Topological Sorting with Regular Constraints

Antoine Amarilli
LTCI, Télécom ParisTech, Université Paris-Saclay

Charles Paperman
Université de Lille

Abstract

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

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Topological sorting, shuffle problem, regular language

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.115

Related Version A full version of the paper is available at [5], `https://arxiv.org/abs/1707.04310`.

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

1 Introduction

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

CTS relates to many applications. For instance, many *scheduling* applications use a dependency graph [1] of tasks, and it is often useful to express other constraints, e.g., some tasks must be performed by specific workers and we should not assign more than p successive tasks to the same worker. We can express this as a CTS-problem: label each task by the worker which can perform it, and consider the target regular language K containing all words where the same symbol is not repeated more than p times. In *concurrency* applications,

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 115; pp. 115:1–115:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

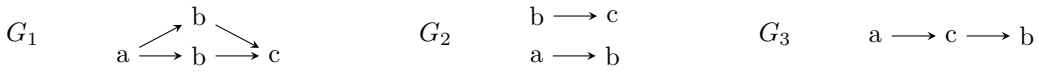

Figure 1 Example labeled DAGs on the alphabet $A = \{a, b, c\}$

we may consider a program with multiple threads, and want to verify that there is no linearization of its instructions that exhibits some unsafe behavior, e.g., executing a read before a write. To search for such a linearization, we can label each instruction with its type, and consider CTS with a target language describing the behavior that we wish to detect. CTS can also be used in uncertain data management tasks, to reason about the possible answers of aggregate queries on uncertain ordered data [4]. It can also be equivalently phrased in the language of partial order theory: seeing the labeled DAG as a *labeled partial order* $<$, we ask if some *linear extension* achieves a word in K.

We thus believe that the CTS-problem is useful, and natural, but we are not aware of previous work studying it, except for a special case called the *shuffle problem*. This problem deals with the *interleaving* of strings, as studied, e.g., in concurrent programming languages [18, 21], computational biology [17], and formal languages [10, 8, 26]. Specifically, we are given a tuple of strings, and we must decide if they have some interleaving that falls in the target language K. This problem was known to be NP-complete [19, 32, 16] when the target language K is given as input (in addition to the tuple of strings), even when K consists of just one target string. To rephrase this shuffle problem in our context, we call *constrained shuffle problem* (CSh) the special case of CTS where we require input DAGs to be a union of directed path graphs (corresponding to the strings).

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

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

- In Section 3, we present our hardness results. We recall the results of [32] on the shuffle problem, and present a general *shuffle reduction* technique to show hardness for more languages. We use it in particular to show that CSh[$(ab)^*$], hence CTS[$(ab)^*$], are NP-hard, and extend this to several other languages.
- In Section 4, we present tractability results. We show that CTS[K], hence CSh[K], is in non-deterministic logspace (NL) when K is a union of *monomial languages*, i.e., of languages of the form $A_1^* a_1 \cdots A_{n-1}^* a_{n-1} A_n^*$, with the a_i being letters and the A_i being subalphabets. Such languages can be used for applications such as pattern matching, e.g., with the language $A^* u A^*$ for a fixed pattern $u \in A^*$. We also show tractability for other languages that are not of this form, e.g. $(ab)^* + A^* aa A^*$ and variants thereof, using different techniques such as Dilworth's theorem [9].
- In Section 5, we use our hardness and tractability results to show a coarser dichotomy result. Specifically, we give an alternative phrasing of the CTS and CSh problems using

semiautomata and DAGs with *multi-letter* labels: this amounts to closing the tractable languages under intersection, inverse morphism, complement, and quotients. In this phrasing, when the semiautomaton is counter-free, we can show that the problems are either in NL or NP-complete. This dichotomy is effective, i.e., the criterion on the semiautomaton is decidable, and it turns out to be the same for CTS and CSh.

- In Section 6, we focus on the constrained shuffle problem, and lift the counter-free assumption of the previous section. We show that CSh$[K]$ is tractable when K is a *group language* or more generally a union of *district group monomials*. This tractability result is the main technical contribution of the paper, with a rather involved proof. It implies, e.g., that the following problem is in NL for any fixed finite group H: given $g \in H$ and words $w_1, \ldots, w_n$ of elements of H, decide whether there is an interleaving of the w_i which evaluates to g according to the group operation.

