Theoretical Computer Science 489–490 (2013) 1–20

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Shuffled languages—Representation and recognition

Martin Berglund [*], Henrik Björklund, Johanna Björklund [1]

Computing Science Department, Umeå University, 901 87 Umeå, Sweden

A R T I C L E I N F O

Article history:
Received 9 December 2012
Received in revised form 15 April 2013
Accepted 25 April 2013
Communicated by Z. Esik

Keywords:
Interleaving
Shuffle languages
Membership problems

A B S T R A C T

Language models that use interleaving, or shuffle, operators have applications in various areas of computer science, including system verification, plan recognition, and natural language processing. We study the complexity of the membership problem for such models, in other words, how difficult it is to determine if a string belongs to a language or not. In particular, we investigate how interleaving can be introduced into models that capture the context-free languages.

1. Introduction

We study the membership problem for various language classes that make use of the shuffle operator $\odot$. When applied to a pair of strings u and v, the operator returns the set of all possible interleavings of the symbols in u and v. For example, the shuffle of ab and cd is $\{abcd, acbd, acdb, cabd, cadb, cdab\}$. This type of interleaving operation has been considered as far back as in a 1965 paper by S. Ginsburg and E. Spanier [22]. The operator is lifted to languages by defining $\mathcal{L}_1 \odot \mathcal{L}_2$ to be the set $\bigcup\{u \odot v \mid u \in \mathcal{L}_1, v \in \mathcal{L}_2\}$. We also consider the shuffle closure operator, whose relationship to the shuffle operator resembles that of the Kleene star to concatenation, iterating the shuffle operation. As our starting point, we take the *shuffle languages* considered by Gischer [23] and by Jedrzejowicz and Szepietowski [31]. These are the languages defined by regular expressions augmented with the shuffle operation and the shuffle closure operators, inspired by Flow Expressions [43].

Shuffling of languages is of interest in a number of different areas.

- One of the original motivations for studying shuffle is in the *modeling* and *verification* of systems, where shuffling is useful for reasoning about interleaved or parallel processes [19,43,41]. There is a close connection between shuffle languages and Petri nets [23,19,8,6].
- The shuffle operator is used in *XML database systems* for schema definitions, see for example the work on schema languages by Gelade et al. [20].
- In *plan recognition*, the challenge is to identify an agent's objectives, based on observations of its actions [11,42]. In a generalized version, a number of independent agents perform their actions in an interleaved fashion. To model such a scenario, one could combine shuffle operators and context-free grammars [29]. For this approach to be tractable, the membership problem for the resulting languages must remain efficiently solvable.
- In *natural language processing*, there is a growing interest in linguistic models for languages with relatively free word order. Recent work in this direction includes parse algorithms for dependency grammars [39,32].

[*] Corresponding author. Tel.: +46 703342841.
E-mail addresses: mbe@cs.umu.se (M. Berglund), henrikb@cs.umu.se (H. Björklund), johanna@cs.umu.se (J. Björklund).

[1] Née Högberg.

0304-3975/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.tcs.2013.04.022

Table 1
Summary of results for the membership problem. The shuffle languages are abbreviated by Sh, the regular by Reg, and the deterministic linear context-free by DLCF. The results of this paper appear in bold face. In the case of CFSA membership NP-hardness follows from [40] and inclusion is demonstrated here.

	Sh	Reg $\odot$ CF	Sh $\odot$ CF	DLCF $\odot$ DLCF	CFSA
Non-uniform	P	**P**	**P**	**NPC**	**NPC**
Uniform	**W[1]-hard**	**P**	NPC	**NPC**	**NPC**

Many fundamental questions regarding the membership problem for shuffled languages remain unanswered. We consider and answer some of them in this paper. In particular, we are interested in language classes that capture the context-free languages. Among the above application areas, such languages are primarily of interest in plan recognition and natural language processing.

It is important to distinguish between the *uniform* and the *non-uniform* version of the membership problem. In the uniform version, both the string and a representation of the language is given as input. It is therefore relevant *how* the language is represented. In the non-uniform version, only the string to be tested is considered as input. The language is fixed, so its representation never enters into the equation.

Contributions. To facilitate the study of languages combining restricted forms of recursion and interleaving, we define *Concurrent Finite State Automata* (CFSA) which have an expressive power between those of context-free grammars and context-sensitive grammars. These automata can be viewed as ground tree rewriting systems (see, e.g., [33,12]) used as language acceptors. We show that the emptiness problem for CFSA is solvable in polynomial time, list the closure properties of the automata, and identify the language classes that correspond to certain syntactic restrictions.

Our results for the complexity of the membership problems for various language classes are summarized in Table 1. It should be noted that all problems we consider, except the membership problem for CFSA, are trivially in NP. For the full class of languages recognized by CFSA, we show that both the uniform and the non-uniform membership problem are NP-complete.

For the *shuffle languages* (as used in [23,31]), the *uniform membership problem* is NP-complete [44,2,35], while the *non-uniform membership problem* can be decided in polynomial time [31]. We shed further light on the complexity of the membership problem by establishing that the uniform version, when parameterized by the number of shuffle operations, is hard for the complexity class W[1]. This result suggests a strong dependence on the number of shufflings. For this reason, we do not expect to find a particularly efficient algorithmic solution to the non-uniform membership problem for language definitions involving many shufflings, even when it is theoretically polynomial.

For the interleaving of a regular language and a context-free language, we show that the uniform (and thus also the non-uniform) membership problem can be solved in polynomial time. The regular language is assumed to be represented by a nondeterministic finite automaton and the context-free language by a context-free grammar.

For the shuffling of a shuffle language and a context-free language, the uniform problem is NP-hard, since this holds already for the shuffle languages. In one of our main results, we show that the non-uniform problem, on the other hand, is solvable in polynomial time.

It is known that already the non-uniform version of the membership problem is NP-hard for the shuffling of two deterministic context-free languages [40]. We strengthen this result by demonstrating that it holds even for the shuffle of deterministic *linear* context-free languages. Here, the results in [28] may also be of interest, since it draws parallels between two-stack Turing machines and the shuffle of context-free languages, which is similar to the technique we use.

It should be noted that we only investigate which broad complexity classes the problems belong to. In particular, for the problems that belong to P, our aim has not been to find optimal algorithms. Future work in this direction includes finding the exact complexities of these problems, as well as heuristic algorithms and tractable restrictions of the NP-complete problems.

Related work. Various aspects of shuffling have been studied in formal language theory and its effects on regular languages have received particular interest. Câmpeanu et al. establish $2^{mn} - 1$ as a tight upper bound on the state complexity of the shuffle of two regular languages [10], represented by finite deterministic automata of size m and n, respectively. Biegler et al. provide a similar result for singleton languages and identify properties that trigger an exponential blow-up in state complexity [5]. On the descriptive side, it follows from a result by Gruber and Holzer that the addition of a shuffle operator to regular expressions may reduce representation sizes exponentially [25]. A generation algorithm with linear complexity for approximate size sampling (i.e., random generation) of regular specifications including shuffle has been provided by Darrasse et al. [14]. Brzozowski et al. consider the complexity of *ideal* languages [9], which are regular languages invariant under shuffle with the universal language [26]. Further results for sub-families of the regular languages are found in [24,27,4,13]. Warmuth and Haussler gives complexity results (NP-completeness proofs) for some specific and very simple shuffle languages, notably, given two strings v and w as input deciding whether $v \in w^{\odot}$ is NP-complete [44].

Shuffling has also been investigated in a more algebraic setting. The axiomatization of shuffle theory was addressed by Ésik together with Bloom [7] and Bertol [16]. Another important direction is shuffling on *trajectories*, which is the idea of shuffling two strings in a way informed by a third string, the trajectory. That is, for example $abc \odot_{011010} def = adebfc$ where the binary string is the trajectory which identifies a specific interleaving of *abc* and *def*. This is then generalized to languages, which makes the shuffle considered here a special case, where the trajectory is always the universal language (e.g., $x \odot y$ is equivalent to $x \odot_{\{0,1\}^*} y$). See [34] for more information on shuffle on trajectories.

Another related formalism is *permutation languages*, first considered by Nagy [37,38], which allow rules of the form $AB \to BA$ in an otherwise normal context-free grammar. These rules can be applied to interchange positions of adjacent non-terminals in intermediary derivation steps, and thus allows for certain forms of shuffling. The present article extends [3], which was presented at the *5th International Conference on Language and Automata Theory and Applications (LATA) 2011*.

2. Preliminaries

Sets and numbers. If S is a set, then $|S|$ denotes the cardinality of S, S^* is the set of all finite sequences of elements of S, and $precl(S)$ is the set of all finite prefix-closed subsets of S^*. In other words, for every $S' \in precl(S)$, if $uv \in S'$ for some $u, v \in S^*$ then $u \in S'$. We write $\mathbb{N}$ for the natural numbers. For $k \in \mathbb{N}$, we write $[k]$ for $\{1, \ldots, k\}$. Note that $[0] = \emptyset$. The domain of a mapping f is denoted $dom(f)$.

A *total order* on a set S is an binary relation $\leq$ on S that is antisymmetric, transitive, and such that for every $s, s' \in S$, either $s \leq s'$ or $s' \leq s$.

An *alphabet* is a finite nonempty set. Let Σ be an alphabet and let ε be the empty string, then $\Sigma \cup \{\varepsilon\}$ is denoted by Σ_ε. The length of a string $w = \alpha_1 \cdots \alpha_n$ is written $|w|$, and for every $\alpha \in \Sigma$, $|w|_\alpha = |\{i \in [n] \mid \alpha_i = \alpha\}|$. The *reversal* of $w = \alpha_1 \cdots \alpha_n$ is $w^{\mathcal{R}} = \alpha_n \cdots \alpha_1$. For strings $w, w' \in \Sigma$ the concatenation is usually denoted ww'. When necessary for clarity, however, the operation is written explicitly as $w \cdot w'$. Concatenation distributes over sets, e.g. for $S, S' \subseteq \Sigma^*$ we have $w \cdot S = \{w \cdot w' \mid w' \in S\}$, and $S \cdot S' = \{w \cdot w' \mid w \in S, w' \in S'\}$.

Trees. The set T_Σ of *(unranked) trees* over the alphabet Σ consists of all mappings $t\colon D \to \Sigma$, where $D \in precl(\mathbb{N})$. The *empty tree*, denoted t_ε, is the unique tree such that $dom(t) = \emptyset$. We henceforth refer to $dom(t)$ as the *nodes of* t and write $nodes(t)$ rather than $dom(t)$. The size of a tree $t \in T_\Sigma$, denoted $size(t)$, is $|nodes(t)|$. The height of t, denoted $height(t)$, is $1 + \max(n \mid \alpha_1 \cdots \alpha_n \in nodes(t))$.

For a tree $t \in T_\Sigma$ and a node $v \in nodes(t)$, the *subtree of* t *rooted at* v is denoted by t/v. It is defined by $nodes(t/v) = \{v' \in \mathbb{N}^* \mid vv' \in nodes(t)\}$ and, for all $v' \in nodes(t/v)$, $(t/v)(v') = t(vv')$. The *leaves* of t is the set $leaves(t) = \{v \in \mathbb{N}^* \mid \nexists i \in \mathbb{N}\ s.t.\ vi \in nodes(t)\}$. The *substitution* of t' into t at node v is denoted $t[\![v \leftarrow t']\!]$. It is defined by

$$nodes(t[\![v \leftarrow t']\!]) = (nodes(t) \setminus \{vu \mid u \in \mathbb{N}^*\}) \cup \{vu \mid u \in nodes(t')\};$$

and, for every $u \in nodes(t[\![v \leftarrow t']\!])$, if $u = vv'$ for some $v' \in nodes(t')$ then $t[\![v \leftarrow t']\!](u) = t'(v')$, otherwise $t[\![v \leftarrow t']\!](u) = t(u)$.

For a tree $t \in T_\Sigma$ let $v_1, \ldots, v_k \in nodes(t)$ be the immediate child nodes of the root ordered by numeric value. That is, $\{v_1, \ldots, v_k\} = \{v \in nodes(t) \mid |v| = 1\}$, ordered such that $v_i < v_{i+1}$ for all $i \in [k-1]$. Then we will write t as $f[t_1, \ldots, t_k]$, where $f = t(\varepsilon)$ and $t_j = t/v_j$ for all $j \in [k]$. In the special case where $k = 0$ (i.e., when $nodes(t) = \{\varepsilon\}$), the brackets may be omitted, thus denoting t as f.

Shuffle operations and shuffle expressions. We recall the definitions of the operations shuffle and shuffle closure, and of shuffle expressions, from [23,31].

The *shuffle* operation $\odot : \Sigma^* \times \Sigma^* \to pow(\Sigma^*)$ is inductively defined as follows: for every $u \in \Sigma^*$ it is given by $u \odot \varepsilon = \varepsilon \odot u = \{u\}$, and by

$$\alpha_1 u_1 \odot \alpha_2 u_2 = \{\alpha_1 w \mid w \in (u_1 \odot \alpha_2 u_2)\} \cup \{\alpha_2 w \mid w \in (\alpha_1 u_1 \odot u_2)\},$$

for every $\alpha_1, \alpha_2 \in \Sigma$, and $u_1, u_2 \in \Sigma^*$. The operation extends to languages with

$$\mathcal{L}_1 \odot \mathcal{L}_2 = \bigcup_{u_1 \in \mathcal{L}_1, u_2 \in \mathcal{L}_2} u_1 \odot u_2.$$

The *shuffle closure* of a language $\mathcal{L} \in \Sigma^*$, denoted $\mathcal{L}^{\odot}$, is

$$\mathcal{L}^{\odot} = \bigcup_{i=0}^{\infty} \mathcal{L}^{\odot i}, \quad \text{where } \mathcal{L}^{\odot 0} = \{\varepsilon\} \text{ and } \mathcal{L}^{\odot i} = \mathcal{L} \odot \mathcal{L}^{\odot i-1}.$$

Shuffle expressions are regular expressions that can additionally use the shuffle operators. The shuffle expressions over the alphabet Σ are as follows. The empty string ε, the empty set $\emptyset$, and every $\alpha \in \Sigma$ is a shuffle expression. If s_1 and s_2 are shuffle expressions, then so are $(s_1 \cdot s_2)$, $(s_1 + s_2)$, $(s_1 \odot s_2)$, s_1^*, and $s_1^{\odot}$. Shuffle expressions that do not use the shuffle closure operator are said to be *closure free*. The language $\mathcal{L}(s)$ of a shuffle expression s is defined in the usual way. *Shuffle languages* are the languages defined by shuffle expressions.

