class \( \text{DA} \) of monoids \[32\]. This implies that the dichotomy is effective: we can decide, given \( S \), which case applies, and this meta-problem is PSPACE-complete \[34\].

In the remainder of this section, we prove Theorem 5.6. For the proof, we first recall some standard algebraic tools on semiautomata, and then show the result.

**Algebraic prerequisites on semiautomata.** Recall that a monoid is a set with an associative binary composition law and a neutral element. We will accordingly define the transition monoid of a semiautomaton \( S = (Q, A, \delta) \). For each letter \( a \in A \), we can define the function \( f_a : Q \to Q \) such that \( f_a(q) := q' \) if \( \delta(q, a) = q' \). Likewise, for \( u \in A^* \), we can define \( f_u : Q \to Q \) in the analogous way: notice that this ensures that \( f_e \) is the identity function, and \( f_{uv} \) for all \( u, v \in A^* \) is \( f_v \circ f_u \), where \( \circ \) denotes function composition. The transition monoid of \( S \) is then the couple \( T(S) = (E, \cdot) \), where the set \( E \) is defined as \( \{ f_u \mid u \in A^* \} \), and where the operator \( \cdot \) is function composition (written implicitly). Note that \( E \) is necessarily finite, because \( Q \) is. 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.

Our result focuses on counter-free semiautomata, and it is well-known (see \[23\]) that a semiautomaton \( S \) is counter-free if and only if its transition monoid \( T(S) \) is aperiodic. We say that \( M \) is an aperiodic monoid if it satisfies the equation \( x^\omega = x^{\omega+1} \), 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 5.6 relies on their transition monoid, and corresponds to the standard class \( \text{DA} \) of aperiodic monoids which has other connections to complexity theory (see \[32\]). A monoid \( M \) is in \( \text{DA} \) if and only if its idempotent power is finite. Our characterization result relies on the following characterization of \( \text{DA} \):

> **Theorem 5.7 (\[32\], 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 \( u_1a_1 \cdots u_n a_n u_{n+1} \) where \( u_i \in A_i^* \) for all \( 1 \leq i \leq n + 1 \).

- There exists a monoid \( M \) in \( \text{DA} \) 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 5.8 ([31], 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 sketch. The first direction is to show that, whenever the transition semigroup of a counter-free semiautomaton \( S \) is in \( DA \), then the multi-letter CTS-problem for \( S \) is in NL. Given an input language defined by \( S \) and an input instance, we reduce it to the CTS-problem in the usual sense by rewriting it to the transition monoid itself (which we use as the alphabet). We then use Theorem 5.7 to argue that the target language in the transition monoid is a union of (unambiguous) monomials, so the latter problem can be solved in NL.

For the second direction, we show that, when the transition semigroup is counter-free but not in \( DA \), then we can use Proposition 5.8 and reduce from the CSh-problem in the usual sense for one of the languages \( (ab)^* \) and \( (ab+b)^* \). Both of these were shown to be NP-hard in Section 3 (Theorem 3.5 and Proposition 3.8).

6 Lifting the Counter-Free Restriction for the CSh-Problem

Our dichotomy theorem in the previous section has two main drawbacks. First, it phrases our problems in terms of semiautomata, to enforce closure under several operations. Second, even within this framework, it only applies to counter-free semiautomata. This second restriction amounts to requiring that the transition monoid of the semiautomaton is aperiodic, which excludes in particular the case where this monoid is a group. In this section, we show how we can lift this second restriction, and show complexity results for arbitrary semiautomata, without the counter-freeness assumption. The key ingredient will be to handle group languages (i.e., languages whose syntactic monoid is a group), which is our main technical result.

Yet, our results when lifting the counter-freeness assumption will have two main limitations. The first limitation is that they will only apply to the constrained shuffle (CSh) problem, not the constrained topological sort (CTS) problem. The second limitation is that we will not show a complete dichotomy: we will show tractability for one class, called \( DO \), and hardness for the complement of a larger class, called \( DS \), leaving a gap between the two classes.

Let us now define the two classes \( DO \) and \( DS \) by extending the definitions of the previous section. Recall that our main dichotomy result (Theorem 5.6) distinguished tractable and intractable semiautomata \( S \) based on their transition monoid \( T(S) \). The tractable monoids were those in the class \( DA \), namely, those satisfying the equation \( (xy)^\omega x(xy)^\omega = (xy)^\omega \), where \( \omega \) denotes the idempotent power of the monoid: this implies in particular that the monoid is aperiodic. Now, a monoid \( M \) is in \( DO \) iff it satisfies the equation \( (xy)^\omega (yx)^\omega (xy)^\omega = (xy)^\omega \), and it is in \( DS \) iff it satisfies the equation \( ((xy)^\omega (yx)^\omega (xy)^\omega)^\omega = (xy)^\omega \). These classes were formally introduced in [28], and it follows from their formal definition that we have \( DA \subseteq DO \subseteq DS \). Further, we can test in PSPACE in \( S \) whether \( T(S) \) falls in one of these classes [31]. We can now state our result on semiautomatica without the counter-free assumption:

Theorem 6.1. Let \( S \) be a semiautomatica. If the transition monoid of \( S \) is in \( DO \), then the multi-letter CSh-problem for \( S \) is NL. If the transition monoid of \( S \) is not in \( DS \), then it is NP-complete.
For the CSh-problem, this result is a generalization of Theorem 5.6. Indeed, the monoids in DO which are aperiodic are exactly the class DA (see [28] and [2], Chapter 8), and the same holds for DS. Hence, for counter-free S, the above result for CSh collapses to Theorem 5.6. However, in general, DO covers more languages than DA. In particular, DO contains all group languages, i.e., the regular languages recognized by an automaton whose transition monoid is a (finite) group.

The main technical challenge to prove Theorem 6.1 will be to extend Theorem 4.3 to such languages, by showing the tractability of district group monomials. A district group monomial is a language of the form \(K_1a_1 \cdots K_na_nK_{n+1}\) where, for all \(i\), we have \(a_i \in A\) and \(K_i\) is a group language over an alphabet \(A_i \subseteq A\). Note that district group monomials generalize monomials, because any \(A_i^*\) is trivially a group language over \(A_i\) (even though it is not a group language over \(A\)). Thus, district group monomials are more expressive than the group monomials defined in earlier work [36], where \(A_i = A\) for all \(i\). We can then show the following result, which is our main technical achievement, and generalizes Theorem 4.3 (for the CSh-problem):

\[\textbf{Theorem 6.2.} \text{Let } K \text{ be a district group monomial. Then } \text{CSh}(K) \text{ is in NL.}\]

Note that this theorem is useful even outside of the semiautomaton phrasing. Indeed, it implies that the CSh-problem is tractable for many non-commutative languages that we had not covered by our previous results, e.g., \((ab^*+a)^*(ba^*b+a)^*\), the language testing whether there is one \(c\) preceded by an even number of \(a\) and followed by an even number of \(b\).

\[\textbf{Proof of Theorem 6.1.} \text{We first explain how Theorem 6.1 follows from Theorem 6.2 before dealing with the much more difficult task of proving Theorem 6.2. The overall scheme is like in the previous section: show that monoids in DO can be reduced to tractable languages (specifically, to district group monomials), and show that monoids not in DS capture an intractable language. For the upper bound, we use the following result, which is the counterpart of Theorem 5.7 but for DO rather than DA:}\]

\[\textbf{Theorem 6.3 ([33], Theorem 1).} \text{Let } K \text{ be a regular language of } A^*. \text{ The following conditions are equivalent:}\]

- \(K\) is an union of unambiguous district group monomials, i.e., of district group monomials \(K = K_1a_1 \cdots K_na_nK_{n+1}\) such that every word \(u \in K\) has a unique decomposition \(u_1a_1 \cdots u_na_nu_{n+1}\) where \(u_i \in K_i\) for all \(1 \leq i \leq n+1\).
- There exists a monoid \(M\) in DO and a morphism \(\eta : A^* \rightarrow M\) 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 5.8 to the non-aperiodic case:

\[\textbf{Proposition 6.4 ([2], Exercise 8.1.6).} \text{A monoid } M \text{ is not in DS iff there exists a morphism } \theta : \{a,b\}^* \rightarrow M \text{ and } P \subseteq M \text{ such that } \theta^{-1}(P) \text{ is either } (ab)^* \text{ or } (ab+b)^*.\]

From these two results, we can prove Theorem 6.1 exactly like we proved Theorem 5.6 in the previous section, using Theorem 6.2 instead of Theorem 4.3. The hard work that remains is to prove Theorem 6.2.

\[\textbf{Proof of Theorem 6.2.} \text{To present a high-level view of the proof of Theorem 6.2, we focus on the case of group languages: the general proof deals with group district monomials, which adds}\]
some complexity but follows the same overall techniques. 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 $I$ (i.e., a topological sort of the associated $H$-DAG) which evaluates to $g$ according to the group operation.

Imagine first that all elements of the group $H$ occur a very large number of times as labels in $I$. Specifically, assume that every group element $H$ occurs in the label of a very large number of different strings of $I$. The key idea in this case is that we can pick many occurrences of each letter in different strings, and obtain an antichain $C$ of incomparable elements, which contains many occurrences of each element of $H$. 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 element, this should give us the freedom to create many different group elements. We cannot obtain all elements of $H$, because the number of occurrences of each group element, i.e., the Parikh image, is fixed by that of $C$. However, it turns out that this is the only constraint on what we can generate in this way. We formalize this intuition in the antichain lemma (Lemma 6.5): 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 $\text{PI}(I)$ of the instance, under our initial assumption.