2 Problem Statement and Main Results

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

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

We now define the *constrained shuffle problem* (CSh). Given two words $u, v \in A^*$, the *shuffle* [32] of u and v, written $u \sqcup\!\sqcup v$, is the set of words that can be obtained by interleaving them. Formally, a word $w \in A^*$ is in $u \sqcup\!\sqcup v$ iff there is a partition $P \sqcup Q$ of $\{1, \ldots, |w|\}$ such that $w_P = u$ and $w_Q = v$, where w_P denotes the sub-word of w where we keep the letters at positions in P, and likewise for w_Q. The *shuffle* $\sqcup\!\sqcup(U)$ of a tuple of words U is defined by induction as follows: we set $\sqcup\!\sqcup() := \{\epsilon\}$, set $\sqcup\!\sqcup(u) := \{u\}$, and set $\sqcup\!\sqcup(u_1, \ldots, u_n, u_{n+1}) := \bigcup_{v \in \sqcup\!\sqcup(u_1, \ldots, u_n)} v \sqcup\!\sqcup u_{n+1}$. The *constrained shuffle problem* CSh$[K]$ for a fixed language $K \subseteq A^*$ is defined as follows: given a tuple of words U, determine if $K \cap \sqcup\!\sqcup(U)$ is nonempty. Of course, CSh$[K]$ is a special case of CTS$[K]$: we can code any tuple of words U as an A-DAG G_U by coding each $u \in U$ as a directed path graph $v_1 \to \cdots \to v_{|u|}$ with $\lambda(v_i) = u_i$ for all $1 \leq i \leq |u|$. Thus, we will equivalently see inputs to CSh as tuples of words (called *strings* in this context) or as A-DAGs that are unions of directed path graphs.

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

In this work, we study the complexity of the problems CTS$[K]$ and CSh$[K]$ depending on the language K. Clearly we can always solve these problems by guessing a topological sort (or an interleaving), and verifying that it achieves a word in K. Hence, the complexity

is always in NP^K, that is, in non-deterministic PTIME with an oracle for the *word problem* of K, which we can call to test if an input word in is K:

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

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

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

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

3 Hardness Results

Our hardness results are based on the *shuffle problem* of formal language theory which asks, given a word $w \in A^*$ and a tuple U of words of A^*, whether $w \in \sqcup\!\sqcup(U)$. This problem is known to be NP-hard already on the alphabet $\{a, b\}$ (see [32]). The shuffle problem is different from CSh, because the target word of the shuffle problem is given as input, whereas the target regular language of CSh is fixed. However, the hardness of the shuffle problem directly implies the hardness of CSh, hence of CTS, for a well-chosen target language:

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

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

In this section, we will refine this approach to show hardness for more languages. We first recall another initial hardness result from [32]. We then introduce a general *shuffle reduction* technique to show the hardness of languages by reducing from other hard languages. Last, we show that CTS and CSh are hard for the language $(ab)^*$ and for other languages.

Initial hard family. To bootstrap the hardness results of [32] on the shuffle problem (on input words) to our CSh-problem (on fixed languages), we generalize the definition of CSh to a *regular language family* $\mathcal{K}$, i.e., a (generally infinite) family of regular languages, each of which is described as a regular expression. The CSh-problem for $\mathcal{K}$, written $\mathrm{CSh}[\mathcal{K}]$, asks, given a regular expression $K \in \mathcal{K}$ and a set of strings U, whether $K \cap \sqcup\!\sqcup(U)$ is nonempty. In other words, we no longer fix one single target language but a family $\mathcal{K}$ of target languages, and the input chooses one target language from the family $\mathcal{K}$. The following is then shown in [32] by reducing from UNARY-3-PARTITION [13]:

▶ **Lemma 3.2. ([32], Lemma 3.2)** *Let $\mathcal{K} := \{(a^ib^i)^* \mid i \in \mathbb{N}\}$. Then* $\mathrm{CSh}[\mathcal{K}]$ *is NP-hard.*