3. Concurrent finite-state automata

In this section, we introduce *concurrent finite-state automata* (CFSA). They are inspired by *recursive Markov models*, but differs from these in two aspects: the global state space is not partitioned into component automata and, more importantly, they differ in that recursive calls can be made in parallel. The latter feature allows for an unbounded number of invocations to be executed simultaneously, but each symbol can only be read by one invocation. In Definition 1, the string $p^{\odot}$ is to be read as single symbol. In the later definition of CFSA semantics, transitions of the form $(q, \alpha, q'[p^{\odot}])$ will be interpreted as rule schema.

Definition 1 (*CFSA*)**.** A *Concurrent FSA* is a tuple $M = (Q, \Sigma, \delta, I)$, where

- Q is a finite set of *states*;
- Σ is an alphabet of *input symbols*;
- $\delta \subseteq Q \times \Sigma_{\varepsilon} \times T$ is a set of *transitions*, where T is the finite set

  $$\{q, q[p], q[p, p'], q[p^{\odot}] \mid q, p, p' \in Q\} \cup \{t_{\varepsilon}\}.$$

  A transition $(q, \alpha, t) \in \delta$ is
  - *terminal* if $|nodes(t)| = 0$, in this case it must also hold that $\alpha = \varepsilon$,
  - *horizontal* if $|nodes(t)| = 1$, and
  - *vertical* if $|nodes(t)| > 1$.
- $I \subseteq Q$ is a set of *initial* states. □

We now establish the semantics of CFSA. Whereas a FSA is in a single state at a time, a concurrent FSA maintains a branching call-stack of states, represented by an unranked tree over an alphabet of states. In each step, exactly one leaf node of the state tree is rewritten. Vertical transitions model the invocation of child processes; horizontal transitions the continued execution within a process; and terminal transitions the completion of a process. A CFSA accepts a string if, upon reading the entire string, it can reach a configuration in which every processes has been completed, i.e., the state tree is empty.

Definition 2 (*Concurrent FSA Semantics*)**.** A *configuration* of the CFSA $M = (Q, \Sigma, \delta, I)$ is a tuple $(w, t) \in \Sigma^* \times T_Q$. The set of all configurations of M is denoted $\Delta(M)$. A configuration $(w, t) \in \Delta(M)$ is *initial* (with respect to the string $w \in \Sigma^*$) if $t \in I$.

Consider the configurations $(w, t), (w', t') \in \Delta(M)$. There is a *transition step* from (w, t) to (w', t'), written $(w, t) \to (w', t')$, if there is a transition $(q, \alpha, s) \in \delta$ and a node $v \in nodes(t)$ such that $w = \alpha w'$, $t/v = q$ (so v is a leaf), and either

- $s \in T_Q$ and $t' = t[\![v \leftarrow s]\!]$, or
- $s = p'[p^{\odot}]$ and $t' = t[\![v \leftarrow p'[\underbrace{p, \ldots, p}_{n}]]\!]$ for some for $p, p' \in Q$ and $n \in \mathbb{N}$.

The reflexive and transitive closure of $\to$ is denoted $\xrightarrow{*}$. The *language recognized by* M is $\mathcal{L}(M) = \{w \in \Sigma^* \mid \exists q \in I : (w, q) \xrightarrow{*} (\varepsilon, t_{\varepsilon})\}$. □

For the sake of brevity only the state-tree part of a configuration, called a *configuration tree*, may be shown in cases where the string is irrelevant.

Remark. For simplicity we will assume that the *terminal* transitions of a CFSA form a subset of $Q \times \{\varepsilon\} \times \{t_{\varepsilon}\}$, that is, we assume that terminal transitions do not read symbols. This causes no loss of generality with respect to the recognized string language, since a CFSA can be rewritten to fulfill this requirement in linear time by adding a designated terminal state q (the only transition for q is $(q, \varepsilon, t_{\varepsilon})$), and change all other terminal rules $(q', \alpha, t_{\varepsilon})$ into the horizontal rule (q', α, q).

Example 1. Recall that a Dyck language consists of all well-balanced strings over a given set of parentheses. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be the Dyck languages over the symbol pairs $\lfloor, \rfloor$ and $\lceil, \rceil$, respectively. Their shuffle $\mathcal{L} = \mathcal{L}_1 \odot \mathcal{L}_2$ is recognized by the concurrent FSA $M = (\{q_0, q_0', q_1, q_1', q_2, q_2'\}, \{\lfloor, \rfloor, \lceil, \rceil\}, \delta, \{q_0\})$, where

$$\delta = \{(q_0, \varepsilon, q_0'[q_1, q_2]),\ (q_0', \varepsilon, t_{\varepsilon}),\ (q_1, \lfloor, q_1'[q_1]),\ (q_1', \rfloor, q_1),\ \ (q_1, \varepsilon, t_{\varepsilon}),\ (q_2, \lceil, q_2'[q_2]),\ (q_2', \rceil, q_2),\ (q_2, \varepsilon, t_{\varepsilon})\}.$$

To illustrate the automaton's semantics, we step through an accepting run of M on the string $w = \lfloor\lfloor\lceil\rfloor\lfloor\rceil\rfloor\rfloor$ (see Fig. 1). Note that since $w \in w_1 \odot w_2$ for $w_1 = \lfloor\lfloor\rfloor\lfloor\rfloor\rfloor \in \mathcal{L}_1$ and $w_1 = \lceil\rceil \in \mathcal{L}_2$, it follows that $w \in \mathcal{L}_1 \odot \mathcal{L}_2$. □

It is known that $\mathcal{L}_1 = \{a^n b^n \mid n \in \mathbb{N}\}$ is a context-free language, but it is not a shuffle language. Conversely, $\mathcal{L}_2 = \{w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c\}$ is a shuffle language but is not context-free. Both $\mathcal{L}_1$ and $\mathcal{L}_2$ is recognized by a CFSA, and so is $\mathcal{L}_1 \cup \mathcal{L}_2$, which is neither a context-free nor a shuffle language. As will be shown below, the CFSA languages properly extend both the context-free languages and the shuffle languages. They also have comparatively nice closure properties.

Theorem 1. *The languages recognized by CFSA are closed under union, concatenation, Kleene star, shuffle and shuffle closure. They are not closed under intersection with a regular language or complementation.*

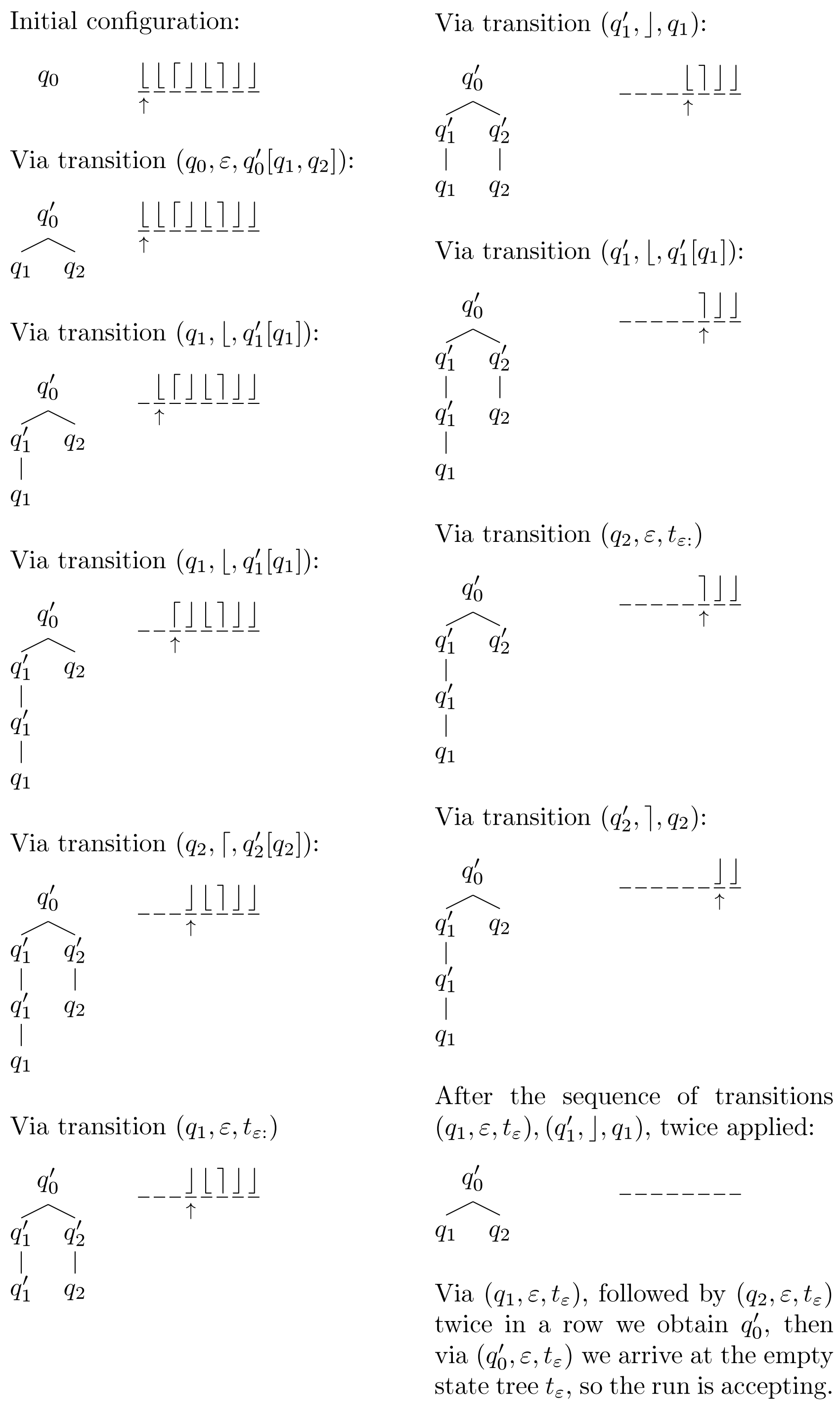

Initial configuration:
Via transition $(q_0, \varepsilon, q_0'[q_1, q_2])$:
Via transition $(q_1, \lfloor, q_1'[q_1])$:
Via transition $(q_1, \lfloor, q_1'[q_1])$:
Via transition $(q_2, \lceil, q_2'[q_2])$:
Via transition $(q_1, \varepsilon, t_{\varepsilon:})$
Via transition $(q_1', \rfloor, q_1)$:
Via transition $(q_1', \lfloor, q_1'[q_1])$:
Via transition $(q_2, \varepsilon, t_{\varepsilon:})$
Via transition $(q_2', \rceil, q_2)$:
After the sequence of transitions $(q_1, \varepsilon, t_\varepsilon), (q_1', \rfloor, q_1)$, twice applied:
Via $(q_1, \varepsilon, t_\varepsilon)$, followed by $(q_2, \varepsilon, t_\varepsilon)$ twice in a row we obtain q_0', then via $(q_0', \varepsilon, t_\varepsilon)$ we arrive at the empty state tree t_ε, so the run is accepting.

Fig. 1. An accepting run of the CFSA M on input $\lfloor\lfloor\lceil\rfloor\lfloor\rceil\rfloor\rfloor$.

Proof. Let $M = (Q, \Sigma, \delta, I)$ and $M' = (Q', \Sigma, \delta', I')$ be CFSA. We assume without loss of generality that $Q \cap Q' = \emptyset$, and that the automata have only one initial state each, i.e., $I = \{q_0\}$ and $I' = \{q_0'\}$. The latter assumption can be made without loss of recognizing power since ε-transitions are allowed.

Union. A CFSA for the union of M and M' can be constructed by adding a new initial state q together with ε-transitions from q to each of q_0 and q_0'.

Concatenation. A CFSA for the concatenation of M with M' can be constructed by adding a new initial state q, new states q' and q'', and the transitions $(q, \varepsilon, q'[q_0])$, $(q', \varepsilon, q''[q'_0])$, and $(q'', \varepsilon, t_\varepsilon)$. This allows the automaton to first simulate a run of M and then a run of M'.

Kleene closure. A CFSA for the Kleene closure of M can be constructed by adding a new initial state q and the transitions $(q, \varepsilon, t_\varepsilon)$ and $(q, \varepsilon, q[q_0])$. This allows the automaton to simulate any number of runs of M, one after the other.

Shuffle. For the shuffle of $\mathcal{L}(M)$ and $\mathcal{L}(M')$ we add states q, q', where q becomes the unique initial state of the new automaton. We also add the vertical transition $(q, \varepsilon, q'[q_0, q'_0])$ and the terminal transition $(q', \varepsilon, t_\varepsilon)$.

Shuffle closure. To construct the shuffle closure of the language of M, we again add states q, q', where q becomes the unique initial state of the new automaton. Additionally, we add the vertical transition $(q, \varepsilon, q'[q_0{}^{\odot}])$ and the terminal transition $(q', \varepsilon, t_\varepsilon)$. This allows the new automaton to spawn any number of copies of M that can then run in parallel over the input string.

Intersection. Consider the languages $\mathcal{L}_1 = (abc)^{\odot}$ and $\mathcal{L}_2 = a^*b^*c^*$. The former is a shuffle language, and the latter clearly a regular language, so both are recognizable by CFSA. As we shall see, their intersection $\mathcal{L} = \{a^nb^nc^n \mid n \in \mathbb{N}\}$ is not. The proof is by contradiction, so let us assume that $\mathcal{L}$ is recognized by some CFSA $M = (Q, \Sigma, \delta, I)$.