The remaining challenge is to lift our assumption: what happens if some group elements only occur in a small number of different strings in the input instance? In this case, intuitively, we can handle this small number of strings with an approach based on dynamic programming, as in the proof of Proposition 2.3 (or as we already did to prove Proposition 4.4). Formally, given the CSh-instance $I$, we will split the group elements between rare and frequent elements, which we call a rare–frequent partition. This will ensure that the rare elements $H_{rare}$ only occur in constantly many input strings (called the rare strings), and the frequent elements $H_{freq}$ occur in sufficiently many different input strings (called the frequent strings).

At this point, the problem looks solved: apply dynamic programming to the rare strings, and use the antichain lemma to argue that the frequent strings can generate any element of the subgroup spanned by $H_{freq}$, up to the commutative constraints. However, a problem 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 $H_{rare}$ and $H_{freq}$. We address this by proving a result called the insertion lemma (Lemma 6.6), 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 appendix). We then formalize the rare–frequent partition and sketch the remainder of the proof of Theorem 6.2.

**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 6.5 (Antichain lemma).** Let $H$ be a finite group and $\mu : A^* \rightarrow H$ 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 $\Pi(G) \in \Pi(\mu^{-1}(g_1 \cdots g_k))$ then there is a topological sort $\sigma$ 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.

**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 [9] written $\tau : N \rightarrow \mathcal{P}(H)$. 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 $\tau$ 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 6.6 (Insertion lemma).** Let $H$ be a finite group and $\mu : A^* \rightarrow H$ 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 $u = w_0'w_1w_1'w_2w_2' \cdots w_nw_n'$, 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 $v = w_0''w_1w_1'' \cdots w_nw_n''$, we have $\mu(u) = \mu(v)$ and $\mu(w_0'' \cdots w_n'') = \mu(w_0' \cdots w_n')$.

**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 $w_j'$). 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 show Theorem 6.2. Let $K$ be a group language on the alphabet $A = \{a_1, \ldots, a_k\}$. We let $\mu : A^* \rightarrow H$ 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 6.6 and, using Lemma 6.5 for the value $k := B$, let $R$ be the value of $n_k$ given by this lemma.

A rare–frequent partition of $I$ consists of a partition of $A$ into rare letters $A_{\text{rare}}$ and frequent letters $A_{\text{freq}}$, and a partition of the strings into rare strings $S_{\text{rare}}$ and frequent strings $S_{\text{freq}}$, where all vertices of $S_{\text{freq}}$ are labeled with letters of $A_{\text{freq}}$, and where $S_{\text{freq}}$, when seen as an instance of $I$ over the alphabet $A_{\text{freq}}$, 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 6.7.** 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 $A_{\text{freq}} \uplus A_{\text{rare}}$ of $A$ and the set of rare strings $S_{\text{rare}}$, such that $|S_{\text{rare}}| \leq R \cdot k^2$.

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

**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 $\mu$, 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 2.3 on the rare strings $S_{\text{rare}}$, with some insertions of a constant number of elements from $H_{\text{freq}}$ 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 6.5) to argue that any such pattern of insertions can indeed be performed using $S_{\text{freq}}$, thanks to the rich antichain that it contains. To show completeness, we rely on the insertion lemma (Lemma 6.6) 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.

**Limitations.** We close the section with comments on the two main limitations of Theorem 6.1. The first limitation is that it is not a dichotomy result: it does not cover the semiautomata whose transition monoid is in $DS \setminus DO$. One example of a language recognized by such a semiautomaton is $(a^+b^+a^+b^+)^*$, the language of words having an even number of subfactors of the form $a^+b^+$. We can show tractability for this language, which implies there are tractable semiautomata in $DS \setminus DO$ (we do not know of intractable examples):

**Proposition 6.8.** Let $K = (a^+b^+a^+b^+)^*$. Then CSh($K$) is in NL.

To extend our proof to $DS$ rather than $DO$, we would need an analogue of Theorem 6.3 but for the class $DS$. However, we are unaware of such a result, and it is unlikely that we can find one, because many fundamental questions about the class $DS$ are still open in algebraic automata theory. For instance, the problem of characterizing the languages that have a syntactic monoid in $DS$ has been open for almost 25 years [2, Open problem 14, page 442].

The second limitation of Theorem 6.1 inherited from Theorem 6.2 is that it only applies to the CSh-problem, although we would conjecture that it also holds for the CTS-problem. The part of the proof that cannot be generalized to the CTS-problem is the rare–frequent partition technique. Indeed, even for $A = \{a, b\}$, in an $A$-DAG $G$, there may be a large antichain $C_a$ of $a$-labeled vertices, and a large antichain $C_b$ of $b$-labeled vertices, and yet no rich antichain. This
is the case, for instance, when $G$ is the series composition of $C_a$ and $C_b$. This prevents us from using the antichain lemma, and yet we do not have a constant-size chain partition either. It seems that we would need an analogue of Dilworth’s theorem but for labeled DAGs, giving a structural decomposition of the labeled DAGs that do not have rich antichains; yet, we have been unable to formalize the intuition (see also [3]).

7 Conclusion and Open Problems

Summary of results. We have studied the complexity of two problems, constrained topological sort (CTS) and constrained shuffle (CSh): having fixed a regular language $K$, we are given a labeled DAG (for CTS) or a tuple of strings (for CSh), and we ask if the input has a topological sort satisfying $K$. We have shown tractability and intractability for several regular languages using a variety of techniques, and hinted at the complexity border between regular languages under suitable closure assumptions, by proving a dichotomy theorem (Theorem 5.6) in the case of counter-free semiautomata, and extending it to show the tractability of district group monomials for the CSh-problem.

Open problems. Our work leaves many research directions open. Under the semiautomata problem phrasing, which makes the problems tamer by enforcing some closure properties, we are left with two open problems in the non-counter-free case of Section 6. First, showing tractability for the DS class rather than DO, but this would probably require a better understanding of DS. Second, extending tractability for district group monomials from the CSh-problem to the CTS-problem, but this would seem to require a new structural understanding of DAGs without rich antichains, which we believe to be an intriguing direction for further research.

To move from the semiautomaton phrasing to the more natural regular language phrasing, we would need to remove closure assumptions. This makes the problem trickier to understand, because we have shown that there are languages which are tractable but whose closure (by complement, intersection, etc.) is not. Maybe the easiest closure operation to remove would be Boolean complementation, which corresponds in the algebraic world to classes of ordered monoids and ordered semiautomata [25]. Removing closure under quotient (for the CSh problem) may also be possible [15]. Altogether, it looks surprisingly tricky to prove our complete dichotomy conjecture (Conjecture 2.4), and to characterize the tractable languages for the CTS-problem and CSh-problem.

We also stated many smaller open questions throughout the paper, asking about the complexity of specific languages. We do not know the complexity of $\text{CTS}((aa + b)^*)$ (i.e., generalizing Proposition 4.5 from CSh to CTS) or the complexity of either problem for any language of the form $(a^i + b)^*$ with $i > 2$. We do not know how far our reduction technique can be applied (see Conjecture 3.9), and we do not correctly understand the complexity of CTS for languages that are tractable for CSh thanks to partial width restrictions, e.g., $(ab)^* + A^*a^iA^*$ for $i > 2$ (see Proposition 4.4).

Another natural question would be to investigate the status of non-regular languages. The simplest example is the Dyck language, which appears to be NP-hard for CTS (at least in the multiletter case), but tractable for CSh, via a connection to scheduling; see [13], problem SS7. More generally, however, all our problems extend to more general language classes, e.g., context-free languages, for which the complexity landscape may be equally enigmatic.
A Proofs for Section 2 (Problem Statement and Main Results)

Proposition 2.2. For any language $K$, the problems $CTS(K)$ and $CSh(K)$ are in $NP^K$.

Proof. As explained in the main text, we guess a permutation $\sigma$ 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 $\sigma$ is in $K$. \hfill \blacksquare

In the rest of this appendix, we state and prove Proposition 2.3.

Proposition 2.3. For any 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 first define formally the notion of chain partition. A chain partition of a DAG $G = (V, E)$ 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 a line graph (also called a chain). Note that, in addition to the edges of the chains, 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 A.1 [8]. 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 [12] 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. 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 $\chi$ 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 $\sigma$, equivalently seen as a total order $<_{\sigma}$, which is minimal according to the lexicographic order defined by $<$: namely, $\sigma$ 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 $\chi_0$ 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 $\chi_0$. 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 \textit{next vertex} of the chain of \( v \) in \( \chi_0 \), and one function first, which takes as input a chain number \( 1 \leq i \leq k \) and returns the \textit{first vertex} of \( i \) in \( \chi_0 \). Formally, 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 \( \chi_0 \), return the vertex next(\( v \)) \( \in V \) which is the successor of \( v \) on chain \( c \) in \( \chi_0 \), if any, or \( \bot \) if \( v \) is the last vertex of chain \( c \). More formally, next(\( v \)) is the vertex of \( V \) such that \( \chi_0(\text{next}(v)) = c \), the edge (\( v, \text{next}(v) \)) is in \( E \), and there is no \( z \in V \) such that \( \chi_0(z) = c \) and \( v \leadsto z \leadsto \text{next}(v) \) in \( G \). As for the function first, for any chain number \( 1 \leq c \leq k \), we let first(\( c \)) be the first element of the chain \( c \) in \( \chi_0 \), that is, we have \( \chi_0(\text{first}(c)) = c \) and there is no \( z \in V \) such that \( \chi_0(z) = c \) and \( z \leadsto \text{first}(c) \). We can now claim:

\textbf{Lemma A.2.} 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 this sequel: prove this lemma, and use it to prove Proposition 2.3.