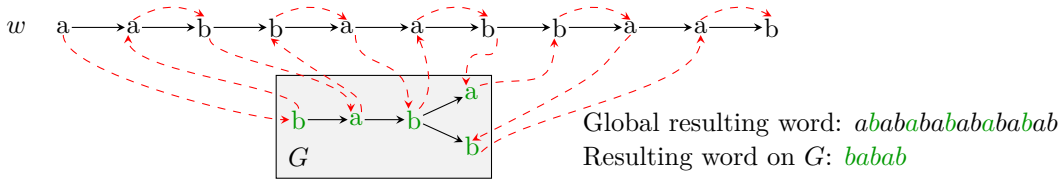

Figure 2 Example of a shuffle reduction from $K := (ba)^*b$ to $K' := (ab)^*$

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

The intuition for the shuffle reduction is as follows: to reduce from K to K', given an input A-DAG G, we build an A-DAG G' formed of G plus an additional directed path labeled by a word w. Thus, any topological sort σ' of G' must be the interleaving of w and of a topological sort σ of G. Now, if we require that σ' achieves K', the presence of w can impose specific conditions on σ. Intuitively, if w is sufficiently long and "far away" from all words of K', then σ' must "repair" w to a word of K' by inserting symbols from G, so the insertions performed by σ may need to be in a specific order, i.e., σ may be forced to achieve a word of K. This means that solving CTS[K'] on G' allows us to solve CTS[K] on G. This intuition is illustrated on Figure 2: to achieve a word of $K' := (ab)^*$ on the DAG G', a topological sort must enumerate elements from G to insert them at the appropriate positions in w, achieving a word of $K := (ba)^*b$. We call *filter sequence* a family of words like w that allow us to reduce any CTS[K]-instance to CTS[K']. Formally:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4 Tractability Results

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

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

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

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

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

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

Monomials. We will now show that CTS is tractable for an important family of languages (and unions of such languages): the *monomial* languages. Having fixed the alphabet A, a *monomial* is a language of the form $A_1^* a_1 A_2^* a_2 \cdots a_n A_{n+1}^*$ with $a_i \in A$ and $A_i \subseteq A$ for all i. In particular, we may have $A_i = \emptyset$ so that $A_i^* = \epsilon$: hence, for every word $u \in A^*$, the language $A^* u A^*$ is a monomial language, which intuitively tests whether a word contains the pattern u. Several decidable algebraic and logical characterizations of these languages are known; in particular, unions of monomials are exactly the languages that are definable in the first-order logic fragment $\Sigma_2[<]$ of formulas with quantifier prefix $\exists^* \forall^*$, and it is decidable to check if a regular language is in this class [24, 22]. We show:

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

Proof sketch. Let K be $A_1^* a_1 A_2^* a_2 \cdots A_n^* a_n A_{n+1}^*$. We can first guess in NL the vertices $v_1, \ldots, v_n$ to which the $a_1, \ldots, a_n$ are mapped, so all that remains is to check, for each such guess, whether we can match the remaining vertices to the A_i. We proceed by induction on n. The base case of $n = 0$ (i.e., $K = A_1^*$) is trivial. For the induction step, using the fact that NL = co-NL (see [15, 28]), we check that the descendants of the last element v_n are all in A_{n+1}^*, and then we compute the set S of vertices that *must* be enumerated before v_n: they are the ancestors of the v_i, and the ancestors of any vertex labeled by a letter in $A \setminus A_{n+1}$. We then use the induction hypothesis to check in NL whether S has a topological sort that achieves a word in $A_1^* a_1 \ldots A_{n-1}^* a_{n-1} A_n^*$. ◀

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

▶ **Proposition 4.4.** *Let $A := \{a, b\}$ and $K := (ab)^* + A^* aa A^*$. The problem* CTS$[K]$ *(hence* 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 the long version [5]); and the proof technique is different.

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

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

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

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

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

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

Proof sketch. We test in NL whether the input A-DAG G contains an antichain C of size $2i$: if it does, then at least i vertices in C must have the same label, and we can enumerate them in succession to achieve $A^*a^iA^*$ or $A^*b^iA^*$, so G is a positive instance. Otherwise, G has width $< 2i$, and Dilworth's theorem [9] implies that its elements can be partitioned into chains, so that CTS can be solved in NL following a dynamic algorithm on them. ◀

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

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

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

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

5 A Coarser Dichotomy Theorem

In the two previous sections, we have established some intractability and tractability results about the constrained topological sort and constrained shuffle problems for various languages. Remember that our end goal would be to characterize the tractable and intractable languages, and show a dichotomy (Conjecture 2.3). This is difficult, and one reason is that the class of tractable languages is not "well-behaved": while it is closed under the union operator (Corollary 4.2), it is is *not* closed under intersection, complement, and other common operations. This makes it difficult to study tractable languages using algebraic language theory [23].