To make the upcoming argument clearer, we introduce some convenient definitions. For every $q \in Q$, M_q denotes the CFSA $(Q, \Sigma, \delta, \{q\})$. The *substrings* of a language $\mathcal{L}$, written *substring*$(\mathcal{L})$, is the set $\{v \mid uvw \in \mathcal{L}$ for some $u, w \in \Sigma^*\}$.

Now, if a transition r of the form $(q, \alpha, q'[p, p']) \in \delta$ is applied in an accepting run of M, then $\mathcal{L}(M_p) \odot \mathcal{L}(M_{p'}) \subseteq$ *substrings*$(\mathcal{L})$. For this reason, $\mathcal{L}(M_p) \cup \mathcal{L}(M_{p'}) \subseteq \alpha^*$ for some $\alpha \in \{a, b, c\}$. Otherwise, if for example $w \in \mathcal{L}(M_p)$ and $w' \in \mathcal{L}(M_{p'})$ with $|w|_a > 0$ and $|w'|_b > 0$, the string $w'w \in w \odot w'$ would be in *substrings*$(\mathcal{L})$, but this is impossible since a b occurs before an a in $w'w$. It follows that the order of p and p' in r is irrelevant. Hence, r can equivalently be replaced by a pair of transitions such that $(\varepsilon, q) \overset{*}{\to} (\alpha, q'[p[p']])$. The same argument justifies the replacement of transitions of the form $(q, \alpha, q'[p^{\odot}])$ with transitions that yield $(\varepsilon, q) \overset{*}{\to} (\alpha, q'[p[p[\dots[p]]]])$.

After this language-preserving normalization, the resulting CFSA only generates monadic configuration trees, which means that no shuffling is done. However, without shuffle operations, $\mathcal{L}(M)$ is a context-free language (cf. Theorem 2), and it is well known that $\mathcal{L}$ is not a context-free language. Consequently, $\mathcal{L}$ is not recognizable by a CFSA.

Complementation. Since the CFSA languages are closed under union, but not under intersection, they are not closed under complementation either, since $(L_1 \cap L_2)$ can be expressed as $\overline{(\overline{L_1} \cup \overline{L_2})}$. □

Restrictions and expressive power. We introduce CFSA to provide an automaton model that can be syntactically restricted to capture the combination of shuffle operations with some well-known languages classes. The restrictions considered here are as follows. A CFSA $M = (Q, \Sigma, \delta, I)$ is:

- *horizontal* if δ contains no vertical transitions;
- *non-branching* if every vertical transition is in $Q \times \Sigma \times \{q'[q] \mid q, q' \in Q\}$;
- *finitely branching* if no vertical transition is in $Q \times \Sigma \times \{q'[q^{\odot}] \mid q, q' \in Q\}$;
- *acyclic* if there is no configuration $(w, t) \in \Delta(M)$ and state $q \in Q$ such that q appears twice on a path from the root of t to a leaf.

Theorem 2. *A language is:*

- *regular if and only if it is recognized by a horizontal CFSA;*
- *context-free if and only if it is recognized by a non-branching CFSA;*
- *a shuffle language if and only if it is recognized by an acyclic CFSA;*
- *a closure-free shuffle language if and only if it is recognized by an acyclic and finitely branching CFSA.*

Proof sketch. Horizontal CFSA are equivalent to nondeterministic finite automata in that they recognize the regular languages.

It is easy to turn a context-free grammar $G = (N, \Sigma, \gamma, S)$ in Chomsky normal form into a non-branching CFSA $M = (Q, \Sigma, \delta, I)$. Let $Q = N \cup \{\overline{q} \mid q \in N\}$, $I = \{S\}$, and define δ as follows.

- For every rule $q \to \alpha$ in γ, where $\alpha \in \Sigma_\varepsilon$, there is a horizontal transition $(q, \alpha, \overline{q})$ and a terminal transition $(\overline{q}, \varepsilon, t_\varepsilon)$ in δ.
- For every rule $q \to pp'$ in γ, there is a transition $(q, \varepsilon, p'[p])$ in δ_2.

For the opposite direction, it is equally easy to turn a non-branching CFSA into a language-equivalent push-down automaton.

Next, we show that acyclic CFSA correspond to the shuffle languages. The only-if direction follows directly from the proof of Theorem 1 since the constructions there preserve automata acyclicity.

Given an acyclic CFSA $M = (Q, \Sigma, \delta, I)$ we show how to construct a shuffle expression s recognizing $\mathcal{L}(M)$. Two states $q, q' \in Q$ are said to be *connected* if there is a transition $(q, \alpha, t) \in \delta$, where the label of the root of t is q', for some $\alpha \in \Sigma_\varepsilon$. With this notion of connectivity, let $C_1, \dots, C_k$ be the connected components of M. Consider the directed graph

$G_M = (C_1, \ldots, C_k, E)$, where $(C_i, C_j) \in E$ if there is a state $q \in C_i$, a *vertical* transition $(q, \alpha, t) \in \delta$, and a state $p \in C_j$ such that p (or $p^{\odot}$) labels a leaf of t. Since M is acyclic, also G_M is acyclic.

Let $\delta_v \subseteq \delta$ be the set of all vertical transitions. We create an alphabet Σ_v with one unique new symbol for each vertical transition. Let $h : \delta_v \to \Sigma_v$ be the bijection mapping each $d \in \delta_v$ to the corresponding alphabet symbol. Also, for each $d \in \delta_v$, let q_d be a new state. Define H to be the CFSA obtained from M by replacing each vertical transition $d = (q, \alpha, q'[\ldots])$ with the horizontal transitions (q, α, q_d) and $(q_d, h(d), q')$. Notice that the connected components of H are the same as the connected components of M and that H is a finite automaton recognizing a regular language.

For each $q \in Q$, let the *regular* expression $r(q)$ be such that $\mathcal{L}(r(q)) = \mathcal{L}(H_q)$, that is, $r(q)$ describes the language that H recognizes when starting from state q. Such a regular expression can be computed using standard constructions.

We are now ready to describe how to construct the shuffle expression corresponding to M. To be precise, for each state $q \in Q$, we will define a shuffle expression $s(q)$ such that the language of $s(q)$ is the language of M_q, in other words, the CFSA obtained from M by replacing I by $\{q\}$. We do this by induction on the structure of G_M.

If C is a leaf of G_M, then there are no vertical transitions in the connected component C. Hence, for every $q \in C$, we have $s(q) = r(q)$.

Suppose that q belongs to a connected component C_i such that for all states in all components reachable from C_i in G_M, we have already computed the corresponding shuffle expressions. In this case we get the shuffle expression for q by taking $r(q)$ and replacing symbols in Σ_v by appropriate shuffle expressions. In particular, consider symbol $h(d) \in \Sigma_v$ that corresponds to $d = (q', \alpha, t) \in \delta_v$. The shuffle expression for $h(d)$ is obtained from t as follows.

- If $t = p[p']$, for some $p, p' \in Q$, then the shuffle expression is $s(p')$.
- If $t = p[p'_1, p'_2]$ then the shuffle expression is $s(p'_1) \odot s(p'_2)$.
- If $t = p[p'^{\odot}]$, then the shuffle expression is $(s(p'))^{\odot}$.

The shuffle expression for M is the union of those for the states in I, i.e.,

$$s = \bigcup_{q \in I} s(q).$$

The equivalence $\mathcal{L}(M) = \mathcal{L}(s)$ can be shown by a standard induction.

Finally, that acyclic and finitely branching CFSA correspond to the closure free shuffle languages follows from the constructions in the proof of Theorem 1 as only the shuffle closure operator induces unbounded branching. □

Since the closure free shuffle languages are regular [21], we can conclude that acyclic and finitely branching CFSA also recognize the regular languages.

To see that CFSA do not provide us with the full power of linear bounded Turing machines, we first note that they can be augmented in polynomial time with "shortcuts", that is, contractions of ε-consuming transition sequences into single transitions.

Definition 3 (*ε-Efficient*)**.** A CFSA $M = (Q, \Sigma, \delta, I)$ is *ε-efficient* if it fulfills the following conditions:

1. For every $q \in Q$, if $(\varepsilon, q) \xrightarrow{*} (\varepsilon, t_\varepsilon)$ then $(\varepsilon, q) \to (\varepsilon, t_\varepsilon)$.
2. For every choice of $q, q' \in Q$, if $(\varepsilon, q) \xrightarrow{*} (\varepsilon, q')$ then $(\varepsilon, q) \to (\varepsilon, q')$.
3. For every choice of $q, q', p, p' \in Q$, if $(\varepsilon, q) \xrightarrow{*} (\varepsilon, q'[p, p']) \xrightarrow{*} (\varepsilon, q'[p])$ then $(\varepsilon, q) \to (\varepsilon, q'[p])$.

Lemma 1. *Every CFSA $M = (Q, \Sigma, \delta, I)$ can be rewritten into a language-equivalent ε-efficient CFSA in polynomial time.*

Proof (*Sketch*)**.** A simple procedure based on the emptiness test (see Theorem 4) suffices to add any missing transition to M in polynomial time. For example, construct the automaton $M' = (Q, \Sigma, \delta', \{q\})$ where $\delta' \subseteq \delta$ contains only the transitions that do not consume any symbol. Then $(\varepsilon, q) \xrightarrow{*} (\varepsilon, t_\varepsilon)$ if and only if M' is nonempty. Once Condition 1 is satisfied, the transitions needed to satisfy the remaining two conditions can be added through similar constructions. □

Lemma 2. *Let $M = (Q, \Sigma, \delta, I)$ be an ε-efficient CFSA. In a sequence of transition steps that accepts the string w and is of minimum length, no intermediary configuration tree needs to have more than $|w|$ leaves or be of height greater than $|Q|(|w|+1)$.*

Proof. Let $c = (w, t)$ and $c' = (w, t')$ be a pair of configurations in $\Delta(M)$. If there is a sequence of ε-transitions from c to c', then there is also a sequence of length at most $n \leq size(t) + 2size(t')$. Such a short sequence can be found by organizing the transitions as follows: $(w, t) \xrightarrow{*} (w, \hat{t}) \xrightarrow{*} (w, t')$ where the $t \to \hat{t}$ part of the derivation *only deletes nodes*, and the $\hat{t} \to t'$ part *never deletes nodes*. This reorganization is possible since M is ε-efficient, so all possible node deletions/relabelings can be performed without generating extraneous nodes. In turn, this means that no node needs to be generated only to subsequently be deleted. It follows that at most $|nodes(t)|$ may need to be deleted, and at most $|nodes(t')|$ nodes may need to be created and/or relabeled with a new state.

Consider an accepting sequence of transitions of minimal length. Only $|w|$ symbols are consumed by the transitions, so if there are $|w| + 1$ leaves in any intermediate configuration tree, then one of them must consume ε. The existence of such a leaf violates the assumption that the sequence is of minimal length (notice that conditions 1–3 in Definition 3 ensure that useless nodes never have to be added). The height bound holds since a higher tree would have $|w|+2$ or more copies of some state q along some path. With $|w|+2$ instances of q-labeled nodes, there are $|w|+1$ such q-delimited sections on the path. Only $|w|$ symbols are consumed, so one of those sections will be matched up against the empty string. The redundant section could be omitted without affecting the accepted string, which violates the assumption that the original sequence was of minimum length. □

Theorem 3. *The languages recognized by CFSA are properly contained in the context-sensitive languages.*

Proof. Let $M = (Q, \Sigma, \delta, I)$ be a CFSA and w string. If there is an accepting run of M on w from an initial state q_0, then a nondeterministic Turing machine can guess and verify this run in linear space by a depth-first left-to-right search.

The TM simulates a run of M on w starting from q_0, but when a vertical transition $(q, \alpha, q'[s])$ is used, where s is a sequence of labels, the TM guesses what prefix w' of the subsequent string is to be consumed by the state trees derived from s, and calls itself recursively with w' and s as arguments. If the recursive call succeeds, it goes on to verify the remainder of w from q'.

Let w' and s be such a prefix and sequence of labels. If s is a single state p, the TM recursively verifies that w' is accepted by M when starting from state p, i.e., $w' \in \mathcal{L}(M_p)$. If s is a pair $p, p' \in Q$, the TM guesses a way to partition w' into subsequences u, u' so that $w' \in u \odot u'$. It then recursively verifies that $u \in \mathcal{L}(M_p)$, and if that is the case, that $u' \in \mathcal{L}(M_{p'})$. Finally, if $s = p^{\odot}$, the TM guesses a non-empty subsequence of w', verifies recursively that this subsequence belongs to $\mathcal{L}(M_p)$, and if so, verifies recursively that the remainder of w' (if it is non-empty) can be accepted from the sequence of labels $p^{\odot}$.

We note that as the TM explores a candidate run top-down, it need only remember a sequence of labels s and a partitioning of the argument string at each level in the call stack. By Lemma 1, the CFSA M can be assumed to be ε-efficient, so by Lemma 2, the height of the call stack can be restricted to $|w|+1$. The information about where in the string partitions end can easily be maintained in linear space, for example as a bit string where the ith zero signifies the ith symbol in w, and the jth one signifies the end of the partition for the jth object on the current call stack. If follows that the information maintained during any step of the simulated run is linear in $|w|$, so the non-uniform membership problem for CFSA languages can be decided by a linearly bounded nondeterministic TM. As shown in the proof of Theorem 1, no CFSA recognizes the language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, so the CFSA languages form a proper subset of the context-sensitive languages. □

Since not all CFSA-languages are context-free (e.g., there are non-context-free shuffle languages), we conclude that their expressive powers lies strictly between that of context-free grammars and that of context-sensitive grammars.

Also unlike linear bounded Turing machines, CFSA can be efficiently checked for emptiness.

Theorem 4. *The emptiness problem for CFSA is decidable in polynomial time.*

Proof. Let $M = (Q, \Sigma, \delta, I)$ be a CFSA. A state q of M is *live* if $\mathcal{L}(M_q)$ is nonempty. Let $\mathcal{F} \subseteq Q$ be the smallest set satisfying the following conditions.