To do this, we need to define the notion of a \textit{configuration}, which will be useful in our algorithms on chain partitions. A \textit{configuration} is a \( k \)-tuple \( X = (v_1, \ldots, v_k) \), where each \( v_i \) is either an element of \( V \) or \( \bot \). Intuitively, \( X \) describes the lowest element of each chain, with \( \bot \) 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 \textit{ancestors} of \( X \), meaning all vertices \( w \) such that \( w \leadsto v_i \) for some \( v_i \), have already been assigned to a chain in some fashion. We say that a configuration \( X \) is \textit{continuable} if there exists a chain partition \( \chi \) which is consistent with \( X \), meaning that \( \chi(v_i) = i \) for all \( 1 \leq i \leq k \) such that \( v_i \neq \bot \). One useful lemma will be the following:

\textbf{Lemma A.3.} 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 A.2. We will then explain how to prove Lemma A.3. Last, we will prove Proposition 2.3 from Lemma A.2.

We start by proving Lemma A.2. The intuition is that we can use the continuation check of Lemma A.3 as a way to compute implicitly the minimal chain partition, by considering all vertices in the minimal topological sort \( \prec_{\sigma} \), and assigning each vertex to the smallest possible chain such that the resulting configuration is continuable. Formally, we show:

\textbf{Proof of Lemma A.2.} We maintain a configuration \( X = (v_1, \ldots, v_k) \), initially \((\bot, \ldots, \bot)\), and extend it \textit{deterministically} at each step using the (nondeterministic) oracle for continuation checking described in Lemma A.3. 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 \( \prec \) 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 (\( v_i, v \)) is in \( G \) or \( v_i = \bot \), 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-Szelepesényi theorem \cite{18, 30}. 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 \( \prec_{\sigma} \) 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 $(\bot, \ldots, \bot)$, and this configuration is continuable, as we know by Dilworth’s theorem (Theorem A.1). 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 $X = (v_1, \ldots, v_k)$ is continuable, letting $\chi$ be a witnessing chain partition, letting $v$ be the next vertex that we consider, we know that, if $v_{\chi(v)} = \bot$, then we can take $i = \chi(v)$. If $v_{\chi(v)} \neq \bot$, 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 $(v_{\chi(v)}, v)$ must exist. Hence, $\chi$ 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 $\chi_0$, in the sense that, at each step of the algorithm, the current configuration is consistent with $\chi_0$, 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 $(\bot, \ldots, \bot)$ is of course consistent by $\chi_0$. Now, assuming consistency of the configuration, as $\chi_0$ 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 $\chi_0$. So we have shown that our algorithm actually computes next and first following $\chi_0$, proving the result.

We now come back to the proof of Lemma A.3

**Proof of Lemma A.3.** The proof follows similar ideas as in Lemma A.2: 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 A.2 (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 A.2 we now know that we can implicitly compute the minimal chain partition within the prescribed time bounds. We are now ready to prove Proposition 2.3

**Proof of Proposition 2.3.** We fix an automaton $A$ for the regular language $K$: remember that, as $K$ is fixed, we can compute $A$ 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 $X = (v_1, \ldots, v_k)$ such that each $v_i$ is either $\bot$ or an element of $V$, which intuitively codes the lowest element for each chain, or $\bot$ if no element of the chain has been seen so
far: initially the configuration is \((\perp, \ldots, \perp)\). 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 \(v'_i := \text{first}(v_i)\); if \(v_i \neq \perp\), then we replace \(v_i\) in \(X_i\) by \(v'_i := \text{next}(v_i)\) 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 \((q, \lambda(v'_i), q')\), 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 \(v_i \neq \perp\) and 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 A.2. Conversely, whenever there is a witnessing topological sort, then we can decompose it along the minimal chain partition \(\chi_0\) 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.

\section{Proofs for Section 3 (Hardness Results)}

\subsection{Proofs for the Shuffle Reduction}

In this appendix, we prove a generalization of Theorem 3.4. First note that Definition 3.2 and 3.3 extend from regular languages to arbitrary languages. We say that a family \(\mathcal{K}\) of languages has a \textbf{logspace commutative word problem} if we can decide in logspace, given a language \(K\) of \(\mathcal{K}\) and a word \(w\), whether \(w \in \text{CCl}(K)\): this is in particular the case of all regular languages.

We can now state and prove a generalization of Theorem 3.4 which applies to all languages with a logspace commutative word problem (not just regular languages):

\begin{theorem}
For any family of languages \(\mathcal{K}\) and language \(K'\), if \(\mathcal{K} \leq_{\text{shuf}} K'\) and \(\mathcal{K}\) has a logspace commutative word problem, then there is a logspace reduction from \(\text{CTS}(\mathcal{K})\) to \(\text{CTS}(K')\), and from \(\text{CSh}(\mathcal{K})\) to \(\text{CSh}(K')\).
\end{theorem}

\begin{proof}
We show the result for the CSh-problem; the result for the CTS-problem is shown in exactly the same fashion. 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 \((f_n)\) be the filter sequence for \(K\) and \(K'\). If \(\text{PI}(I) \neq \text{PI}(K)\), then clearly \(I\) is not a positive instance of the CSh-problem for \(K\), so it suffices to reduce to some negative instance of the CSh-problem for \(K'\). Specifically, letting \(n := |I|\), we can compute \(f_n\), choose any topological sort \(v\) of \(I\), and we have \(|v| = n\) and \(v \notin K\),
so that \( v \cup f_n \) is disjoint from \( K' \). Picking any element \( w \) from this set and constructing the corresponding string yields a suitable negative instance for \( K' \) (that has only one topological sort, namely, \( w \)), and this reduction is computable by a logspace transducer by our computability hypothesis on \( (f_n) \). Hence, we assume in what follows that \( \text{PI}(I) \in \text{PI}(K) \).

Now, letting \( n := |I| \), let us call \( I' \) the instance of the CSh-problem for \( K' \) that contains \( I \) and a separate string labeled with \( f_n \): by our computability hypothesis on \( (f_n) \), this is computable by a logspace transducer. We now argue that \( I' \) is a positive instance to the CSh-problem for \( K' \) iff \( K \) is a positive instance to the CSh-problem for \( K \). Indeed, assuming that there is a topological sort \( v \) of \( I \) in \( K \), we have \( |v| = n \), and \( \text{PI}(v) \in \text{CCl}(K) \) by definition, so by definition of \( (f_n) \) we have \( v \cup f_n \cap K' \neq \emptyset \). Hence, let \( v' \) be an element of this set. It is in \( v \cup f_n \), so it can be obtained as a topological sort of \( I' \) by shuffling \( f_n \) with the topological sort \( v \) of \( I \), 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 \( v' \) of \( I' \) in \( K' \), it defines a topological sort \( v \) of \( I \), such that \( v' \in v \cup f_n \). By our hypothesis on \( I \), we know that \( \text{PI}(I) \in \text{PI}(K) \), which ensures that \( v \in \text{CCl}(K) \), and clearly \( |v| = n \). Thus, as \( v' \) witnesses that \( v \cup f_n \cap K' \) is non-empty, we must have \( v \in K \), so that \( v \) witnesses that \( I \) is a positive instance to the CSh-problem for \( K \). This establishes correctness, and concludes the proof.

### B.2 Hardness Proofs for Other Languages

**Proposition 3.6.** Let \( u \in A^* \) such that \( |u|_a > 0 \) and \( |u|_b > 0 \) for \( a \neq b \) in \( A \). Then CSh\((u^*)\) (hence CTS\((u^*)\)) is NP-hard.

**Proof.** Fix \( u \in A^* \) and the two witnessing letters \( a \) and \( b \). We first make a straightforward 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 \( (ab)^* \) 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, by definition of \( (ab)^* \), clearly it suffices to define \( f_{2n} \) for \( n \in \mathbb{N} \), which we define as \( f_{2n} := x(u'u'_{-a}u'u'_{-b}u^n yy) \); this is clearly logspace-computable. We show that this is a filter sequence by picking \( n \in \mathbb{N} \) and letting \( v \) be a word such that \( |v|_a = |v|_b = 2n \).