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

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

The three last results of this proposition also apply to the constrained topological sort problem, but the first one does not, and in fact CTS-tractable languages *are* closed under quotients. This observation implies that there are regular languages K such that $\mathrm{CSh}[K]$ is tractable but $\mathrm{CTS}[K]$ is NP-hard; one concrete example is $K := b^*A^* + aaA^* + (ab)^*$ (see long version [5]). We sketch the proof of Proposition 5.1:

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

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

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

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

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

Proposition 5.1 suggests that tractable languages would be easier to study algebraically if we ensured that they were closed under all these operations, i.e., if they formed a variety [23]. In this section, we enforce this by moving to an alternative phrasing of the CTS and CSh problems. This allows us to leverage algebraic techniques and show a dichotomy theorem in this alternative phrasing, under an additional *counter-free* assumption. We first present the alternative phrasing, and then present the additional assumption and our dichotomy result.

Alternative phrasing. The first change in our alternative phrasing is that the input DAG G will now be an A^*-*DAG*, i.e., a DAG labeled with words of A^* rather than letters of A. As before, a topological sort σ of G *achieves* a word $\lambda(\sigma) \in A^*$ obtained by concatenating the λ-images of the vertices of G in the order of σ: but vertex labels are now "atomic" words

whose letters cannot be interleaved with anything else. The *multi-letter* CTS and CSh problems are the variants defined with A^*-DAGs; intuitively, this ensures that tractable languages are closed under inverse morphisms.

The second change is that we will not fix one single target language, but a *semiautomaton* [14], i.e., an automaton where initial and final states are not specified. Formally, a semiautomaton is a tuple (Q, A, δ) where Q is the set of states, A is the alphabet, and $\delta : Q \times A \to Q$ is the transition function; we extend δ to words as usual by setting $\delta(q, \epsilon) := q$ and $\delta(q, u_1 \cdots u_{n+1}) := \delta(\delta(q, u_1), u_2 \cdots u_{n+1})$. We will fix the target semiautomaton, and the initial and final states will be given in the input instance (in addition to the DAG). This enforces closure under quotients (by choosing the initial and final states) and complement (by toggling the final states). Further, to impose closure under intersection, the input instance will specify a *set* of pairs of initial-final states, with a logical AND over them. The question is to determine whether the input DAG achieves a word accepted by *all* the corresponding automata; and this enforces closure under intersection.

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

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

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

We conclude the section by introducing some technical tools used for this result and for Section 6, and by giving a proof sketch. The criterion of the dichotomy on S is phrased in terms of the *transition monoid* of S, which we now define (see, e.g., [23] for details). Remember that a monoid is a set that has an associative binary operation and a neutral element. The *transition monoid* $T(S)$ of a semiautomaton $S = (Q, A, \delta)$ is the set of functions $f : Q \to Q$ that are "achieved" by S in the following sense: there is a word $u \in A^*$ such that $\delta(q, u) = f(q)$ for all $q \in Q$. In particular, the neutral element is the identity function, which is achieved by taking $u := \epsilon$; and the binary operation on $T(S)$ is function composition, which is associative. Note that the transition monoid is finite and can be computed from S.

We assumed that S is counter-free, and this is equivalent [20] to saying that $T(S)$ is in the class $\mathbf{A}$ of *aperiodic* finite monoids (formally defined by the equation $x^{\omega+1} = x^\omega$ where ω is the idempotent power [23] of the monoid). Within $\mathbf{A}$, our dichotomy criterion on $T(S)$ is based on a certain subclass of $\mathbf{A}$, called $\mathbf{DA}$ (see [30]): S is tractable iff $T(S)$ is in $\mathbf{DA}$, and it is PSPACE-complete [31] to test whether this holds (using the formal definition of $\mathbf{DA}$ by the equation $(xy)^\omega x (xy)^\omega = (xy)^\omega$). We can now sketch the proof of Theorem 5.2:

Proof sketch. We first show that if $T(S)$ is in **DA** then the multi-letter CTS and CSh problems for S are in NL. For this, we rely on one characterization of **DA** (from [30]): if $T(S)$ is in **DA** then the regular languages recognized by S (for any set of initial-final states) are unions of *unambiguous monomials*, in particular they are unions of monomials, so we have tractability by Corollary 4.2 and Theorem 4.3.