1. $F_0 = \{q \mid (q, \varepsilon, t_\varepsilon) \in \delta\}$
2. $F_i \subseteq F_{i+1}$
3. if $(q, \alpha, q') \in \delta$ and $q' \in F_i$ then $q \in F_{i+1}$
4. if $(q, \alpha, q'[p^{\odot}])$ and $q' \in F_i$, for any $p \in Q$, then $q \in F_{i+1}$ (recall that the shuffle closure may generate zero instances of p),
5. if $(q, \alpha, q'[s]) \in \delta$ for some $q' \in F_i$ and some s such that every state that appears in s belongs to F_i, then $q \in F_{i+1}$
6. $\mathcal{F} = \cup_{i=0}^{\infty} F_i$

Claim. A state q of M is live if and only if $q \in \mathcal{F}$.

For the if-direction, we prove by induction on the smallest i such that $q \in F_i$ that q is live. For $i = 0$ this is trivially true, since $(q, \varepsilon, t_\varepsilon) \in \delta$, and thus M_q accepts the string ε.

Assume that every state in F_i is live, and consider the state $q \in F_{i+1} \setminus F_i$. If $(q, \alpha, q') \in \delta$, with $q' \in F_i$, then there is a string w such that $M_{q'}$ accepts w. This means that M_q accepts αw and we conclude that q is live. If there is no such rule, there must be a rule $(q, \alpha, q'[s])$ in δ such that q' and either $s = p^{\odot}$ or every state that appears in s belongs to F_i. If this is the case, then there is a word $w_{q'}$ accepted by $M_{q'}$. If $s = p$, there is a word $w_p \in \mathcal{L}(M_p)$ and conclude that M_q accepts $\alpha \cdot w_p \cdot w_{q'}$. Similarly, if $s = p, p'$ there are strings $w_p \in \mathcal{L}(M_p)$, $w_{p'} \in \mathcal{L}(M_{p'})$, and $w_{p \odot p'} \in w_p \odot w_{p'}$ such that M_q accepts $\alpha \cdot w_{p_1 \odot p_2} \cdot w_{q'}$. Finally, if $s = p^{\odot}$, we know that M_q accepts $\alpha \cdot w_{q'}$. Thus q is live.

For the other direction, assume that q is live as witnessed by some word
$w = \alpha_1 \cdots \alpha_m$ in $\mathcal{L}(M_q)$ with $\alpha_i \in \Sigma \cup \{\varepsilon\}$. Let

$$(w, q) = (w_1, t_1) \to \cdots \to (w_m, t_m) = (\varepsilon, t_\varepsilon)$$

be an accepting sequence of transition steps of M_q on w. We show by induction that every state that appears in $t_1, \ldots, t_m$ is in $\mathcal{F}$. In particular, this means that q belongs to $\mathcal{F}$, because $t_1 = q$. Since $t_m = t_\varepsilon$, all states in t_m belong to $\mathcal{F}$. Assume that all states appearing in t_i belong to $\mathcal{F}$ and consider t_{i-1}. One of the following cases apply (for some leaf node v).

1. $t_{i-1} = t[\![v \leftarrow q]\!]$, $t_i = t[\![v \leftarrow q']\!]$, and there is a transition $(q, \alpha_i, q') \in \delta$. If this is the case, $q \in \mathcal{F}$ and thus all states of t_{i-1} belong to $\mathcal{F}$.
2. $t_{i-1} = t[\![v \leftarrow q]\!]$, $t_i = t[\![v \leftarrow q'[u_1, \ldots, u_n]]\!]$, and there is a transition $(q, \alpha_i, q'[s]) \in \delta$ such that

- $s = p$, $n = 1$, and $u_1 = p$,
- $s = p_1, p_2$, $n = 2$, $u_1 = p_1$ and $u_2 = p_2$, or
- $s = p^{\odot}$ and $u_1 = \cdots = u_n = p$.

  In either case, $q \in \mathcal{F}$ and thus all states of t_{i-1} belong to $\mathcal{F}$.
3. $t_{i-1} = t[\![v \leftarrow q]\!]$, $t_i = t[\![v \leftarrow t_\varepsilon]\!]$. In this case, q belongs to F_0 and we can conclude that all states appearing in t_{i-1} belong to $\mathcal{F}$.

The set $\mathcal{F}$ can be computed in polynomial time and $\mathcal{L}(M)$ is empty if and only if $\mathcal{F} \cap I = \emptyset$. Thus emptiness for CFSA can be decided in polynomial time. □

4. Membership problems

4.1. The membership problem for unrestricted CFSA

The membership problem for unrestricted CFSA is intractable, both in the uniform and the non-uniform case.

Theorem 5. *Both the uniform and the non-uniform membership problem for CFSA is NP-complete.*

Proof. NP-hardness for the uniform membership problem for shuffle expressions is already known; see, e.g., [44,2,35]. The non-uniform membership problem is also NP-complete, this follows from both [40] and Corollary 4 in this paper. Corollary 4 states that the non-uniform membership problem for the shuffle of two deterministic linear context-free languages is NP-hard. Theorem 2 says that CFSA can represent all context-free languages and Theorem 1 that they are closed under the shuffle operation, so a CFSA can be constructed to represent the shuffle of context-free languages, establishing NP-hardness.

Demonstrating that the membership problem for CFSA is *in* NP is deferred to Lemma 3 below. □

Lemma 3. *Given a CFSA $M = (Q, \Sigma, \delta, I)$ and a string $w \in \Sigma^*$ it is possible to determine if $w \in \mathcal{L}(M)$ in nondeterministic polynomial time.*

Proof sketch. Due to Lemma 1, we may assume that M is ε-efficient. We show that there is a polynomial P such that for every $w \in \mathcal{L}(M)$, there is a state $q_0 \in Q$ and a sequence of transition steps

$$(w, q_0) = (w_1, t_1) \rightarrow \cdots \rightarrow (w_n, t_n) = (\varepsilon, t_\varepsilon)$$

such that $n \leq P(|Q| + |w|)$. This result allows an accepting sequence of transition steps to be "guessed" as part of a nondeterministic polynomial-time decision algorithm for the membership problem.

It follows from Lemma 2 that the size of the configuration trees necessary to accept an input string w is bounded by $|w|^2|Q|$, and any sequence of transitions on polynomially sized trees can be limited to a polynomial number of steps. There is thus, for every $w \in \mathcal{L}(M)$, a sequence of polynomial length, which means that a nondeterministic algorithm can check membership by guessing the sequence. □

4.2. The membership problem for acyclic CFSA

We now turn to the membership problem for acyclic CFSA, i.e., the restriction of CFSA that recognizes the shuffle languages.

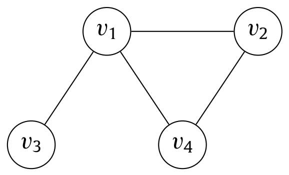
Corollary 1. *For acyclic CFSA*

1. *the non-uniform membership problem is solvable in polynomial time, and*
2. *the uniform membership problem is NP-complete.*

Proof. The result for non-uniform membership follows directly from Theorem 2 and the fact, proved in [31], that non-uniform membership for shuffle expressions is polynomial. For the uniform membership problem, membership in NP follows from Theorem 5. NP-hardness follows by an easy adaptation of a result by Barton on the complexity of ID/LP parsing [2]. □

The uniform membership problem is NP-complete already for acyclic and finitely branching CFSA, which only recognize regular languages. This is not too surprising since the similar NFA(&) employed by Gelade et al. [20], which also recognize the regular languages, has PSPACE-complete uniform membership. For some languages, CFSA offer a more succinct form of representation than NFA and the shuffle automata from [31]. One example is the language family $\{\{a^n\} \mid n \in \mathbb{N}\}$, for which the smallest NFAs and shuffle automata have sizes linear in n, while the smallest CFSAs are logarithmic in n.

Corollary 1 states that the membership problem is polynomial for a fixed automaton but NP-hard if the automaton is considered as part of the input. The question then remains whether the size of the automaton merely influences the coefficients of the polynomial or if it affects the degree itself. We give a partial answer by showing that when parameterized by the maximal size of a configuration tree for the automaton, the uniform membership problem for acyclic and finitely branching CFSAs is *not fixed-parameter tractable*, unless FPT = W[1]. This class equivalence is considered very unlikely and would have far-reaching complexity-theoretic implications. For more on parameterized complexity theory, see, e.g., [15,17].



$$w = v_1v_1v_1v_2v_2v_2v_3v_3v_3v_4v_4v_4e_{1,2}e_{1,3}e_{1,4}e_{2,4}$$

$$s = (v_1v_1v_1 + v_2v_2v_2 + v_3v_3v_3 + v_4v_4v_4)$$

$$t = \Sigma^*$$

$$u = v_1v_2e_{1,2} + v_1v_3e_{1,3} + v_1v_4e_{1,4} + v_2v_4e_{2,4}$$

Fig. 2. A graph together with the corresponding input word w and the regular expressions s, t, and u, given $k = 3$.

We state the result for acyclic and finitely branching CFSA, but it could be equivalently stated for closure-free shuffle expressions. We first define the parameterized version of the problem.

Definition 4. An instance of the parameterized uniform membership problem for acyclic and finitely branching CFSA is a pair (M, w) where M is an acyclic and finitely branching CFSA over a finite alphabet Σ and w is a string in Σ^*. The parameter is the maximal size of any configuration tree for M. The question is whether $w \in \mathcal{L}(M)$. □

For acyclic and finitely branching CFSA, the maximal size of the configuration trees depends only on the automaton. If the membership problem for these automata was fixed-parameter tractable, it would have an algorithm with running time $f(k) \cdot n^c$, where f is a computable function, k is the parameter (the maximal tree size), n is the instance size, and c is a constant. Theorem 6 gives strong evidence to the contrary.

Theorem 6. *The parameterized uniform membership problem for acyclic and finitely branching CFSA is W[1]-hard.*

The proof is by a fixed-parameter reduction from parameterized clique, which is known to be $W[1]$-complete [15].

Definition 5. An instance of k-Clique is a pair (G, k), where $G = (V, E)$ is an undirected graph and k is an integer. The question is whether there is a set $C \subseteq V$ of size k such that the subgraph of G induced by C is complete. The parameter is k. □

Proof. The proof consists in a reduction from k-Clique to the membership problem at hand. Let $(G = (V, E), k)$ be an instance of k-Clique, and let $n = |V|$ and $m = |E|$. We construct an alphabet Σ, a shuffle expression r, and a string $w \in \Sigma^*$ such that $|\Sigma| = O(n + m)$, $|r| = O(k \cdot n^2 + k^2 \cdot m)$, $|w| = O(k \cdot n + m)$, the shuffle operator appears $O(k^2)$ times in r, and $w \in \mathcal{L}(r)$ if and only if G has a clique of size k. To construct Σ, we assume that the vertices in V are named $v_1, v_2, \ldots, v_n$ and that the edges are named $e_{i,j}$ where $i < j$ are the numbers of the two incident vertices and let $\Sigma = V \cup E$. The word w is $v_1^k \cdot v_2^k \cdots v_n^k \cdot$ *edges*, where *edges* is any enumeration of the edges in E.

We define the regular languages s, t, u by

- $s = (v_1^k + v_2^k + \cdots + v_n^k)^{n-k}$,
- $t = \Sigma^*$, and
- $u = \Sigma_{e_{i,j} \in E}(v_i \cdot v_j \cdot e_{i,j})$.

Finally, we define

$$r = s \odot t \odot \left(\bigodot_{i=1}^{k(k-1)/2} u \right).$$

A graph, together with the expressions and the input string resulting from the reduction with $k = 3$ is shown in Fig. 2. The intuition behind the reduction is as follows:

- The expression s matches $n - k$ sequences of k copies of a vertex name. This leaves only k such sequences in w for the rest of r to match against, so the remainder of the expression can only use k distinct vertex names. In the example shown in Fig. 2, we have $n = 4$ and $k = 3$. Thus s matches exactly one group of three identical vertex names.
- Each instance of expression u matches one sequence $v_i \cdot v_j \cdot e_{i,j}$. Thus, the $k(k-1)/2$ instances of u match against $k(k-1)$ vertex names and $k(k-1)/2$ edge names. Due to the matching of s, the $k(k-1)$ vertex names can only be chosen from among k vertices. Thus the $k(k-1)/2$ edge names, which are distinct since *edges* is an enumeration of E, represent edges that have both their endpoints in a set of vertices of size k. In the example from Fig. 2, we have $k = 3$ and thus $3(3-1)/2 = 3$ copies of u are used. Since s matches one group of vertex names, these three copies of u can only be matched against a total of three distinct vertex names.
- The expression t matches all remaining vertex and edge names.
- Any graph that has $k(k-1)/2$ distinct edges whose endpoints are all in a set of vertices of size k has a clique of size k.

Thus w belongs to $\mathcal{L}(r)$ if and only if G has a clique of size k. Notice that $|r|$ is polynomial in $|G|$ and that the number of shuffle operators depends only on k.

Using Theorem 2 it is easy to find an acyclic and finitely branching CFSA M_r such that $\mathcal{L}(M_r) = \mathcal{L}(r)$, the size of M_r is polynomial in the size of G, and the maximum size of a configuration tree for M_r is $O(k^2)$. Thus there is a fixed-parameter reduction from k-Clique to parameterized membership for acyclic and finitely branching CFSA, so the latter problem is W[1]-hard. □

The following corollary is immediate.

Corollary 2. *The uniform membership problem for closure-free shuffle expressions, parameterized by the number of shuffle operators, is W[1]-hard.*

4.3. *The membership problem for* Reg $\odot$ CF *and* Sh $\odot$ CF

We next show that the shuffle of a context-free language and a regular language is efficiently recognizable, even if the language descriptions are considered to be part of the input.