If \( v = (ab)^n \), we can clearly interleave \( v \) and \( f_{2n} \) to obtain a word of \( u^* \) by interleaving 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, by considering a candidate interleaving and assuming by contradiction that we do not do these insertions, e.g., in \( u'_{-a} \), then consider the subword formed of one \( u'_{-a} \) where we did not insert any \( a \) in or around it, and the neighboring letter from the beginning or end of \( u' \), taking one such letter which is not \( a \) (which is always possible by hypothesis on \( u' \)). The number of \( a \)'s in this is one less than in \( u \), but the size of this is at least \( |u| \) (or more if some insertions of other symbols than \( a \) were made), which is impossible by our preliminary observation. So we must insert one \( a \) in or around each \( u'_{-a} \), and likewise for the \( u'_{-b} \); these insertions are distinct, and they use up all letters of \( u \), so for \( f_{2n} \uplus v \) to intersect
\(u^*\) nontrivially, we must have \(v = (ab)^n\). This shows that \((f_n)\) is indeed a filter sequence, and allows us to conclude by Theorem 3.4.

\[\blacktriangleright\textbf{Proposition 3.7.} \text{Let } L := (aa + bb)^*. \text{ The problem } \text{CSh}(L) \text{ (hence } \text{CTS}(L)) \text{ is } \text{NP-hard.} \]

\[\textbf{Proof.} \text{ We show a reduction from } K := (ab)^* \text{ to } K' := (aa + bb)^*, \text{ using the following filter sequence, which is clearly logspace-computable: } f_n := (ab)^n. \text{ We will show a slight strengthening of the definition of filter sequences, by additionally showing that for every } n \in \mathbb{N} \text{ and } v \in \text{CCI}(K) \text{ with } |v| < n, \text{ we have } v \sqcup f_n \cap K' = \emptyset. \text{ This allows us to conclude using Theorem 3.4.} \]

We show this strengthened claim by induction on \(n \in \mathbb{N}\); we focus on even values of \(n\) as the claim is vacuous when \(n\) is odd. The cases \(n = 0\) is immediate. We show the case of \(n = 2\) as it will be useful for the induction step. Let \(v \in \text{CCI}(K)\). We know that \(|v|\) is even, and it is clear that if \(|v| = 0\) then indeed \(v \sqcup f_2 \cap K' = \emptyset\) because \(f_2 \notin K'\). Assume now that \(|v| = 2\), i.e., \(v \in \{ab, ba\}\). For the forward direction, consider \(f_2 = ab\). If \(v \in K\), i.e., \(v = ab\), then there is an interleaving of \(v\) and \(f_2\) which is in \(K'\), namely, \(aabb\). For the converse direction, assume that there is an interleaving \(w\) of \(v\) and \(f_2\) which is in \(K'\). Given the definition of \(K'\), it is clear that we must have inserted in \(w\) an \(a\) from \(v\) either immediately before or immediately after the \(a\) from \(f_2\), and the same is true for the \(b\) from \(f_2\). Hence, \(v\) must contain an occurrence of \(a\) before an occurrence of \(b\), so we deduce that \(v = ab\). This concludes the case \(n = 2\).

For the induction, let us consider an even \(n \in \mathbb{N}\), assume that the claim holds for \(n\), and consider the word \(f_{n+2} = abf_n\). Let \(v\) be an arbitrary word of \(\text{CCI}(K)\) such that \(|v| \leq n + 2\), and show the claim. For the forward direction, if \(v = (ab)^{(n+2)/2}\), then we can clearly build the word \((aabb)^{(n+2)/2}\) as an interleaving of \(v\) and \(f_n\), which is in \(K'\). For the converse direction, we must informally show that there is no suitable interleaving if \(|v| < n + 2\), and that otherwise there is one only when \(v = (ab)^{(n+2)/2}\). Thus, let \(w\) be an interleaving of \(v\) and \(f_n\) which is in \(K'\). We can Let us write \(w = pw'\), where \(p\) contains the letters from the first \(ab\) factor of \(f_{n+2}\), is in \(K'\), and is as short as possible. This ensures that \(p\) ends immediately after the first \(b\) of \(f_{n+2}\), or after an additional \(b\) taken from \(v\) which cannot be the first \(a\) of \(f_n\). Hence, we can write \(w = pw'\) with \(p\) containing the first \(ab\) of \(f_{n+2}\) and no other letter from \(f_{n+2}\), and \(p, w' \in (aa + bb)^*\). Now, let us similarly split \(v = qv'\), where \(q\) contains the letter occurrences that are in \(p\), and \(v'\) contains those that are in \(w'\). Observe that \(p\) is an interleaving of \(q\) and \(f_2 = ab\), and that \(p \in (aa + bb)^*\). Thus, by an immediate strengthening of the case \(n = 2\), we know that \(q\) must contain one occurrence of \(a\) before one occurrence of \(b\). Hence, we have \(|v'| \leq n\). Observing further that \(w'\) is an interleaving of \(v'\) and \(f_n\) which is in \((aa + bb)^*\), we can now apply the induction hypothesis to deduce that, necessarily, \(|v'| = n\) and \(v' \in (ab)^*\). This implies that we must have \(q = ab\), \(|v| = n + 2\), and \(v \in (ab)^*\). This concludes the backwards induction of the claim and establishes the induction step, which concludes the proof.

\[\blacktriangleright\]

\section*{C Proofs for Section 4 (Tractability Results for the DAG Problem)}

\subsection*{C.1 Additional Explanations About \((ab)^* + A^*aaA^*\)}

We first substantiate a claim made in the main text, namely:

\[\blacktriangleright\textbf{Claim C.1.} \text{ The regular language } K = (ab)^* + (a + b)^*aa(a + b)^* \text{ cannot be expressed as a union of monomials.} \]

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 [26] 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:

For all \( x, y \in A^* \) having same content, \( x^\omega \geq x^\omega yx^\omega \) where “\( x \) and \( y \) having the same content” means that, for each letter \( a \in A \), we have \( |x|_a > 0 \) iff \( |y|_a > 0 \).

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 \( u, v \in A^* \), \( uv \in K \) implies \( u xv \in K \). 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 \( x^n \leq_K x^{2n} \), we have \( x^n \leq_K x^n yx^n \).

For our choice of language \( K \), we can show that this rephrased condition does not hold, by taking \( x := ab \) and \( y := abab \) 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 and \( x^1 yx^1 = abababab \) which is not in \( K \), so we have shown that \( x^n \not\leq_K x^n yx^n \). ▶

C.2 Proof of Theorem 4.3

Theorem 4.3. For any monomial language \( K = A_1^*a_1A_2^*a_2 \cdots A_n^*a_nA_{n+1}^* \), the CTS-problem for \( K \) is in NL.

Proof. First, we can guess in NL the elements \( v_1, \ldots, v_n \) 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 elements, 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 \( K = A_1^*a_1 \cdots A_n^*a_nA_{n+1}^* \), 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 \( A_1^* \) which is commutative.

For the induction step, let \( K = A_1^*a_1A_2^*a_2 \cdots a_{n+1}A_{n+2}^* \) and \( K' = A_1^*a_1A_2^* \cdots a_nA_{n+1}^* \). 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 elements 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 \( v_1, \ldots, v_n \),
- 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 \( w \).

We now claim the following:

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

Note that \( G' \) is always computable in NL, so, once this claim is proved, we have an NL algorithm for CTS(\( K \)) running the NL algorithm (given by 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 \( \sigma \) of \( G \) satisfying \( K \) by concatenating the topological sort \( \sigma' \) 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, we know that all ancestors of \( v_{n+1} \) are in \( G' \), and we know that \( v_{n+1} \) itself is not in \( G' \), because \( \sigma' \) witnesses that \( G' \) did not contain any vertex labeled by \( a_{n+1} \), thanks to the fresh pivot assumption. We now argue that \( \sigma \) achieves \( K \): this is because \( \sigma' \) achieves \( K' \), \( v_{n+1} \) achieves \( a_{n+1} \), and by assumption all remaining elements 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, \( \sigma \) is a topological sort of \( G \) that achieves \( K \), establishing the backward implication.

For the forward direction, consider a topological sort \( \sigma \) of \( G \) that achieves \( K \). Thanks to the fresh pivot assumption, we know that \( v_{n+1} \) is matched to \( a_{n+1} \). Let \( U \) be the elements enumerated before \( v_{n+1} \) in \( \sigma \), and let \( \sigma' \) be the topological sort induced by \( \sigma \) on them: we know that \( \sigma' \) satisfies \( K' \). We now claim that \( V' \subseteq U \). Indeed, first, by the fresh pivot hypothesis, \( \sigma' \) 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 \( \sigma \) enumerates \( v_{n+1} \) just after \( \sigma' \), we know that \( \sigma' \) 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 \( \sigma' \), 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 \( \sigma'' \) of \( V' \) defined as the restriction of \( \sigma' \) 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 \( \sigma'' \) witnesses that \( G' \) is a positive instance to \( K' \). Now, as \( \sigma \) 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

- **Proposition 4.5.** Let \( A := \{a, b\} \) and \( K := (aa + b)^* \). The problem \( \text{CSh}(K) \) 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 we assume this in the sequel.

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 we can always remove any input string that consists only of \( b \)'s. Now, if there are less than 3 input strings, then we can conclude in NL by Proposition 2.3 so we assume that there are at least 3 strings in the input instance which contain some \( a \)'s.