For the converse direction, we use a second characterization of **DA** (from [29]): if $T(S)$ is *not* in **DA** then there is a choice of initial-final state pairs for which S computes a language K whose inverse image by some morphism is either $(ab)^*$ or $(ab+b)^*$. We know that these languages are intractable (Theorem 3.6 and Proposition 3.9) so we conclude by showing a PTIME reduction from one of these two languages: this is possible in our alternative problem phrasing, in particular using the multi-letter labels to invert the morphism. ◀

6 Lifting the Counter-Free Assumption for CSh

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

To extend Theorem 5.2 for the CSh-problem, we will again classify the semiautomata S based on their transition monoid $T(S)$. However, instead of **DA**, we will use the two classes **DO** and **DS** introduced in [27] (formally **DO** is defined by the equation $(xy)^\omega(yx)^\omega(xy)^\omega = (xy)^\omega$ and **DS** by the equation $((xy)^\omega(yx)^\omega(xy)^\omega)^\omega = (xy)^\omega$ for ω the idempotent power). Both **DO** and **DS** are supersets of **DA**, specifically we have $\mathbf{DA} \subseteq \mathbf{DO} \subseteq \mathbf{DS}$, and we can test in PSPACE in S whether $T(S)$ is in each of these classes [31]. Our main result is then:

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

This result generalizes Theorem 5.2 for the CSh-problem, because both **DO** and **DS** collapse to **DA** for aperiodic monoids (see [27] and [2, Chapter 8]); formally, $\mathbf{DO} \cap \mathbf{A} = \mathbf{DS} \cap \mathbf{A} = \mathbf{DA}$. However, **DO** covers more languages than **DA**: the main technical challenge to prove Theorem 6.1 is to show that CSh is tractable for these languages. One important example are the *group languages* over A: these are the regular languages recognized, for some choice of initial-final state pairs, by a semiautomaton S over A such that $T(S)$ is a group. A more general example are *district group monomials*, which are the languages of the form $K_1a_1 \cdots K_na_nK_{n+1}$ where, for all i, we have $a_i \in A$ and K_i is a group language over some alphabet $A_i \subseteq A$. Note that district group monomials are more expressive than the *group monomials* defined in earlier work [25] (which set $A_i := A$ for all i), and they also generalize the monomials that we studied in Section 4 (any A_i^* is trivially a group language over A_i, even though it is not a group language over A). In fact, to prove Theorem 6.1, what we need is to generalize Theorem 4.3 (for CSh) from monomials to district group monomials:

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

Note that this theorem, like Theorem 4.3, applies to the original phrasing of CSh, not the alternative phrasing with semiautomata and multi-letter DAGs. Thus, Theorem 6.2 implies that the original CSh-problem is tractable for many languages that we had not covered previously, e.g., $(ab^*a+b)^*c(ba^*b+a)^*$, the language testing whether there is one c preceded by an even number of a and followed by an even number of b. The proof of Theorem 6.2 is our main technical achievement, and we sketch it below (see the long version [5] for details):

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

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

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

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

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

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

7 Conclusion and Open Problems

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

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

Another direction would be to connect CSh and CTS to the framework of *constraint satisfaction problems* (CSP) [11], which studies the complexity of homomorphism problems for fixed "constraints" (right-hand-side of the homomorphism). If this were possible, it could lead to a better understanding of our tractable and hard cases. However, CTS does not seem easy to rephrase in CSP terms: topological sorts and regular language constraints seems hard to express in terms of homomorphisms, even in extensions such as *temporal* CSPs [6, 7].

One last question would be to investigate CTS and CSh for *non-regular* languages. The simplest example is the Dyck language, which appears to be NP-hard for CTS (at least in the multi-letter setting), but tractable for CSh, via a connection to scheduling; see [12], problem SS7. More generally, CTS and CSh could be studied, e.g., for context-free languages, where the complexity landscape may be equally enigmatic.

References

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

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

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

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

5 Antoine Amarilli and Charles Paperman. A dichotomy on constrained topological sorting. *CoRR*, abs/1707.04310, 2017. `arXiv:1707.04310`.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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