Theorem 7. *The uniform membership problem for the shuffle of two languages, one represented by context-free grammar and one represented by a nondeterministic finite automaton, is solvable in polynomial time.*

Proof (*Sketch*)**.** It is well known that the shuffle of a regular and a context-free language is context-free. It remains to argue that a grammar for the language can be constructed in polynomial time. To achieve this, it is enough to construct a nonterminal (A, q_1, q_2) for every nonterminal A of the input grammar and every pair (q_1, q_2) of states of the input automaton. Working bottom up, it is straightforward to construct the rules of the grammar in such a way that a string w can be produced from (A, q_1, q_2) if and only if there are w_1 and w_2 such that $w = w_1 \odot w_2$, where w_1 can be produced from A in the input grammar and w_2 can take the input automaton from q_1 to q_2. □

Since acyclic and finitely branching CFSA only provide a more compact representation of the regular languages, Theorem 7 extends to the non-uniform membership problem for the shuffle of a context-free language and a closure-free shuffle language:

Corollary 3. *The non-uniform membership problem for the shuffle of two languages, one represented by a context-free grammar and one represented by an acyclic and finitely branching CFSA, is solvable in polynomial time.*

Extending Theorem 7 with techniques inspired by [31], we get the following:

Theorem 8. *The non-uniform membership problem for the shuffle of a shuffle language and a context-free language is solvable in polynomial time.*

Since the languages are not part of the input, we may assume that they are represented by an acyclic CFSA M, and a context-free grammar G, respectively. We prove the above theorem in several steps. First, we show that we can assume that the CFSA for a shuffle language has certain structural properties. Second, we define *simple* configuration trees, and show that any computation of a CFSA for a shuffle language that has the above-mentioned structural properties can be assumed to use only simple configuration trees. Third, we show an upper bound on the number of different simple configuration trees that need to be taken into account during a computation, and provide a compact representation for these. Finally, we prove the theorem, using an extension of the CYK algorithm.

The first structural property of CFSAs that we consider is *stratification*.

Definition 6. An acyclic CFSA $M = (Q, \Sigma, \delta, I)$ is *stratified* if, for every $q \in Q$, there is at most one $p \in Q$ such that, in a configuration tree, a node with label p can be the parent of a node with label q. □

To proceed, we need a canonical translation from shuffle expressions to CFSAs:

Definition 7. Let s be a shuffle expression. Then M_s is the CFSA constructed from s as in the proof of Theorem 1. We call M_s the *canonical* CFSA for s. □

Observation 1. Let s be a shuffle expression, and let $M_s = (Q, \Sigma, \delta, q_0)$ be the canonical CFSA for s. Then M_s has the following properties.

- It is *stratified*.
- It is *acyclic*.
- For each $q \in Q$, there is at most one vertical transition (p, α, t) in δ with q labeling the root of t. We say that an automaton A with this property is *vertically separated*. We write *scp*(A) (for shuffle-closure-parent) for the set of states that can have an unbounded number of children in configuration trees, i.e., $scp(M_s) = \{q \mid \exists p, p', \alpha : (p', \alpha, q[p^{\odot}]) \in \delta\}$. □

Having covered the first step of our proof outline, we continue to introduce and reason about so-called simple configuration trees. For this purpose, we introduce the notions of pruned configuration trees and symmetrically equivalent nodes. Prunings delete subtrees produced through shuffle-closure; a pair of nodes in a configuration tree t are symmetrically equivalent if they are identical modulo an automorphism in a pruned version of t, i.e., when we disregard their exact number of descendant subtrees created through shuffle-closure.

Definition 8 (*Pruning*)**.** Let M_s be the canonical CFSA for a shuffle expression s, let t be a configuration tree of M_s and let v, v' be nodes in t.

We denote by $P(v, v')$ the set of the closest shuffle-closure-parent descendants of v and v'. More formally, let $P(v, v')$ be the set of nodes u of t such that:

1. $t(u) \in scp(M_s)$,
2. u is a descendant of v or v', and
3. there is no node with a label in $scp(M_s)$ on the path from v (or v') to u.

The *pruning* of t with respect to v, v', written $prune(t, v, v')$, is obtained from t by removing all subtrees rooted at children of nodes in $P(v, v')$. □

Definition 9 (*Symmetrical Equivalence*)**.** Let M_s be the canonical CFSA for a shuffle expression s, let t be a configuration tree of M_s and let v, v' be nodes in t. The nodes v and v' are *symmetrically equivalent* if there is an automorphism f on the nodes of $t' = prune(t, v, v')$ such that

- $f(v) = v'$ and $f(v') = v$,
- for every $u \in nodes(t')$, $t'(f(u)) = t'(u)$, and
- for every $u, u' \in nodes(t')$, it holds that $f(u)$ is a child of $f(u')$ if and only if u is a child of u'.

We write $se(v, v')$ if v and v' are symmetrically equivalent.

It is easy to check that symmetrical equivalence is an equivalence relation in the algebraic sense, and thus reflexive, symmetric and transitive. When considering a configuration tree from a computational point of view, we notice that the ordering of its nodes is not important, only its hierarchical structure. It is therefore meaningless to distinguish between symmetrically equivalent nodes in the rewriting process. For our purposes this is an advantage, because we only have to remember to what class of symmetrically equivalent nodes a subtree attaches, not the exact location.

Observation 2. Let $k \in \mathbb{N}$, let t and $s_1, \ldots, s_k$ be configuration trees, let $v_1, \ldots, v_k$ be symmetrically equivalent nodes in $nodes(t)$, and let

$$T = \{t[\![v_1 \leftarrow s_{\phi(1)}, \ldots, v_k \leftarrow s_{\phi(k)}]\!] \mid \phi \text{ is a permutation on } [k]\}.$$

For every $t_1, t_2 \in T$ and $w \in \Sigma^*$, if $(t_1, w) \overset{*}{\rightarrow} (\varepsilon, t_\varepsilon)$ then $(t_2, w) \overset{*}{\rightarrow} (\varepsilon, t_\varepsilon)$. □

Due to Observation 2, it is never useful to apply a transition $r = (p', \alpha, q[p^{\odot}])$ below a node v, when there is a symmetrically equivalent node v' below on which r has already been applied. This claim, which will be proved later on, means that the search space can be reduced to so-called *simple* configuration trees.

Definition 10. A configuration tree t is *simple* if is it does not contain symmetrically equivalent nodes v and v', such that both v and v' have descendants which are labeled by states in $scp(M_s)$ and have children.

A run of a CFSA is *simple* if all configuration trees of the run are simple. □

Lemma 4. *Let M_s the be the canonical CFSA for a shuffle expression s and let w be a word. Then M_s has a simple accepting run on w, if and only if M_s has an accepting run on w.*

Proof sketch. For the "only if" direction we note that every simple accepting run is an accepting run.

For the opposite direction, we provide a rewrite procedure that rearranges the configuration trees in an accepting run into an alternative run that is also accepting. After applying this procedure a finite number of times we are guaranteed to reach a run that is both accepting and simple.

Assume that M_s has an accepting run $\rho = t_0, t_1, \ldots, t_n$ on w, and that ρ is not simple. Let t_i be the first non-simple tree. Then the transition from t_{i-1} to t_i must have been a vertical transition of the form $(p', \varepsilon, q[p^{\odot}])$ that changed the label of some leaf node u from p' to q and gave it a number of children with label p, say m children. Also, there must be an ancestor v of u (possibly, $v = u$) and a node v' such that v and v' are symmetrically equivalent in t_i. Let ϕ be the corresponding automorphism on $prune(t_i, v, v')$. Let $u' = \phi(u)$. If all the children of u were instead children of u', the tree t_i would be a simple configuration tree. And, indeed, because of the vertical separation of M_s, the transition that labeled u' by q must have been $(p', \varepsilon, q[p^{\odot}])$. Thus, it could as well have created m extra children of u' with label p, in addition to the children it originally created. This would not have affected any transitions up to configuration tree t_{i-1}. Symmetrically, the transition from t_{i-1} to t_i might not have created any children at all under u. Thus, with the same sequence of transitions, we could

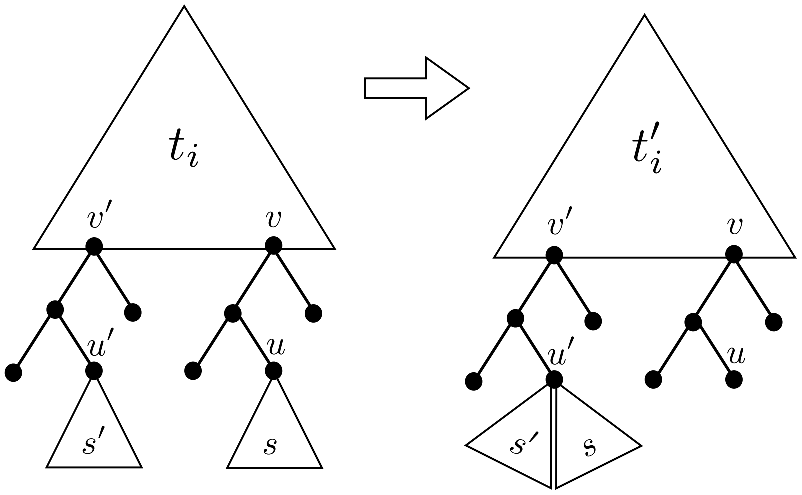


Fig. 3. Children obtained through shuffle-closure can be moved between descendants of symmetrically equivalent nodes.

equally well have ended up with the configuration tree t'_i which is identical to t_i except that u has no children in t'_i and u' has m more p-labeled children than in t_i. Fig. 3 depicts the situation.

It remains to argue that any sequence of transitions used in ρ from t_i forward is also possible from t'_i. Let $j > i$ be the smallest number such that in t_j, either v has no children or v' has no children. We show that the partial run $\rho_{i,j} = t_i, \ldots, t_j$ can be mirrored in a partial run $\rho'_{i,j} = t'_i, \ldots, t'_j$, using the same transitions. If a transition of $\rho_{i,j}$ affects a node in t_k that does not belong to the subtree of v or v', we mirror it directly on t'_k. Now consider a transition from t_k to t_{k+1} that affects a node in a subtree of v or v'. If the same operation is possible on t'_k, we perform it. If not, this can only have two causes.

1. The affected node in t_k is a descendant of u that does not exist in t'_k. In this case, we perform the operation on the corresponding descendant of u'.
2. The affected node in t_k is u' which in t'_k still has children. In this case, we perform the operation on u.

In each of t_i and t_j, we have that exactly one of v and v' is childless. If this is the same node in both trees, they are identical and we are done. If not, we still have to argue that the transitions from t_j forward can be mirrored from t'_j. If not, we use the fact that in t_i and t'_i, v and v' were symmetrically equivalent. Thus we are free to use the automorphism ϕ to reinterpret the sequence $t'_i, \ldots, t'_j$. Under this reinterpretation, t_j and t'_j are identical.

After performing the above operation, all configuration trees up to and *including* t_i are simple. This means that after going through the procedure at most a linear number of times, all configuration trees will be simple. □

Lemma 4 concludes the second step in our proof outline. What remains is to provide a compact representation for simple configuration trees. This makes it necessary to compress the potentially large number of subtrees produced through shuffle closures. Under nodes labeled by states in $scp(M_s)$ we therefore only record which types of subtrees appear, and annotate each of them with a "repetition counter", which encodes the number of times they appear.

Definition 11. Let $M_s = (Q, \Sigma, \delta, q_0)$ be the canonical CFSA for a shuffle expression and let t be a simple configuration tree of M_s. The *compact configuration tree* $\mathrm{cct}(t)$ for t is a tree with nodes labeled by $Q \times \mathbb{N}^*$ where the second component is used as a sequence of counters, one for each direct subtree of the node in question. We define $\mathrm{cct}(t)$ by induction on the structure of t as follows.

- If $t = q$, then $\mathrm{cct}(t) = (q, \langle\rangle)$.
- If $t = q[t_1, \ldots, t_k]$ and $q \in Q \setminus scp(M_s)$, then

$$\mathrm{cct}(t) = (q, \langle\underbrace{1, \ldots, 1}_{k}\rangle)[\mathrm{cct}(t_1), \ldots, \mathrm{cct}(t_k)].$$

  Notice that in this case, k always equals 1 or 2.
- If $t = q[t_1, \ldots, t_k]$ and $q \in scp(M_s)$, then

$$\mathrm{cct}(t) = (q, \langle n_1, \ldots, n_m\rangle)[\mathrm{cct}(t'_1), \ldots, \mathrm{cct}(t'_m)],$$

  where
  1. $t'_1, \ldots, t'_m$ is an enumeration of the elements in $\{t_1, \ldots, t_k\}$, so t'_i is not isomorphic to t'_j for any $i, j \in [m]$, making m the number of unique trees, up to isomorphism, in $t_1, \ldots, t_k$,
  2. $n_i = |\{j \mid j \in [k], t_j \text{ isomorphic to } t'_i\}|$ for all i.

We write $\mathrm{CCT}(M_s)$ for the set of all compact configuration trees of M_s. □

It should be clear that there is a many-to-one correspondence between simple configuration trees t and their respective compact configuration trees $\mathrm{cct}(t)$.

Next, we show that the size of compact representation trees for simple configuration trees depends only on the automaton, not on the input word.

Lemma 5. *Let M_s be the canonical CFSA for a shuffle expression s. Then there is a constant $c \in \mathbb{N}$ that depends only on M_s, such that for every simple configuration tree t of M_s, the size of $\mathrm{cct}(t)$ is at most c.*

Proof. Let t be a simple configuration tree of M_s. Since M_s is acyclic we know that $height(t)$, and thus also $height(\mathrm{cct}(t))$, is at most $|Q|$. We argue that the index (i.e., the number of equivalence classes) of the relation *se* on t is completely decided by M_s.

Let *SCFree* be the set of subtrees t' of simple configuration trees of M_s such that in t' no $scp(M_s)$-labeled node has children. We note that since the height of trees in *SCFree* is bounded by $|Q|$ and since they branch only binarily, we know that $|SCFree|$ is finite and depends only on M_s.