Given an input instance \( I \) to the CSh-problem for \( K \), we call a block in a string 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 any string $C$, the $a$-alternation of $C$ is no greater than the total $a$-weight of the other strings (which we call assumption (i)), 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 decide to remove any two contiguous $a$’s. 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: that any input instance satisfying (*) with an even number of $a$’s and with at least 3 non-empty strings can be rewritten through simplifications to an instance satisfying (**), and that given an instance satisfying (**) 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 that, 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 is disallowed by assumption (**). 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 size of each of these two strings and the total size 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 $C''$ 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 size, and it is obvious that if condition (***) holds of a string \( C \) then it holds of a string with the exact same size (as the size of the two strings is the same, and so is the size 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 size 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 in this case, 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 that, 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 size. 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 size \( n \), 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 than \( C \) was to bring it down to \( n \) or \( n + 1 \), in which case it witnesses 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 Dichotomy Theorem)

### D.1 Proofs of Closure Counterexamples

**Proposition 5.1.** There exists a word \( u \in A^* \) and a regular language \( K \) such that \( \text{CSh}(K) \) is tractable but \( u^{-1}K = (ab)^* \), so that \( \text{CSh}(u^{-1}K) \) is NP-hard by Theorem 3.5.

**Proof.** Take \( A := \{a, b\} \) and \( K := b^*A^* + aaA^* + (ab)^* \). Take \( u := ab \). It is clear that \( u^{-1}K = (ab)^* \). However, \( \text{CSh}(K) \) is tractable by the following reasoning. Consider an input instance to \( \text{CSh}(K) \). 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 \( \text{NL} \) whether the only possible topological sort achieves \( K \).

In particular, for this choice of \( K \), we know that \( \text{CSh}(K) \) is in \( \text{NL} \) but \( \text{CTS}(K) \) is NP-hard, as will follow from Proposition 5.2.


Proposition 5.2. For any word \( u \in A^* \) and regular language \( K \), there is an \( \log \)-space reduction from \( CTS(u^{-1}K) \) to \( CTS(K) \).

Proof. Fix \( u \in A^* \) and \( K \). Given an \( A \)-DAG \( G \), to solve \( CTS(u^{-1}K) \) on \( G \), construct the DAG \( G' \) obtained by adding a chain of elements whose label is \( u \) and setting it as an ancestor of 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.

Proposition 5.3. There exists two regular languages \( K_1 \) and \( K_2 \) such that \( CTS(K_1) \) and \( CTS(K_2) \) are both in PTIME, but \( K_1 \cap K_2 = (ab)^* \), so that \( CSh(K_1 \cap K_2) \) is NP-hard by Theorem 3.5.

Proof. 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 \( CTS(K_1) \) and \( CTS(K_2) \) 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 \( CTS(K_1) \) is in PTIME, then by Proposition 5.2 as PTIME-membership is clearly preserved by renaming the symbols, we have also shown that \( CTS(K_2) \) is in PTIME. So we focus on \( K_1 \).

We will show a greedy algorithm in PTIME to solve \( CTS(K_1) \), and explain why it succeeds. The algorithm has two states:

- **State a** (the initial state), where:
  - being out of symbols means that we have succeeded, i.e., we have constructed a topological sort in \( (ab)^* \);
  - enumerating a \( a \) allows us to move to state \( b \);
  - enumerating a \( b \) allows us to “win”, i.e., that we can continue the topological sort in any way and remain in the language.

- **State b**, 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 the target language;
  - enumerating an \( a \) is not possible;
  - enumerating a \( b \) allows us to move back to state \( a \).

We accordingly design the algorithm as follows:

- In state \( b \):
  - if there is an available \( b \), enumerate any of them and move to state \( a \);
  - otherwise fail.

- In state \( a \):
  - 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 \( b \);
  - 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
and CTS-problem for morphism $\theta$.

We first give some detail about the proof of Proposition 5.8:

We now modify which concludes the correctness proof.

which case there is no constraint on $v$.

D.2 Proof of the Dichotomy Theorem (Theorem 5.6)

Theorem 5.6. 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.

We first give some detail about the proof of Proposition 5.8

Proposition 5.8 ([31], Lemma 10). An aperiodic monoid $M$ is not in $DA$ iff there exists a morphism $\theta : \{a,b\}^* \rightarrow M$ and $P \subseteq M$ such that $\theta^{-1}(P)$ is either $(ab)^*$ or $(ab+b)^*$.

Proof. This result follows from [31], 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 [31]). 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 [29, Theorem V.1.3].

We can now prove Theorem 5.6.

**Proof.** Fix the input semiautomaton $S$. We wish to show that the multi-letter CTS-problem is tractable for $S$ iff the transition semigroup $T(S)$ of $S$ is in $DA$. We call $SL(K)$ the set of possible languages that can be defined from $S$ depending on the input instance, namely, depending on the set $\{(i_0, F_0), \ldots, (i_k, F_k)\}$ 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 $SL(S)$, 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 $SL(S)$ 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 over words over the alphabet $M$ defined by $\psi(m) := m$ for all $m \in M$. Intuitively, applying $\psi$ to a sequence of elements of $M$ simply evaluates the sequence in $M$.

Let $I = (G, (i_0, F_0), \ldots, (i_k, F_k))$ be an instance of the semiautomaton CTS-problem and let $K_j$ be the language recognized by the automaton $(Q, A, \delta, i_j, F_j)$ for all $0 \leq j \leq k$. We must determine whether $G = (V, E, \lambda)$ has a topological sort in $K := \cap_j K_j$. 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 $K' := \{u \in M^* | \psi(u) = \eta(K)\}$. In other words, $K' = \psi^{-1}(\eta(K))$, so $K'$ is recognized by $M$ which is a monoid in $DA$: by Theorem 5.7, we know that $K'$ is a union of monomials.

Our goal is then to reduce to $CTS(K')$. 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 Lemma 4.1, we know that $CTS(K')$ is in NL.

To show the equivalence, we will show that for any topological sort $\sigma$ of $(G, V)$, the word $\lambda(\sigma)$ achieved by $\sigma$ in $G$ is in $K$ iff the word $\lambda'(\sigma)$ achieved by $\sigma$ 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 $\eta$. 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 $\eta$ is a morphism, so applying $\eta^{-1}$ 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 5.8: there exists a morphism $\theta : \{a, b\}^* \to M$, a set $P \subseteq M$, and a regular language $H \in \{(ab)^*, (ab + b)^*(\epsilon + a)\}$ such that $\theta^{-1}(P) = H$. Our goal is to use $\theta$ 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 \( I_f = (G', (q_0, \{f(q_0)\}), \ldots, (q_n, \{f(q_n)\})) \) where \((q_i)_{i=0,\ldots,n}\) 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 \( I_f \) if and only if \( f_z = f \) in \( M \). This construction is in NL. Let us show that \( G \) is a positive instance to \( \text{CSh}(H) \) if and only if 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 \( \sigma \) 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 \( \theta \) 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 \( \sigma \) 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 \( \sigma \) of \((V, E)\) such that the word \( z := \lambda'(\sigma) \) achieved by \( \sigma \) in \( G' \) is such that \( f_z = f \). Now, we know that \( \theta^{-1}(f) \subseteq H \). Hence, the word \( \lambda(\sigma) \) achieved by \( \sigma \) in \( G \) is in \( H \), so \( G \) is a positive instance to \( \text{CSh}(H) \), which establishes the desired equivalence.

We have thus shown a reduction from \( \text{CSh}(H) \) to the CSh-problem to the semiautomaton CSh-problem of \( S \). We can then conclude that the latter problem is NP-hard, because \( \text{CSh}(H) \) is NP-hard: either \( H = (ab)^* \) and this follows from Theorem 3.5 or \( H = (ab + b)^*(e + a) \), in which case we conclude from Proposition 3.8.

### E Proofs for Section 6 (Lifting the Counter-Free Restriction for the CSh-Problem)

This appendix (except Section 6.5) consists of the full proof of Theorem 6.2. We split it in several subsections.

#### E.1 Proof of the Antichain Lemma (Lemma 6.5)

**Lemma 6.5 (Antichain lemma).** Let \( H \) be a finite group and \( \mu : A^* \rightarrow H \) be a surjective morphism. For any integer \( k > 0 \), there exists an integer \( n_k \) such that, for any \( A\text{-DAG} G = (V, E, \lambda) \) with an \( n_k \)-rich antichain, for any elements \( g_1, \ldots, g_k \) of \( H \), if \( \text{PI}(G) \in \text{PI}(\mu^{-1}(g_1 \cdots g_k)) \) then there is a topological sort \( \sigma \) of \( G \) decomposable as \( \sigma = \sigma_1 \cdots \sigma_k \) such that \( \mu(\lambda(\sigma_i)) = g_i \) for each \( i \in \{1, \ldots, k\} \).

To prove the antichain lemma, let us fix the finite group \( H \) and morphism \( \mu \). 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 automata theory, we will say that a language \( K \) is recognized by the morphism \( \mu \) 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.1 ([16], 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 [10] for a counterexample). Let us accordingly define a finite monoid \( N \), and let \( \text{Com}_\mu : A^* \to N \) be a surjective morphism such that, for each \( g \in H \), the morphism \( \text{Com}_\mu \) recognizes \( \text{CCl}(\mu^{-1}(g)) \). 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.1. Further, thanks to commutativity, we can choose \( N \) to be a finite commutative monoid. Let \( \omega \) be a positive idempotent power of \( N \), that is, a value \( \omega \in \mathbb{N} \setminus \{0\} \) such that we have \( p^{\omega} = p^{2\omega} \) 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 \( \omega \) 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 \([9]\) 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 \( \tau : M \transient M' \) if \( \tau \) 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:

\[ \text{Lemma E.2.} \] Let \( \tau : N \transient H \) 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.

\[ \text{Proof.} \] We will show the result using the following claim (*): for any fully recurrent element \( r \), we have \( |\tau(r)| = |\tau(r')| \) 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 \( p^\omega = (s_1^{r_1})^{\omega} \cdots (s_n^{r_n})^{\omega} = s_1^{r_1 \omega} \cdots s_n^{r_n \omega} \), and similarly for \( q \). This allows us to conclude from (*) because we have \( |\tau(p)| = |\tau(q)| = |\tau(p^\omega)| = |\tau(q^\omega)| = |\tau(q)| \).

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 arbitrarily 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 \( |\tau(r^i)| \leq |\tau(r^i) \cdot \{ x \}| \leq |\tau(r^i) \tau(r)| \leq |\tau(r^{i+1})| \). This shows part (i).

We now show part (ii). To do so, let us show first that \( r^{i+1} = r \). Indeed, write \( r = s_1^{r_1} \cdots s_n^{r_n} \), and we simply conclude using the fact that \( s_i^{r_1+\omega} = s_i^{r_i-\omega}(s_i^{\omega})^2 = s_i^{r_i-\omega}s_i^{\omega} = s_i^{r_i} \). This implies that we have \( r^{j+1} = (r^\omega)^jr = r^{\omega}r = r \), 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 6.5):

**Proof.** 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 $\mu$ is surjective, and is finite because $H$ is finite. Let $\text{Com}_\mu : A^* \to N$ be the surjective morphism defined as above, where $N$ is a commutative monoid, and let $\omega$ be the idempotent power of $N$. Finally, let $\tau : N \Rightarrow H$ be the relational morphism defined by $\tau(x) = \mu(\text{Com}_\mu^{-1}(x))$. 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 $\text{Com}_\mu(G) \subseteq \text{Com}_\mu(\mu^{-1}(g_1 \cdots g_k))$. Indeed, the forward implication is immediate, and the converse holds because $\text{Com}_\mu$ recognizes $\text{CCL}(\mu^{-1}(g_1 \cdots g_k))$, so the rephrased condition implies that $\text{CCL}(G) \subseteq \text{CCL}(\mu^{-1}(g_1 \cdots g_k))$, which clearly implies the original condition. Further, by composing with $\tau$ and simplifying using the definition of $\tau$, the condition rephrases to $g_1 \cdots g_k \in \tau(\text{Com}_\mu(G))$. 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, $\text{PI}(G)$, we know from the commutativity of $N$ that all topological sorts of $G$ have the same image by $\text{Com}_\mu$, namely, $\text{Com}_\mu(G)$. Hence, $T$ is included in $\tau(\text{Com}_\mu(G))$. Our goal is to show that, when $G$ has a $\omega$-rich antichain, we have $T = \tau(\text{Com}_\mu(G))$. Indeed, in this case, we know that, for any $g_1$ such that $\text{PI}(G) \in \text{PI}(\mu^{-1}(g_1))$, we have $g_1 \in \tau(\text{Com}_\mu(G))$ 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 $T = \tau(\text{Com}_\mu(G))$.

Let $C$ be a $\omega$-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 $\omega$-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 $\sigma^-$ and $\sigma^+$ 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 $\sigma$ is a topological sort of $C$. Hence, $T' := \mu(\sigma^-) \cdot \tau(\text{Com}_\mu(C)) \cdot \mu(\sigma^+) \subseteq T$, so $|T'| \leq |T|$. Let us now write $s_\alpha := \text{Com}_\mu(a)$ for each letter $\alpha \in A$. We can write $\text{Com}_\mu(C) = \Pi_{\alpha \in A} s_\alpha^{i_\alpha}$, where $i_\alpha$ is the number of vertices labeled by $\alpha$ in $C$. As $C$ is $\omega$-rich, we have $i_\alpha \geq \omega$. Thus, $\text{Com}_\mu(C)$ is fully recurrent by definition. Now, it is clear that $\text{Com}_\mu(G)$ is also fully recurrent, because $G$ is $\omega$-rich also. Thus, by Lemma 6.2, we have $|\tau(\text{Com}_\mu(C))| = |\tau(\text{Com}_\mu(G))|$. Now, we know that $\mu(\sigma^-)$ (resp. $\mu(\sigma^+)$) act bijectively on the left (resp. right) of $H$, so we also have $|\tau(\text{Com}_\mu(C))| = |T'|$. We have thus shown that $|\tau(\text{Com}_\mu(G))| = |T'| \leq |T|$. As $T \subseteq \tau(\text{Com}_\mu(G))$, we deduce that $T = \tau(\text{Com}_\mu(G))$. 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 $\eta_k+1$-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 $\sigma^-$ and $\sigma^+$ be
arbitrary topological sorts of $G^-$ and $G^+$ respectively. Let us choose elements $g_1, \ldots, g_{k+1}$ of $H$ such that $g_1 \cdots g_{k+1} \in \tau(\text{Com}_\mu(G))$: remember that this implies that $g_1 \cdots g_{k+1} \in \tau(\text{Com}_\mu(G))$.

Now, let us consider $g' := g_{k+1} \cdot \mu(\sigma^+)^{-1}$. Let $u_{g'} \in A^*$ be a word that realizes the minimum in the definition of $\gamma$, 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 $n_{k+1}$-rich antichain. Further, the definition of $C_{g'}$ ensures that it has a topological sort $\sigma'$ that realizes the word $u_{g'}$, so that $\mu(\sigma') = g'$. By composing it with $\sigma^+$, 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 an $n_k$-rich antichain. To do so, we must check that $\text{PI}(G') \in \text{PI}(\mu^{-1}(g_1 \cdots g_k))$, which as we argued is equivalent to $g_1 \cdots g_k \in \tau(\text{Com}_\mu(G'))$. As $G$ is the disjoint union of $G'$ and $G''$, we have $\text{Com}_\mu(G) = \text{Com}_\mu(G') \text{Com}_\mu(G'')$, so by composing by $\tau$ and applying the definition of a relational morphism we have:

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

Now, as both $G$ and $G'$ contain an antichain which is at least $\omega$-rich, we know that $\text{Com}_\mu(G)$ and $\text{Com}_\mu(G')$ are fully recurrent. By applying Lemma E.2, again we know that $|\tau(\text{Com}_\mu(G))| = |\tau(\text{Com}_\mu(G'))|$. Remember now that $\sigma' \sigma^+$ is a topological sort of $G''$ such that $\mu(\lambda(\sigma' \sigma^+)) = g_{k+1}$. Hence, $g_{k+1} \in \tau(\text{Com}_\mu(G''))$. Now, as $g_{k+1}$ acts bijectively on $\tau(\text{Com}_\mu(G'))$ in the group $H$, we deduce that $\tau(\text{Com}_\mu(G)) = \tau(\text{Com}_\mu(G'))g_{k+1}$. Now, since we have $g_1 \cdots g_{k+1} \in \tau(\text{Com}_\mu(G))$ by hypothesis, we deduce that indeed $g_1 \cdots g_k \in \tau(\text{Com}_\mu(G'))$, 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.2 Proof of the Insertion Lemma (Lemma 6.6)

#### Lemma E.6 (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 $u = w'_0w_0w_1w_2w_3' \cdots w_nw'_n$, 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 $v = w''_0w'_1w''_2 \cdots w''_n$, we have $\mu(u) = \mu(v)$ and $\mu(w_0' \cdots w'_n) = \mu(w''_0 \cdots w''_n)$.

**Proof.** Fix the alphabet $A$, the morphism $\mu$, and the group $H$. By Ramsey’s theorem, there exists a constant $B$ such that, for any complete graph $\Gamma$ whose edges are labeled with triples of elements of $H$, if $\Gamma$ 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 $\Gamma$ 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 $g_{i,j} := \mu(w_i \ldots w_{j-1})$, $g'_{i,j} := \mu(w_i w'_1 \cdots w_{j-1} w'_{j-1})$, and $g''_{i,j} := \mu(w'_1 \cdots w'_{j-1})$.

Now, by Ramsey’s theorem, as $\Gamma$ has more than $B$ vertices, it has a monochromatic triangle. This implies that there are $1 \leq l < m < r \leq n$ such that $g_{i,m} = g_{m,r} = g_{i,r}$ and $g'_{i,m} = g'_{m,r} = g'_{i,r}$. Now, as by definition we have $g_{i,r} = g_{i,m} g_{m,r}$, this means that we have $g_{i,r} = g'_{i,r}$, and as $H$ is a group we can simplify and deduce that $g_{i,r} = e$, the neutral element of $H$. We deduce in the same way that $g'_{i,r} = e$. Hence, we have shown $g_{i,r} = g'_{i,r}$, which means that (*) $\mu(w_1 w'_1 \cdots w_{r-1} w'_{r-1}) = \mu(w_1 \cdots w_{r-1})$. Further, we deduce in the same way that (**) $g''_{i,r} = e$.