Let $Layer(i, t)$ be the tree obtained from t by removing all nodes v such that there are i or more $scp(M_s)$-labeled nodes on the path from the root to v (not including v itself). We argue by induction on i, that the index of *se* on $Layer(i, t)$ depends only on i and on M_s. Since i is itself bounded by $|Q|$ this will in the end give us what we need.

In the base case, where $i = 1$, the claim holds, since $Layer(1, t) \in SCFree$ and thus $Layer(1, t)$ has a maximum number of nodes that depends only on M_s. The index of *se* can of course not exceed the number of nodes.

For the inductive case, we assume that there is a number e_i that depends only on i and on M_s, such that for all simple configuration trees t of M_s, the index of *se* on $Layer(i, t)$ is at most e_i. We obtain $Layer(i + 1, t)$ from $Layer(i, t)$ by adding trees from *SCFree* as children to $scp(M_s)$-labeled leaves of $Layer(i, t)$. For two nodes v_1 and v_2 in $nodes(Layer(i + 1, t)) \setminus nodes(Layer(i, t))$ not to be symmetrically equivalent, they must either belong to two such subtrees from *SCFree* that are not isomorphic or their closest ancestors in $Layer(i, t)$ belong to different equivalence classes of *se*. This means that in $Layer(i + 1, t)$ there can be no more than $e_i \cdot |SCFree| \cdot m$ equivalence classes of *se*, where m is the maximum size of any tree in *SCFree*. Using the induction hypothesis, this quantity depends only on i and M_s.

Since t is a simple configuration tree, in any set of symmetrically equivalent nodes, there is at most one whose corresponding subtree contains an $scp(M_s)$-labeled node that has children. Take a set $\{v_1, \ldots, v_n\}$ of symmetrically equivalent nodes (n can be arbitrarily large). Then, $\{t/v_1, \ldots, t/v_n\}$ contains at most two unique trees, the single one with $scp(M_s)$-labeled nodes with children being one, while all other subtrees are necessarily isomorphic. This immediately implies that the number of unique, up to isomorphism, subtrees of t depends only on M_s.

All that remains is to note that in $\mathrm{cct}(t)$, every node either has at most two children (non-*scp* nodes) or it has only unique, up to isomorphisms, children. Since the number of unique subtrees depends only on M_s, and $height(t) \leq |Q|$ this means that the number of nodes of $\mathrm{cct}(t)$ depends only on M_s. □

Lemma 6. *Let $M_s = (Q, \Sigma, \delta, I)$ be the canonical CFSA for a shuffle expression. Then there exists a constant $k \in \mathbb{N}$ that depends only on M_s, such that the number of distinct compact configuration trees needed by M_s for accepting all words in $\mathcal{L}(M_s)$ of length at most n is bounded by $O(n^k)$.*

Proof. We may assume, thanks to Lemma 1, that M_s is ε-efficient. This means that no intermediate configuration tree in a run over a word of length n needs to contain more than $n + 1$ leaf nodes. Indeed, whenever a configuration tree contains $n + 1$ leaf nodes, by the pigeon hole principle, at least one of the states must ultimately derive ε, since there are only n symbols in the string. As such, whenever a configuration contains $n + 1$ leafs we can safely nondeterministically choose a leaf state which can derive ε and replace it by t_ε in the next step, creating a new run. Iterating this process produces a run in which no configuration tree has more than $n + 1$ leaf nodes.

Since an acyclic CFSA will have configuration trees of height at most $|Q|$, no configuration tree needs to be of size greater than $(n + 1)|Q|$.

Lemma 5 establishes that there is a constant c such that no compact configuration tree corresponding to a simple configuration tree of M_s has more than c nodes. This also means that they contain at most c repetition counters (the counters that are placed as part of the children in *scp*-nodes in the $\mathrm{CCT}(M_s)$ construction). We have also shown that no intermediary configuration tree of M_s running on a word of length n needs to have more than $(n + 1)|Q|$ nodes.

To conclude, we note that during any step of a simple run of M_s on a word of length n, there are less than $(|Q| + 1)^c$ possible compact configuration trees when ignoring the values of the repetition counters. Furthermore, there are less than $(n + 1)|Q|$ "units" to be divided among the c counters, which can be done in less than $((n + 1)|Q|)^c$ ways. Therefore, there are less than $(|Q| + 1)^c((n + 1)|Q|)^c$ possible compact configuration trees for any step of M_s. Since c depends only on M_s, we have the desired bound of $O(n^k)$ with k depending only on M_s. □

Finally, we are ready to prove Theorem 8.

Proof (*Of Theorem* 8)**.** To compute membership for the shuffle of a shuffle language and a context-free language, we outline an extension of the CYK algorithm. The extension maintains triples consisting of a nonterminal from the context-free grammar G and two configuration trees with respect to the CFSA M_s. A triple (A, t, t') is assigned to a substring w' of the input string w if

1. $w' \in w'_1 \odot w'_2$,
2. the string w'_1 can take M from t to t', and
3. the string w'_2 can be derived from A in the grammar G.

A pair of triples (A, t, t') and (B, t', t'') for the substrings w' and w'' can be combined into a triple (C, t, t'') for the substring $w'w''$ if there is a derivation rule $C \to AB$ in G. To decide whether there is a parse for w, one starts by deriving all possible triples for every substring of w of length 1, and then uses the above combination rule to dynamically complete the parse chart.

A string of length n has $O(n^2)$ substrings, which means that $O(n^2)$ sets of triples have to be computed. From Lemma 6 we know that there is a $k \in \mathbb{N}$, that depends only on the shuffle language involved, such that no more than $O(n^k)$ distinct configuration trees have to be considered. If G has m nonterminals, there are thus no more than $O(m \cdot n^k)$ possible triples. Given that we have the sets of triples for all substrings of w, deciding whether a particular triple belongs to the set of triples for w can be done in polynomial time. Since m and k are constants, the problem is polynomial in the length n of the string. □

5. An NP-complete shuffle of two deterministic linear context-free languages

In this section we will construct two deterministic linear context-free languages such that deciding the membership problem for their shuffle is NP-complete. Phrased differently we will demonstrate the non-uniform membership problem for DLCF $\odot$ DLCF is NP-complete.

5.1. Proof preliminaries

Deterministic linear context-free languages, denoted DLCF, will be used extensively in the following. It is assumed that the reader is familiar with the relevant formalisms for these languages, for instance, deterministic pushdown automata restricted to a single pushdown reversal. For an introduction to the subject, see, e.g., the textbook by Hopcroft and Ullman [30]. We typically give inductive definitions for the DLCF languages, from which push-down automata can be easily deduced.

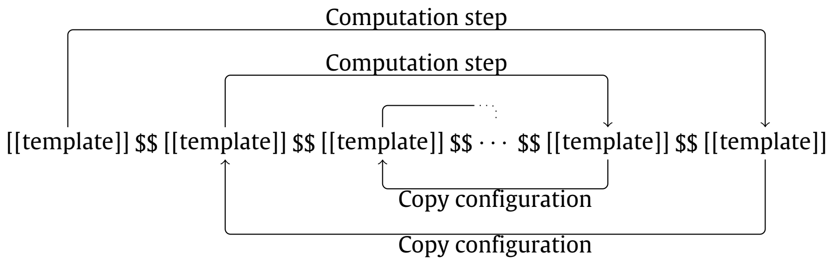
Non-deterministic polynomial time-bounded Turing machines are used heavily in the proofs to demonstrate NP-completeness. Full definitions of the machines are given, but for more complete background information on these topics see, for example, [18,36].

5.2. Proof overview

To make two DLCF languages perform a computation, they have to be made interdependent. This is done by constructing a template input string containing sequences of double-bracketed bits:

$$w = [[01]][[01]][[01]]\$\$[[01]][[01]][[01]]\$\$[[01]][[01]][[01]].$$

Assume that the *first* language contributes the string [0][1][0]\$[1][1][1]\$[1][0][1] to w, then the *second* language *has to* contribute the string [1][0][1]\$[0][0][0]\$[0][1][0] if the whole input string w is to be assembled. Notice that the bit sequence in this string is the complement of the bit sequence in the first. In this way the two shuffled languages can communicate arbitrary choices by only accepting properly bracketed input. The proof will use this to let one language make computation steps for a Turing machine, while the other language copies the configuration to link the computation steps. The following figure acts as a visual aid to see how the languages will cooperate to simulate the computation. Many details are left out; the figure only serves as a structural overview.


Computation step
Computation step
[[template]] $$ [[template]] $$ [[template]] $$ ··· $$ [[template]] $$ [[template]]
Copy configuration
Copy configuration

5.3. Parsing the shuffle of deterministic linear context-free languages

The reduction hinges on representing the computations of Turing machines as strings. To simplify the presentation, we give custom definitions of non-deterministic Turing machine configurations and runs, and work with totally ordered state spaces.

Definition 12. Let S be an *ordered set*, i.e., a set S together with a total order on S. For $i \in [|S|]$, let $S(i)$ denote the ith element in S according to this total order. When a set is given in the form of an enumeration $S = \{s_1, \ldots, s_n\}$, it is implied that $s_i = S(i)$.

Definition 13. A *non-deterministic Turing machine (NTM)* is a tuple (Q, Δ) where

- Q is a finite ordered set of states,
- $\Delta \subset Q \times \{0, 1\} \times \{\leftarrow, \rightarrow\} \times Q \times \{0, 1\}$ is a finite set of rules,
- $Q(1)$ is the initial state, and $Q(|Q|)$ is the accepting state.

The DLCF languages that we shall consider are made up of symbols from the alphabet

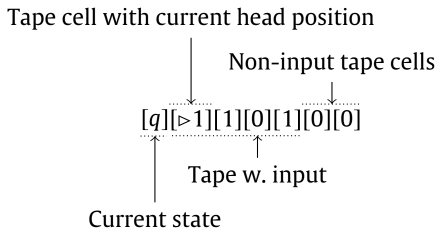
$$\Sigma_M = Q \cup \{0, 1, \triangleright, [,], \$, \#\}.$$

A configuration becomes a simple string containing both the state, tape contents, and tape position, allowing rule applications to be expressed as string rewrites.

Definition 14. The set of *configurations* of an NTM $M = (Q, \Delta)$, denoted C_M, is the set

$$C_M = \{[\} \cdot Q \cdot \{]\} \cdot \{[0], [1]\}^* \cdot \{[\triangleright 0], [\triangleright 1]\} \cdot \{[0], [1]\}^* \subset \Sigma_M^*.$$

Example 2. As will be seen in Definition 16, an NTM will be provided with tape cells it can work on by padding the input with additional cells filled with zeros. For example, an NTM with initial state q, the input string 1101, and 6 tape cells at its disposal would start in the following configuration.



Definition 15. For an NTM $M = (Q, \Delta)$, the rule $r \in \Delta$ is *applicable* to a configuration $c \in C_M$ and yields the configuration $c' \in C_M$ under the following conditions. Let $r = (q, \alpha, d, q', \alpha')$. For all strings t_1 and t_2, and $\beta \in \{0, 1\}$,

- if $d = \rightarrow$ and $c = [\cdot q \cdot] \cdot t_1 \cdot [\triangleright \cdot \alpha \cdot][\cdot \beta \cdot] \cdot t_2$ then $c' = [\cdot q' \cdot] \cdot t_1 \cdot [\cdot \alpha' \cdot][\triangleright \cdot \beta \cdot] \cdot t_2$,
- if $d = \leftarrow$ and $c = [\cdot q \cdot] \cdot t_1 \cdot [\cdot \beta \cdot][\triangleright \cdot \alpha \cdot] \cdot t_2$ then $c' = [\cdot q' \cdot] \cdot t_1 \cdot [\triangleright \cdot \beta \cdot][\cdot \alpha' \cdot] \cdot t_2$.

We denote this rule application by $c \xrightarrow{r} c'$, or $c \rightarrow c'$ leaving r implicit.

Example 3. For example, in the configuration $[q][0][1][0][\triangleright 1][1][0]$ it is possible to apply the rule $(q, 1, \rightarrow, q', 0)$ to produce the configuration $[q'][0][1][0][0][\triangleright 1][0]$. In the configuration $[q][0][1][\triangleright 0]$ a rule $(q, 0, \rightarrow, q', 0)$ cannot be applied since there is no room to move to the right, nor can the rule $(q, 1, \leftarrow, q', 0)$, since $\triangleright$ is pointing to a 0 and the rule requires a 1.

Let us now define what it means for an NTM to accept a language in time bounded by some function.

Definition 16. Take an NTM $M = (Q, \Delta)$, a function $\psi : \mathbb{N} \rightarrow \mathbb{N}$ and a string $\alpha_1 \cdots \alpha_n \in \{0, 1\}^*$. The *initial configuration* is defined as

$$I(M, \psi, \alpha_1 \cdots \alpha_n) = [\cdot Q(1) \cdot]\underbrace{[\triangleright \cdot \alpha_1 \cdot] \cdots [\cdot \alpha_n \cdot][0][0] \cdots [0]}_{\psi(n)+1 \text{ bracketed bits}},$$

the set of *final configurations* is $F(M) = ([\cdot Q(|Q|) \cdot] \cdot \Sigma_M^*) \cap C_M$.

M accepts $\alpha_1 \cdots \alpha_n$ in ψ-bounded time if and only if the initial configuration can be transformed into some final configuration by *exactly* $\psi(n)$ rule applications. That is, there exists $\psi(n) + 1$ configurations, $c_1, \ldots, c_{\psi(n)+1}$ such that $c_1 = I(M, \psi, \alpha_1 \cdots \alpha_n)$, $c_{\psi(n)+1} \in F(M)$ and $c_i \rightarrow c_{i+1}$ for all $i \in [\psi(n)]$. The *language M accepts in ψ-bounded time* is exactly the set of strings M accepts in ψ-bounded time.

The above definition is slightly irregular in that M is required to take *exactly* $\psi(n)$ steps to accept a string of length n, but any Turing machine that would accept the string in *at most* $\psi(n)$ steps can remain in the accepting state indefinitely to fulfill this condition. This makes the above definition equivalent to, e.g., that of Minsky [36]. It follows that every problem $L \in$ NP is, when suitably encoded, accepted by some NTM M in polynomially bounded time [18].

The template string defined next will be used as the input for the membership query, encoding the Turing machine input and a long specially formatted suffix to make the shuffled computation possible.

Definition 17. The *run template string* for running the machine $M = (Q, \Delta)$ in ψ-bounded time on the input string $\alpha_1 \cdots \alpha_n \in \{0, 1\}^*$ ($n \in \mathbb{N}$) is denoted $S(M, \psi, \alpha_1 \cdots \alpha_n)$ and is defined as follows. First the *configuration template* is

$$T = [[\cdot Q(1) \cdots Q(|Q|) \cdot]]\, \underbrace{[[\triangleright 01]] \cdots [[\triangleright 01]]}_{\psi(n)+1 \text{ times}}.$$

Then $S(M, \psi, \alpha_1 \cdots \alpha_n)$ equals

$$I(M, \psi, \alpha_1 \cdots \alpha_n) \cdot \underbrace{\$\$ \cdot T \cdot \$\$ \cdot T \cdots \$\$ \cdot T}_{\psi(n) \text{ occurrences of } T} \cdot \$\$\#\# \cdot \underbrace{\$\$ \cdot T^{\mathcal{R}} \cdot \$\$ \cdot T^{\mathcal{R}} \cdot \$\$ \cdots T^{\mathcal{R}}}_{\psi(n)+1 \text{ occurrences of } T^{\mathcal{R}}}.$$

Example 4. Let $M = (\{q_1, q_2\}, \Delta)$, and let $\psi(2) = 1$, then $S(M, \psi, 1)$ is

$$[q_1][\triangleright 1][0]\$\$[[q_1q_2]][[\triangleright 01]][[\triangleright 01]]\$\$\#\#\$\$]]10\triangleright[[\,]]10\triangleright[[\,]]q_2q_1[[\$\$]]10\triangleright[[\,]]10\triangleright[[\,]]q_2q_1[[.$$

The logical "bracketed" units are divided by a dotted line as a visual aid, since the $T^{\mathcal{R}}$ strings are made hard to read by their reversed brackets.

Next we define the shuffle complement with respect to a template.

Definition 18. For all strings $w, t \in \Sigma_M^*$, let $comp(w, t)$ denote the *shuffle complement of w with respect to t*, defined as

$$comp(w, t) = \{x \in \Sigma_M^* \mid t \in w \odot x\}.$$

Example 5. $comp([q_1][0][\triangleright 1], [[q_1q_2q_3]][[\triangleright 01]][[\triangleright 01]]) = \{[q_2q_3][\triangleright 1][0]\}$.

A very small but important lemma follows.

Lemma 7. *For any configuration $c \in C_M$ and configuration template T (as in Definition 17) if it holds that $|c|_{[} = \frac{1}{2}|T|_{[}$ then*

1. *$comp(c, T) = \{c'\}$ for some string c', and*
2. *$comp(c', T) = \{c\}$.*

Proof (*Sketch*). If we have a configuration template string T as in Definition 17 and a configuration c, such that $|c|_{[} = \frac{1}{2}T_{[}$, then this means that T and c have the *same number of bracketed sections* (T has each section double-bracketed, $[[\triangleright 01]]$, c has each single-bracketed as in $[\triangleright 1]$). As a consequence $comp(c, T) = \{c'\}$ is a singleton. This is easy to see, by observing that the interleaving of c can only ever pick *one* of the [symbols in each [[pair in T, since it needs to read a] symbol before reading another left bracket. This forces it to skip the other bracket in the pair, meaning that the bracketed sections will match up one-to-one in the shuffle.

This in turn enforces that c' will *also* have $|c'|_{[} = \frac{1}{2}|T|_{[}$, and will have similarly single-bracketed sections, containing the complement of those in c with respect to the string $\triangleright 01$. The same argument therefore establishes that $comp(c', T) = \{c\}$. $\square$

Next we define a deterministic linear context-free language which will encode the steps a given NTM can make.

Definition 19. For an NTM $M = (Q, \Delta)$ the *step language for M*, denoted $L_{step(M)}$, is the smallest language that contains the string #, and all strings $c_1 \cdot \$ \cdot l \cdot \$ \cdot c_2^{\mathcal{R}}$, where $l \in L_{step(M)}$, $c_1, c_2 \in C_M$, and $c_1 \xrightarrow{r} c_2$ for some $r \in \Delta$.

Example 6. Let $M = (\{q_1, q_2\}, \{(q_1, 0, \rightarrow, q_2, 1)\})$, then for example

$$[q_1][\triangleright 0][1][0]\$\#\$]0[\,]1\triangleright[\,]1[\,]q_2[\in L_{step(M)},$$

$$[q_1][0][\triangleright 0][0]\$\#\$]0\triangleright[\,]1[\,]0[\,]q_2[\in L_{step(M)},$$

$$[q_1][\triangleright 0][1][0]\$[q_1][\triangleright 0][1][0]\$\#\$]0[\,]1\triangleright[\,]1[\,]q_2[\$]0[\,]1\triangleright[\,]1[\,]q_2[\in L_{step(M)}.$$

It might not be immediately obvious that this language is both linear and deterministic, so let us look at how a deterministic linear push-down automaton can accept it. An automaton for $L_{step(M)}$ can start by pushing the first half of the string onto its stack, validating that it is in the regular language $(C_M \cdot \$)^*$ in the process. When it encounters # it switches to popping off the stack, while popping $c_1 \in C_M$ reading the reverse of $c_2 \in C_M$ on the string, and immediately rejecting unless c_2 differs from c_1 by exactly one rule application from Δ. The automaton can achieve this by checking that c_1 and c_2 are equal in all positions except the states and the immediate neighborhoods of the $\triangleright$ symbol, both of which are constant-sized and can be remembered in the state of the automaton. It then validates that these differences correspond to a rule in Δ.

Now we turn to the other DLCF language, which is responsible for linking the computation steps by making copies of the complement of configurations. It consists of strings of the form $\bar{c}_1 \cdot \$ \cdot \bar{c}_2 \cdots \bar{c}_2^{\mathcal{R}} \cdot \$ \cdot \bar{c}_1^{\mathcal{R}}$ where each $\bar{c}_i$ is such that $\{\bar{c}_i\} = comp(c, T)$ for some configuration c and configuration template T. Compare the constructed strings to those in Example 5.

Definition 20. For an NTM $M = (Q, \Delta)$ the *inverted copy language for M*, denoted $L_{copy(M)}$, is defined as $L_{copy(M)} = \$ \cdot L$ where L is in turn defined as follows. First let

- $\bar{Q}_i = [\cdot Q(1) \cdot Q(2) \cdots Q(i-1) \cdot Q(i+1) \cdots Q(|Q|) \cdot]$ for $i \in [|Q|]$,
- $U = \{[\triangleright 0], [\triangleright 1], [0], [1]\} \cdot \{[\triangleright 0], [\triangleright 1], [0], [1]\}^*$.

Then the strings in L are exactly the following. First, for all $t \in U$

$$\#\$ \cdot (\bar{Q}_{|Q|} \cdot t)^{\mathcal{R}} \in L.$$

Second, for all $\bar{c} \in \{\bar{Q}_i \mid i \in [|Q|]\} \cdot U$, and $l \in L_{copy(M)}$

$$\bar{c} \cdot \$ \cdot l \cdot \$ \cdot \bar{c}^{\mathcal{R}} \in L_{copy(M)}.$$

Example 7. Let $M = (\{q_1, q_2, q_3\}, \Delta)$, where q_3 is the final (last) state. Then among the strings in $L_{copy(M)}$ are

$$\#\$]0[\,]1\triangleright[\,]0\triangleright[\,]q_2q_1[\,,$$

$$[q_1q_3][\triangleright 0][\triangleright 1][1]\$\#\$]0[\,]1\triangleright[\,]0\triangleright[\,]q_2q_1[\$]1[\,]1\triangleright[\,]0\triangleright[\,]q_3q_1[\,,$$

$$[q_2q_3][1][\triangleright 0]\$[q_1q_3][\triangleright 0][\triangleright 1][1]\$\#\$]0[\,]1\triangleright[\,]0\triangleright[\,]q_2q_1[\$]1[\,]1\triangleright[\,]0\triangleright[\,]q_3q_1[\$]0\triangleright[\,]1\triangleright[\,]q_3q_2[\,.$$

It should be clear that this language is both deterministic and linear, the symbol # marking the center playing a key role. The argument is similar to the one in the proof of Lemma 7, but slightly simpler, because no rules need to be taken into account.

This only leaves us to assemble the pieces to prove the main result.

Theorem 9. *Take any $w \in \{0,1\}^*$, NTM M and function $\psi : \mathbb{N} \to \mathbb{N}$. Then M accepts w in ψ-bounded time if and only if $S(M, \psi, w) \in L_{step(M)} \odot L_{copy(M)}$.*

The proof of Theorem 9 is divided into Lemmas 8 and 9; the first showing the "only if" direction, the second the "if" direction.

Lemma 8. *Take any string $\alpha_1 \cdots \alpha_n \in \{0,1\}^*$, NTM $M = (Q, \Delta)$ and function $\psi : \mathbb{N} \to \mathbb{N}$. If M accepts the string $\alpha_1 \cdots \alpha_n$ in ψ-bounded time then $S(M, \psi, \alpha_1 \cdots \alpha_n) \in L_{step(M)} \odot L_{copy(M)}$.*

Proof. Let $c_1, \ldots, c_{\psi(n)+1} \in C_M$ be the sequence of configurations which makes M accept $\alpha_1 \cdots \alpha_n$ (so $c_1 = I(M, \psi, \alpha_1 \cdots \alpha_n)$ and $c_{\psi(n)+1} \in F(M)$). Then construct the string

$$w_{step} = c_1 \cdot \$ \cdot c_2 \cdot \$ \cdots \$ \cdot c_{\psi(n)} \cdot \$\#\$ \cdot c^{\mathcal{R}}_{\psi(n)+1} \cdot \$ \cdot c^{\mathcal{R}}_{\psi(n)} \cdot \$ \cdots \$ \cdot c^{\mathcal{R}}_2.$$

Notice that $w_{step} \in L_{step(M)}$ by construction. Now, for each $i \in [\psi(n)+1]$ let $\{\bar{c}_i\} = comp(c_i, T)$ where T is a configuration template as in Definition 17. Recall that this complement is always a singleton. Now let

$$w_{copy} = \$ \cdot \bar{c}_2 \cdot \$ \cdot \bar{c}_3 \cdot \$ \cdots \bar{c}_{\psi(n)} \cdot \$\#\$ \cdot \bar{c}^{\mathcal{R}}_{\psi(n)+1} \cdot \$ \cdot \bar{c}^{\mathcal{R}}_{\psi(n)} \cdots \$ \cdot \bar{c}^{\mathcal{R}}_2.$$

It is then straightforward to check that $w_{copy} \in L_{copy(M)}$ by construction.

As an abbreviation denote the template string $S(M, \psi, \alpha_1 \cdots \alpha_n)$ by w. All that remains is to show that $w \in w_{step} \odot w_{copy}$. To illustrate:

$$\begin{array}{rlllllllll} w = & c_1 & \$\$ & T & \$\$ \cdots \$ & T & \$\$\#\#\$\$ & T^{\mathcal{R}} & \$ \cdots \$ & T^{\mathcal{R}}, \\ w_{step} = & c_1 & \$ & c_2 & \$ \cdots \$ & c_{\psi(n)} & \$\#\$ & c^{\mathcal{R}}_{\psi(n)+1} & \$ \cdots \$ & c^{\mathcal{R}}_2, \\ w_{copy} = & & \$ & \bar{c}_2 & \$ \cdots \$ & \bar{c}_{\psi(n)} & \$\#\$ & \bar{c}^{\mathcal{R}}_{\psi(n)+1} & \$ \cdots \$ & \bar{c}^{\mathcal{R}}_2. \end{array}$$

w and w_{step} both start with c_1, so cancel that bit. Next w contains two dollar signs, one corresponds to the initial in w_{copy} and one the next symbol in w_{step}. After that a T configuration template is next in w, c_2 is next in w_{step}, and $\bar{c}_2$ is next in w_{copy}. By construction $T \in c_2 \odot \bar{c}_2$, leaving us again with \$\$ next in w and a single \$ next in the other strings, and so on through all of w. $\square$

Lemma 9. *Take any string $\alpha_1 \cdots \alpha_n \in \{0,1\}^*$, NTM $M = (Q, \Delta)$ and function $\psi : \mathbb{N} \to \mathbb{N}$. If $S(M, \psi, \alpha_1 \cdots \alpha_n) \in L_{step(M)} \odot L_{copy(M)}$ then M accepts $\alpha_1 \cdots \alpha_n$ in ψ-bounded time.*

Proof. Let $w = S(M, \psi, \alpha_1 \cdots \alpha_n)$, and let the strings $w_{step} \in L_{step(M)}$ and $w_{copy} \in L_{copy(M)}$ such that $w \in w_{step} \odot w_{copy}$ (the lemma assumes these exist).