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 $w'_0, \ldots, w'_{l-1}, w'_l, \ldots, w'_n$. 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
\[
u := w'_0 w_1 w'_1 \cdots w_{l-1} w'_l \cdots w_{r-1} w'_r \cdots w_{r-1} w''_{r-1} \cdots w''_n
\]
we have $\mu(v) = \mu(v')$, and we have (***) $\mu(w'_0 \cdots w'_{l-1} w'_l \cdots w''_n) = \mu(w''_0 \cdots w''_n)$. Let us accordingly define $J \subseteq \{0, \ldots, n\}$ by $\{j \mid j \in J, j < l\} \cup \{j + (r - l) \mid j \in J, j \geq l\}$, 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 $w''_j$ defined from this choice of $J$. From the equality (**), we know that we can replace $(w_l \cdots w_{r-1})$ by $(w_l w'_1 \cdots w_{r-1} w'_{r-1})$ in $u'$ without changing its image by $\mu$, 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 $w_l \cdots w_{r-1} = w_l w''_1 \cdots w_{r-1} w''_{r-1}$; 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 $w''_0 \cdots w''_n$, to establish the second required equality. This concludes the proof.

### E.3 Proof of Theorem 6.2 for the Case of Group Languages

We recall the relevant definitions from the main text. Let $K$ be a group language on the alphabet $A = \{a_1, \ldots, a_k\}$, let $\mu : A^* \rightarrow H$ be the syntactic morphism of $K$, where $H$ is a finite group generated by the $\mu(a_i)$. Consider an instance $I = (S_1, \ldots, S_n)$ to the CSh-problem, where each $S_i$ is a chain of vertices labeled with letters of the alphabet $A$. Recall that $B$ is the bound whose existence is shown in Lemma 6.6 and, using Lemma 6.5 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 $I$ from the main text. We will now state and prove Lemma 6.7.

**Lemma 6.7.** 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 $S_{\text{rare}}$, 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 $S_{\text{freq}}$, then
remove \( a \) from \( A_{\text{freq}} \), add \( a \) to \( A_{\text{rare}} \), remove the \( \leq R \cdot k \) strings that contain \( a \) from \( S_{\text{freq}} \), and add them to \( S_{\text{rare}} \). 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 \( S_{\text{freq}} \) cannot contain any letter of \( A_{\text{rare}} \) and that each letter of \( A_{\text{freq}} \) occurs in at least \( R \cdot k \) different strings of \( S_{\text{freq}} \). By picking \( R \) strings of \( S_{\text{freq}} \) for each letter of \( A_{\text{freq}} \) in a way that does not overlap, we see that \( S_{\text{freq}} \) contains an \( R \)-rich antichain for the alphabet \( A_{\text{freq}} \). Hence, a suitable rare–frequent partition exists.

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

Hence, we assume that we have computed in \( \text{NL} \) a rare–frequent partition of \( I \), given by \( A_{\text{rare}}, A_{\text{freq}}, S_{\text{rare}} \), and (implicitly) \( S_{\text{freq}} \). We will write \( H_{\text{freq}} \) for the subgroup of \( H \) equal to \( \mu(A_{\text{freq}}) \), i.e., the subgroup spanned by \( A_{\text{freq}} \).

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 \( \mu \), 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 6.5, and is obviously commutative. Further recall the morphism \( \text{Com}_\mu : A^* \rightarrow N \) from Section E.1, where \( N \) is a commutative monoid that recognises the inverse image of all elements of \( H \), in particular \( g \). Recall also the relational morphism \( \tau : N \xrightarrow{\tau} H \) defined by \( \tau(x) = \mu(\text{Com}_\mu^{-1}(x)) \).

We will state a condition, called \((*)\), and construct an \( \text{NL} \) 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 \( \rho \) of \( S_{\text{rare}} \), which can be decomposed as \( \rho_1 \cdots \rho_n \), and a sequence \( g_0, \ldots, g_n \) of elements of \( H_{\text{freq}} \), 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 \( H_{\text{freq}} \) 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 \( S_{\text{rare}} \), and check whether the resulting CSh instance has a topological sort using the \( \text{NL} \) algorithm of Proposition 2.3 (because its number of chains 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 \( H_{\text{freq}} \) 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 \( R := nB. \) Focus on \( S_{\text{freq}} \), which has an \( R \)-rich antichain for \( A_{\text{freq}} \), 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 6.5), we know that there is a topological sort \( \sigma = \sigma_0 \cdots \sigma_n \) of \( S_{\text{freq}} \) such that \( \mu(\lambda(\sigma_i)) = g_i \) for each \( i \in \{0, \ldots, n\} \). Now, considering the topological sort \( \rho_1, \ldots, \rho_n \) of \( S_{\text{rare}} \) 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 \( S_{\text{rare}} \) and \( S_{\text{freq}} \), and furthermore \( \mu(\lambda(\sigma_0 \rho_1 \cdots \rho_n \sigma_n)) = \mu(\lambda(\sigma_0))\mu(\lambda(\rho_1))\mu(\lambda(\sigma_1)) \cdots \mu(\lambda(\rho_n))\mu(\lambda(\sigma_n)) \), 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 $\sigma'$ of $I$ achieving $g$, i.e., $\mu(\sigma') = g$. We can decompose it as an interleaving of $S_{\text{rare}}$ and $S_{\text{freq}}$, 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 $S_{\text{rare}}$, and $\sigma_0\cdots\sigma_{n'}$ being a topological sort of $S_{\text{freq}}$ (in particular, we have $\mu(\sigma_0\cdots\sigma_{n'}) \in \tau(\text{Com}_\mu(S_{\text{freq}}))$, which we call condition (#2'). We now use the insertion lemma (Lemma 6.6) 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 $\rho'_i$ which are contiguous, calling the result $\rho_1, \ldots, \rho_n$, where we have (#3) $n < B$, and write $g_i$ the $\mu$-image of the $i$-th $w_i$ which is non-empty: this image is in $H_{\text{freq}}$ because the chains in $S_{\text{freq}}$ are only labeled with letters in $A_{\text{freq}}$. This gives us a topological sort $\rho_1, \ldots, \rho_n$ of $S_{\text{rare}}$, and a sequence $g_0, \ldots, g_n$ of elements of $H_{\text{freq}}$, 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.4 Proof of Theorem 6.2 with District Group Monomials

We now show the complete proof of Theorem 6.2 by adapting the proof of Appendix E.3 from the case of group languages to that of district group monomials. We write $K = K_{0}a_{1}K_{1}\cdots a_{m}K_{m}$, 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 $(S_{1}^{j}, \ldots, S_{n}^{j})$ of strings, and an $m$-tuple of instances $I_{1}^{j}, \ldots, I_{m}^{j}$, with each $I_{j}^{j}$ being a $n$-tuple $((S_{1}^{j})^{j}, \ldots, (S_{n}^{j})^{j})$ as before, with the stipulation that, for each $1 \leq j \leq m$, all $(S_{i}^{j})^{j}$ 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 $(S_{i}^{0})^{j}(S_{i}^{1})^{j}\cdots(S_{i}^{m})^{j}$ 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 $I_{j}$ 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 6.6, use Lemma 6.5 with $k := B$ to obtain $n_{k}$, and let $R := n_{k}$. Given a slicing $I_{0}\ldots I_{m}$ and $I_{1}^{j}\ldots I_{m}^{j}$, a rare–frequent partition of the slicing consists of one partition $A_{\text{rare}}^{j}, A_{\text{freq}}^{j}$ for all $1 \leq j \leq m$, and one global partition of the strings $S_{1}, \ldots, S_{n}$ into rare strings $S_{\text{rare}}$ and frequent strings $S_{\text{freq}}$ (again, the frequent strings are not explicitly represented). We require that (i) for every chain $S$ of $S_{\text{freq}}$, considering its slices $S_{0}^{j}, \ldots, S_{m}^{j}$, for each $1 \leq j \leq m$, the slice $S_{0}^{j}$ contains only letters of $A_{\text{freq}}^{j}$; that (ii) for every $1 \leq j \leq m$, the $S_{j}$ for $S$ in $S_{\text{freq}}$, when seen as a subinstance of $I$ over the alphabet $A_{\text{freq}}^{j}$ contains an $R$-rich antichain; and that (iii) for every $1 \leq j \leq m$, the one non-empty chain of $I_{j}^{j}$ is in $S_{\text{rare}}$.