No string in $L_{step(M)} \cup L_{copy(M)}$ has two \$ symbols in a row, while every \$ occurrence in w consists of two \$ symbols. This enforces that every such \$\$ substring in w is divided up so that one belongs to w_{step} and one to w_{copy} (so $|w_{step}|_\$ = |w_{copy}|_\$ = \frac{1}{2}|w|_\$$). Combining this with the way $L_{step(M)}$ and $L_{copy(M)}$ are constructed it follows that the shuffling must have this structure

$$\begin{array}{rlllllllll} w = & c_1 & \$\$ & T & \$\$ \cdots \$ & T & \$\$\#\#\$\$ & T^{\mathcal{R}} & \$ \cdots \$ & T^{\mathcal{R}}, \\ w_{step} = & c_1 & \$ & c_2 & \$ \cdots \$ & c_{\psi(n)} & \$\#\$ & d^{\mathcal{R}}_{\psi(n)+1} & \$ \cdots \$ & d^{\mathcal{R}}_2, \\ w_{copy} = & & \$ & e_2 & \$ \cdots \$ & e_{\psi(n)} & \$\#\$ & e^{\mathcal{R}}_{\psi(n)+1} & \$ \cdots \$ & e^{\mathcal{R}}_2, \end{array}$$

for some configurations $c_1, \ldots, c_{\psi(n)}, d_2, \ldots, d_{\psi(n)+1} \in C_M$, and some strings $e_2, \ldots, e_{\psi(n)+1}$. That is, the assumption that $w \in w_{step} \odot w_{copy}$ does together with the placement of \$ symbols imply that

$$T \in c_i \odot e_i \quad \text{for all } i \in \{2, \ldots, \psi(n)\}, \tag{1}$$

$$T \in d_i \odot e_i \quad \text{for all } i \in \{2, \ldots, \psi(n)+1\}. \tag{2}$$

The second is not reversed since $T^{\mathcal{R}} \in d^{\mathcal{R}}_i \odot e^{\mathcal{R}}_i \iff T \in d_i \odot e_i$. Next, recall from Lemma 7 that $comp(c_i, T)$ and $comp(d_i, T)$ are singletons for all $i \in \{2, \ldots, \psi(n)\}$. Eqs. (1) and (2) dictate that the string $e_i \in comp(c_i, T)$ and the string $e_i \in comp(d_i, T)$, so $comp(c_i, T) = comp(d_i, T) = \{e_i\}$. Reversing this (again by Lemma 7) yields $comp(e_i, T) = \{c_i\} = \{d_i\}$, so $c_i = d_i$. Let (the previously undefined) $c_{\psi(n)+1}$ be equal to $d_{\psi(n)+1}$ as well. The construction of $L_{step(M)}$ and $L_{copy(M)}$ dictates that

- $c_i \to d_{i+1}$, and therefore $c_i \to c_{i+1}$, for all $i \in [\psi(n)]$,
- $c_1 = I(M, \psi, \alpha_1 \cdots \alpha_n)$,
- $comp(e_{\psi(n)+1}, T) = \{c_{\psi(n)+1}\} \subset F(M)$ (since $e_{\psi(n)+1}$ does *not* contain the final state by construction).

From this it follows that $c_1, \ldots, c_{\psi(n)+1}$ is a correct configuration sequence which makes M accept $\alpha_1 \cdots \alpha_n$. □

It follows from Theorem 9 that the non-uniform membership problem for the shuffle of DLCF languages is NP-complete.

Corollary 4. *For an input string w it is an NP-complete problem to decide whether or not $w \in L \odot L'$ when L and L' are deterministic linear context-free languages, even when L and L' are fixed.*

Proof. The problem is trivially *in* NP. Membership in context-free languages can be decided in polynomial time, and we can, in polynomial time, guess any w_1 and w_2 such that $w = w_1 \odot w_2$ and check if $w_1 \in L$ and $w_2 \in L'$.

Hardness follows easily from Theorem 9. Pick any NTM M and *polynomial* function ψ such that M runs in ψ-bounded time. This characterizes NP by definition. Fix the languages $L = L_{step(M)}$ and $L' = L_{copy(M)}$. It is then possible to check if M would accept an input string w in ψ-bounded time by checking if $S(M, \psi, w) \in L \odot L'$. The reduction is polynomial since $S(M, \psi, w)$ produces a string that is of length $\mathcal{O}(\psi(|w|)^2)$ and can, because of its exceedingly simple structure, be constructed in time $\mathcal{O}(\psi(|w|)^2)$. Thus, choosing M such that it accepts an NP-complete language in polynomial time (e.g. a universal NTM) concludes the proof. □

Corollary 5. *Corollary 4 holds even for languages over an alphabet of size* 3.

Proof sketch. We use the alphabet {0, [,]} and use the 0 symbol to encode all other symbols in unary. To start with, let 1 be written as 00 and ▷ as 0000 (changing the template bit [[▷01]] into [[0000000]]). For example, if one automaton reads [00000], corresponding to [▷0] (or [0▷], but the order is irrelevant), then the remaining subsequence is [00], correctly corresponding to [1].

Now $$ can be represented as [[0^{16}]] (i.e. 16 zeros with brackets around), with each language reading $ as [0^8]. Similarly ## can be represented as [[0^{32}]], each language reading # as [0^{16}]. Notice that these are sufficiently long that they cannot be divided into subsequences that allow them to be confused with any other case.

Finally, the state vector part of the template, [[$q_1 q_2 \cdots$]], can be replaced by $|Q|$ "bits", [[000]][[000]] ⋯ [[000]]. The step language reads [00] (representing "1") in the ith position if q_i is the current state, and [0] in all other positions, and the copy language copies the remainder as usual. □

6. Conclusions and future work

Concurrent finite-state automata combine the expressive power of context-free and shuffle languages. The CFSA languages are properly included in the context-sensitive languages, and minor restrictions of the device suffice to obtain the regular, context-free, and shuffle languages. CFSA have comparatively nice closure properties, and can be checked for emptiness in polynomial time.

It remains the case, however, that the computational complexity of the membership problem for a formalism is of critical importance in practical applications. This paper demonstrates the non-uniform membership problem NP-hard for a quite restricted CFSA (of the form DLCF $\odot$ DLCF). This may appear daunting, but many possibilities remain for deriving new language classes from CFSA that allow for more efficient parsing, both through syntactic and semantic restrictions. It is also not unexpected that finding efficiently decidable membership problems is difficult. This paper demonstrates that the efficiency of deciding membership for the shuffle languages relies heavily on using only a limited number of shuffle operations. On the other hand, the non-uniform membership problem for the shuffle of a shuffle language and a context-free language can be decided in polynomial time. Considering the difficulty of the membership problem for very restricted CFSA it is also positive to find that the general uniform membership problem for CFSA is just NP-complete. For the shuffle of a regular language and a context-free language even the uniform membership problem is polynomial.

Future work will strive to determine the complexity of the non-uniform membership problem for further restrictions of CFSA. For example unordered shuffle could be considered, or perhaps the class of languages that a CFSA could generate if at most one path in the configuration tree may exceed a constant depth at a time, or even cases where the CFSA is forced to encode some information about the shuffling choices in the string. In the other direction, to better understand the boundary where the membership problem turns NP-hard, the construction used to demonstrate that the membership problem for the shuffle of two deterministic linear context-free languages should be possible to extend (to, e.g., visibly pushdown languages [1]). Success in this direction will, hopefully, give a better understanding of what properties of a CFSA need to be restricted to get an efficient membership problem while still giving rise to a powerful class of languages.

Acknowledgements

We thank an anonymous reviewer of a previous version of this paper for informing us that the non-uniform version of the membership problem is NP-hard for the shuffling of two deterministic context-free languages [40] and for many helpful suggestions. Henrik Björklund was supported by the Swedish Research Council (Vetenskapsrådet) grant 621-2011-6080.

References

[1] R. Alur, P. Madhusudan, Visibly pushdown languages, in: Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, STOC '04, New York, NY, USA, ACM, 2004, pp. 202–211.
[2] G.E. Barton, On the complexity of ID/LP parsing 1, Computational Linguistics 11 (4) (1985) 205–218.
[3] M. Berglund, H. Björklund, J. Högberg, Recognizing shuffled languages, in: Proc. Language and Automata Theory and Applications, 2011, pp. 142–154.
[4] J. Berstel, L. Boasson, O. Carton, J.E. Pin, A. Restivo, The expressive power of the shuffle product, Information and Computation 208 (11) (2010) 1258–1272.
[5] F. Biegler, M. Daley, I. McQuillan, On the shuffle automaton size for words, in: Proc. Descriptional Complexity of Formal Systems, 2009, pp.79–89.
[6] H. Björklund, M. Bojańczyk, Shuffle expressions and words with nested data, in: Proc. Mathematical Foundations of Computer Science, 2007, pp. 750–761.
[7] S.B. Bloom, Z. Ésik, Axiomatizing shuffle and concatenation in languages, Information and Comptuation 139 (1) (1997) 62–91.
[8] M. Bojańczyk, A. Muscholl, T. Schwentick, L. Segoufin, C. David, Two-variable logic on words with data, in: Proc. Logic in Computer Science, 2006, pp. 7–16.
[9] J. Brzozowski, G. Jirskov, B. Li, Quotient complexity of ideal languages, in: A. López-Ortiz (Ed.), Proc. Latin American Theoretical Informatics Symposium, in: Lecture Notes in Computer Science, vol. 6034, Springer, Berlin, Heidelberg, 2010, pp. 208–221.
[10] C. Câmpeanu, K. Salomaa, S. Yu, Tight lower bound for the state complexity of shuffle of regular languages, Journal of Automata, Languages and Combinatorics 7 (2002) 303–310.
[11] S. Carberry, Techniques for plan recognition, User Modeling and User-Adapted Interaction 11 (1–2) (2001) 31–48.
[12] T. Colcombet, On families of graphs having a decidable first order theory with reachability, in: Proc. International Colloquium on Automata, Languages and Programming, 2002, pp. 98–109.
[13] M. Daley, M. Domaratzki, K. Salomaa, Orthogonal concatenation: language equations and state complexity, Journal of Universal Computer Science 16 (5) (2010) 653–675.
[14] A. Darrasse, K. Panagiotou, O. Roussel, M. Soria, Boltzmann generation for regular languages with shuffle, in: Proc. GASCOM 2010, Montréal, Canada, 2010.
[15] R. Downey, M. Fellows, Parameterized Complexity, Springer-Verlag, 1999.
[16] Z. Ésik, M. Bertol, Nonfinite axiomatizability of the equational theory of shuffle, Acta Informatica 35 (6) (1998) 505–539.
[17] J. Flum, M. Grohe, Parameterized Complexity Theory, Springer-Verlag, 2006.
[18] M.R. Garey, D.S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, USA, 1990.
[19] V. Garg, M. Ragunath, Concurrent regular expressions and their relationship to Petri nets, Theoretical Computer Science 96 (2) (1992) 285–304.
[20] W. Gelade, W. Martens, F. Neven, Optimizing schema languages for XML: numerical constraints and interleaving, SIAM Journal on Computing 39 (4) (2009) 1486–1530.
[21] S. Ginsburg, The Mathematical Theory of Context Free Languages, McGraw-Hill, 1966.
[22] S. Ginsburg, E.H. Spanier, Mappings of languages by two-tape devices, Journal of the ACM 12 (July) (1965) 423–434.
[23] J. Gischer, Shuffle languages, Petri nets, and context-sensitive grammars, Communications of the ACM 24 (9) (1981) 597–605.
[24] A.C. Gómez, J.E. Pin, Shuffle on positive varieties of languages, Theoretical Computer Science 312 (2004) 433–461.
[25] H. Gruber, M. Holzer, Finite automata, digraph connectivity, and regular expression size, in: Proc. International Colloquium on Automata, Languages and Programming, Springer, 2008.
[26] L.H. Haines, On free monoids partially ordered by embedding, Journal of Combinatorial Theory (6) (1968) 94–98.
[27] Y.S. Han, K. Salomaa, D. Wood, Operational state complexity of prefix-free regular languages, in: Proc. Automata, Formal Languages, and Related Topics, 2009, pp. 99–115.
[28] D. Haussler, H.P. Zeiger, Very special languages and representations of recursively enumerable languages via computation histories, Information and Control 47 (3) (1980) 201–212.
[29] J. Högberg, L. Kaati, Weighted unranked tree automata as a framework for plan recognition, in: Proc. Fusion, 2010.
[30] J.E. Hopcroft, J.D. Ullman, Introduction To Automata Theory, Languages, And Computation, 1st ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.
[31] J. Jedrzejowicz, A. Szepietowski, Shuffle languages are in P, Theoretical Computer Science 250 (1–2) (2001) 31–53.
[32] M. Kuhlmann, G. Satta, Treebank grammar techniques for non-projective dependency parsing, in: Proc. Conference of the European Chapter of the Association for Computational Linguistics, 2009, pp. 478–486.
[33] C. Löding, Ground tree rewriting graphs of bounded tree width, in: Proc. Symposium on Theoretical Aspects of Computer Science, 2002, pp. 559–570.
[34] A. Mateescu, G. Rozenberg, A. Salomaa, Shuffle on trajectories: syntactic constraints, Theoretical Computer Science 197 (1-2) (1998) 1–56.
[35] A. Mayer, L. Stockmeyer, Word problems — this time with interleaving, Information and Computation 115 (1994) 293–311.
[36] M.L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.
[37] B. Nagy, Languages generated by context-free grammars extended by type $AB \rightarrow BA$ rules, Journal of Automata, Languages and Combinatorics 14 (2009) 175–186.
[38] B. Nagy, On a hierarchy of permutation languages, in: M. Ito, Y. Kobayashi, K. Shoji (Eds.), Proc. Automata, Formal Languages and Algebraic Systems, World Scientific, Singapore, 2010, pp. 163–178.
[39] J. Nivre, Non-projective dependency parsing in expected linear time, in: Proc. Annual Meeting of the Association for Computational Linguistics and the Joint Conference on NLP of the Asian Federation of NLP, 2009, pp. 351–359.
[40] W.F. Ogden, W.E. Riddle, W.C. Rounds, Complexity of expressions allowing concurrency, in: Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL '78, New York, NY, USA, ACM, 1978, pp. 185–194.
[41] W.E. Riddle, An approach to software system behavior description, Computer Languages 4 (1) (1979) 29–47.
[42] C. Schmidt, N. Sridharan, J. Goodson, The plan recognition problem: an intersection of psychology and artificial intelligence, Artificial Intelligence 11 (1, 2) (1978).
[43] A. Shaw, Software descriptions with flow expressions, IEEE Transactions on Software Engineering 4 (1978) 242–254.
[44] M.K. Warmuth, D. Haussler, On the complexity of iterated shuffle, Journal of Computer and System Sciences 28 (3) (1984) 345–358.