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.3. For any slicing $I_{0}, \ldots, I_{m}, I_{1}^{j}, \ldots, I_{m}^{j}$, there exists a rare–frequent partition such that $|S_{\text{rare}}| \leq m \cdot R \cdot k^{2}$.
Proof. We apply Lemma 6.7 to each $I_j$ for $1 \leq j \leq m$ to obtain one rare–frequent partition for it, written $A^j_{\text{rare}} \cup A^j_{\text{freq}} = A_j$ and $S^j_{\text{rare}} \cup S^j_{\text{freq}} = I_j$, 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 $S_{\text{rare}}$ to be the union of the chains $S$ of $I$ such that $S^j$ is in $S^j_{\text{rare}}$ for some $0 \leq j \leq m$, plus the chains that are non-empty in some $I'_j$. We take $S_{\text{freq}}$ 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 $A_j = A^j_{\text{rare}} \cup A^j_{\text{freq}}$ for $0 \leq j \leq m$ and the set $S_{\text{rare}}$ of (globally) rare strings of size at most $R \cdot k^2$. Let us further guess the slices $S^j_i$ 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 chain of $S_{\text{rare}}$, 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 $g_0, \ldots, g_m$ 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 $H^j_{\text{freq}}$, the subgroup of $H_j$ spanned by $A^j_{\text{freq}}$. 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 $\gamma_j$); second, reduce the problem to a simpler problem by replacing all chains of $S_{\text{freq}}$ with an additional chain 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 chains, 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 $I_j$, 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 chains 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 $S_{\text{freq}}$.
- 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 chains that contain an occurrence of $a$ in the restriction of slice $I_j$ to $S_{\text{freq}}$, witnessing that it has an $R$-rich antichain for the alphabet $A^j_{\text{freq}}$.
- For each $0 \leq j \leq m$, letting $w$ be the word containing all letters of the restriction of slice $S_j$ to $S_{\text{freq}}$ 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 \(ρ'_0 \cdots ρ'_{n_j}\) of the slice \(S^j_{\text{rare}}\) of \(S_{\text{rare}}\) whose concatenation, interleaved with the singleton elements of the \(I'_j\), is a topological sort of \(S_{\text{rare}}\), and \(g_0^j λ(ρ'_0) \cdots g_{n_j-1}^j λ(ρ'_{n_j-1})g_{n_j}^j = γ_j\). This can be decided in NL by adapting the algorithm of Proposition 2.3 as previously, running it on each slice with one additional chain.

Overall, our algorithm succeeds iff there is a guess of \(γ_i\), of \(S_{\text{rare}}\) (at most \(RK^2\) of them), partitions \(A^j_{\text{freq}} \sqcup A^j_{\text{rare}}\), 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 \(S_{\text{rare}}\) of rare strings (whose slices are written \(S^j_{\text{rare}}\)), and frequent strings \(S_{\text{freq}}\) (with the same convention for slices), partitions \(A^j_{\text{freq}} \sqcup A^j_{\text{rare}}\), a slicing \(I_0, \ldots, I_m\) and \(I'_1, \ldots, I'_m\), a topological sort of \(S_{\text{freq}}\) constituting of topological sorts \(ρ'_0 \cdots ρ'_{n_j}\) of each \(S^j_{\text{rare}}\) 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 \(γ_0, \ldots, γ_m\) of \(H_0 × \cdots × H_m\), 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 \(A^j_{\text{freq}}\), and contains an \(R\)-rich antichain on the sub-alphabet \(A^j_{\text{freq}}\).
- 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 \(S_{\text{freq}}^j\) be the slice of \(S_{\text{freq}}\) defined in the expected way, we have \(g_0^j \cdots g_{n_j}^j \in τ_j(\text{Com}_{μ_j}(S_{\text{freq}}^j)).\)
- \((#)\) For all \(0 \leq j \leq m\), we have \(g_0^j λ(ρ'_0) \cdots g_{n_j-1}^j λ(ρ'_{n_j-1})g_{n_j}^j = γ_j\)

We claim that we can deduce from this the existence of a witnessing topological sort. To do this, as before, we will use Lemma 6.3 in the \(S_{\text{freq}}^j\) for all \(0 \leq j \leq m\). From our definition of \(R\), as \(n_j < B\), as \(S_{\text{freq}}^j\) contains an \(n_k\)-rich antichain (seen as an instance on the sub-alphabet \(A^j_{\text{freq}}\)), as \(g_0^j \cdots g_{n_j}^j \in τ_j(\text{Com}_{μ_j}(S_{\text{freq}}^j))\), there is a topological sort \(σ_1^j \cdots σ_{n_j}^j\) of \(S_{\text{freq}}^j\) such that \(μ_j(λ(σ_i^j)) = g_i^j\) 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 \(ρ'_0 \cdots ρ'_{n_j}\) of \(S^j_{\text{rare}}\) interleaved with the topological sort \(σ_1^j, \ldots, σ_{n_j}^j\) of \(S_{\text{freq}}^j\), 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 6.3 to argue that there exists a rare–frequent partition consisting of a partition \(S_{\text{rare}} \sqcup S_{\text{freq}}\) of the chains, and \(A^j_{\text{rare}} \sqcup A^j_{\text{freq}}\) of the alphabets \(A_j\), such that \(|S_{\text{rare}}| ≤ m \cdot R \cdot k^2\). In each slice, the witnessing topological sort must consist of a topological sort of the \(S^j_{\text{rare}}\) interleaved with topological sorts of the \(S^j_{\text{freq}}\). As in the previous proof, we now use Lemma 6.6 to argue that we can assume that there are at most \(n_j\) such insertions, without changing the
We show the side result on the language in Proof. We anticipate on some later definitions used in the appendix. Consider an input instance $\lambda$. Hence, we have indeed shown that $\text{NL}$ to the instance $\lambda$. We consider the topological sort of $\lambda$. Let $\sigma$ be a subset of $\gamma$. Let $\alpha$ be the number of subfactors of the form $a+b^+$ in the word $w_1$ achieved by $\sigma_1$. Now, consider the topological sort $\sigma_2$ obtained by combining $\sigma_-$ and $\sigma_+$ with $\sigma_2$ achieving the word $abab$. Let $n_2$ be the number of subfactors of the form $a+b^+$ in the word $w_2$ achieved by $\sigma_2$. We claim that $n_2 = n_1 + 1$. Indeed, consider the subfactor $a+b^+$ that contains $\sigma_1$ in $\sigma_1$. In $\sigma_2$, the other subfactors are unchanged, and this subfactor is split into two subfactors, one ending at the first $b$ of $\sigma_2$, the other one starting at the second $a$ of $\sigma_2$. So indeed $n_2 = n_1 + 1$. Hence, one of $n_1, n_2$ is even, and the corresponding $\sigma_i$ witnesses that $\text{G}$ is a positive instance to CSh($\lambda$).

Hence, it suffices to handle the case where $G$ has no 2-rich antichain. This implies that there is one symbol $\alpha \in A$ which occurs in at most one chain $S$, which means that the other chains $S_1, \ldots, S_m$ 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 chain $S'$ of elements labeled $\beta$ whose length is $\sum_i |S_i|$. Hence, we can reduce the problem in $\text{NL}$ to the instance $\{S, S'\}$. As it has two chains, we can conclude in $\text{NL}$ using Proposition 2.3. Hence, we have indeed shown that CSh($\lambda$) is in $\text{NL}$.
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. Preprint: https://a3nm.net/publications/amarilli2017possible.pdf To appear.
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. Samuel Eilenberg. Automata, languages, and machines. Vol. A. Academic Press, 1974. Pure and Applied Mathematics, Vol. 58.
10. Joey Eremondi, Oscar H Ibarra, and Ian McQuillan. On the complexity and decidability of some problems involving shuffle. 2016. arXiv:1608.01199.
11. 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.
12. D. R. Fulkerson. Note on Dilworth’s decomposition theorem for partially ordered sets. In Proc. Amer. Math. Soc, 1955.
13. M. R. Garey and David S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. 1979.
14. Michael R. Garey and David S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. SIAM J. Comput, 1975.
15. Mai Gehrke, Serge Grigorieff, and Jean-Éric Pin. Duality and equational theory of regular languages. In Proc. ICALP, 2008.
16. Antonio Cano Gómez, Giovanna Guaiana, and Jean-Eric Pin. When does partial commutative closure preserve regularity? In Proc. ICALP, 2008.
17. M. Holcombe. Algebraic Automata Theory. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1982. doi:10.1017/CBO9780511525889
18. Neil Immerman. Nondeterministic space is closed under complementation. SIAM J. Comput, 17(5), 1988.
19. David S. Johnson. The NP-completeness column: an ongoing guide. Journal of Algorithms, 5(2), 1984.
20. John Kececioglu and Dan Gusfield. Reconstructing a history of recombinations from a set of sequences. Discrete Applied Mathematics, 88(1-3), 1998.
21. Takayuki Kimura. An algebraic system for process structuring and interprocess communication. In Proc. STOC, 1976.
22. Anthony Mansfield. On the computational complexity of a merge recognition problem. Discrete Applied Mathematics, 5(1), 1983.
23. Robert McNaughton and Seymour Papert. Counter-Free Automata. MIT Press, 1971.
24. W. F. Ogden, W. E. Riddle, and W.C. Rounds. Complexity of expressions allowing concurrency. In Proc. POPL, 1978.
25. Jean-Éric Pin. A variety theorem without complementation. Russian Mathematics (Izvestija vuzov. Matematika), 39, 1995.
26. Jean-Éric Pin and Pascal Weil. Polynominal closure and unambiguous product. TCS, 30(4), 1997.
27. Romeo Rizzi and Stéphane Vialette. On recognizing words that are squares for the shuffle product. TCS, 2017.
28. M. P. Schützenberger. Sur le produit de concaténation non ambigu. Semigroup forum, 13, 1976/77.


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

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

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

32 Pascal Tesson and Denis Thérien. Diamonds are forever: the variety DA. *Semigroups, algorithms, automata and languages*, 1, 2002. [http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.5612](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.5612)

33 Pascal Tesson and Denis Thérien. Complete classifications for the communication complexity of regular languages. *TCS*, 38(2), 2005. [doi:10.1007/s00224-004-1190-2](https://doi.org/10.1007/s00224-004-1190-2).

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

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

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