Shuffle Languages for Plan Recognition

Henrik Björklund and Johanna Högberg

Department of Computing Science, Umeå University
90187 Umeå, Sweden
{henrikb, johanna}@cs.umu.se

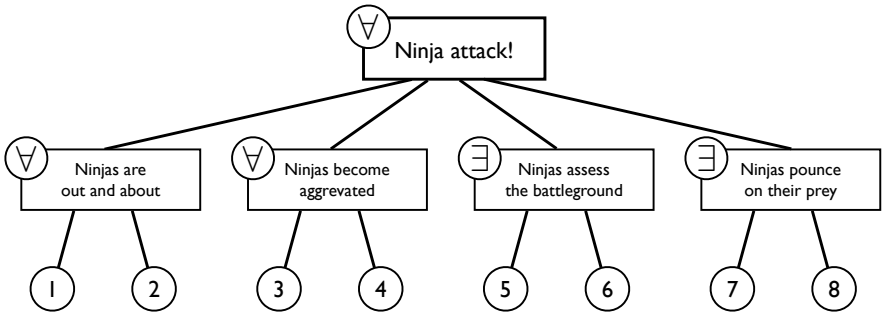
Abstract. Considering applications in plan recognition, we study how shuffle operators can be incorporated into language models that also capture the context-free languages. Our aim is to provide a formal framework for plan recognition, in which the balance between expressive power and computational tractability can be tuned through natural restrictions on the model. We also study the complexity of the membership problem for various restrictions of the suggested model.

1 Introduction

The goal of this paper is to provide a formal framework for language theoretic aspects of plan recognition and to study the complexity of various related computational problems. Plan recognition is the task of inferring an agent's plan or goal, based on observations of the agent's actions, or the effects of those actions [**?**,**?**]. The field, which is also known as *activity recognition*, has applications in, e.g., military surveillance, sensor analysis, and medicine, wherein cognitive support for patients suffering from dementia is a frequently used example. Plan recognition relies on a *plan library*, i.e., a set of activity schemas that can be executed by the agent. The *plan recognition problem* consists in deducing which plans in the library are active, given a series of observations. The abstract machinery used to solve this problem is called a *plan recognizer*.

In [**?**], the authors remark that the features and algorithms used in the fields of plan recognition and natural language processing overlap, but that dialog between the two fields has not been effective. They continue to draw a parallel between *hierarchical task networks* (HTNs) and context-free grammars, and argue that parsing algorithms for the latter can be used to match HTNs against observation series. A similar approach is taken in [**?**], where we investigate the usefulness of unranked tree automata (UTA) for plan recognition. In the intended application, a human analyst enters plan description trees (PDT) into the plan library. Each PDT declares a plan for accomplishing a specific goal, and the nodes of the PDT correspond to intermediate plans. The internal nodes are labelled by operator symbols that combine simple plans into more complex plans, and the leaf nodes are labelled by observable events. The intermediate levels in the PDT provide structure and simplify understanding [**?**].

Each PDT p describes a language $\mathcal{L}$ over an alphabet of observable events Σ, consisting of the sequences in Σ^* that realise the top-level plan of p. Similarly, a



1. No Chuck Norris reruns on TV this week to keep the Ninjas at home.
2. The local apparel store is sold out of black turtlenecks and tights. Again!
3. You refer to ninjutsu as a pajamas sport.
4. You compare a Sai to an ugly-looking fork.
5. Black lint is found in the tumble dryer.
6. All the black Smarties are missing from your easter egg.
7. The trip-wires in your ceiling go off.
8. Your spider sense is tingling.

Fig. 1. A PDT for a ninja attack. The observable events are given as text.

set of PDTs describe the union of its constituent PDTs' languages. To solve the plan recognition problem in this setting means to identify occurrences of $\mathcal{L}$ in a sequence of events. This can be done by turning the PDT into a CFG or UTA, and then using traditional chart parsing to find (fragments) of words in $\mathcal{L}$.

Example 1. A toy PDT t for recognizing that you might be targeted by a ninja attack is given in Figure **??**. The plan operator that labels an internal node v of t determines how many of the subplans immediately below v need to be realised, for the subplan at v to be realised (this example only uses operators $\forall$ and $\exists$).

A limitation that is common for both CFG and UTA is that they cannot represent parallel activities. To compensate, *shuffling* can be introduced into the model. Consider Example **??** again. Suppose that for the two leftmost subplans in Figure **??**, the temporal order within the subplans is important, but not the temporal order between the subplans. Then, any interleaving of the sequences 12 and 34, i.e., the sequences in $\{1234, 1324, 1342, 3412, 3142, 3124\}$, would indicate that both subplans were active. In general, if one subplan requires the sequence w of events, while another requires w', but there is no temporal ordering required between the two subplans, then a mechanism that recognizes the shuffle of w and w', written $w \odot w'$, can detect that both subplans are active. We can also consider the shuffling of two languages, the shuffle closure of one language, etc.

Various aspects of shuffling has been studied in the theory of formal languages, see, e.g., [**?**,**?**,**?**,**?**]. In this paper, we take the *shuffle languages* considered by Jedrzejowicz and Szepietowski [**?**] as the starting point and study how they

can be integrated into a formal framework for studying plan recognition from the perspective of formal languages and complexity.

The formalism we suggest is an automaton model with an equivalent grammatical interpretation. It generalizes both shuffle languages and context-free languages. It is thus very expressive, and cannot be expected to have nice algorithmic properties in general. Rather, we suggest it as a flexible framework for investigating various kinds of combinations of shuffling with other constructions that are useful in plan recognition. We also start this investigation by studying the expressive power and closure properties of (restrictions of) the formalism, and by proving complexity results for some relevant classes.

Contributions. We introduce *concurrent finite state automata* (CFSA), provide an automata- and a grammar-based semantics for the device (which yield the same class of languages), and investigate the expressive power and closure properties of CFSA (Section **??**).

For the shuffle languages (as used in [**?**]), the *uniform membership problem*, where both the language description and the word to be checked are part of the input, is NP-complete [**?**], while the *non-uniform membership problem* is solvable in polynomial time [**?**]. We shed further light on the complexity of the membership problem by showing that the uniform version, parameterized by the number of shuffle operations, is hard for the complexity class W[1]. 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 if it is theoretically polynomial. We conclude by showing that the non-uniform membership problem for the shuffling of a context-free language and a regular language is solvable in polynomial time.

2 Preliminaries

Sets and numbers. If S is a set, then S^* is the set of all finite sequences of elements of S, and $\mathit{precl}(S)$ is the set of all finite prefix-closed subsets of S^*. The powerset of S is denoted by $\mathit{pow}(S)$. We write $\mathbb{N}$ for the natural numbers, or $\mathbb{N}^+$ if we wish to exclude 0 from $\mathbb{N}$. For $k \in \mathbb{N}$, we write $[k]$ for $\{1, \ldots, k\}$. Note that $[0] = \emptyset$. The domain of a mapping f is denoted $\mathit{dom}(f)$.

An *alphabet* is a a finite nonempty set. We denote $\Sigma \cup \{\varepsilon\}$ by Σ_ε and the set of all regular expressions over the alphabet Σ by $\mathrm{Reg}(\Sigma)$. The length of a string $w = \alpha_1 \cdots \alpha_n \in \Sigma^*$ is written $|w|$, and for every $\alpha \in \Sigma$, $|w|_\alpha = |\{i \in [n] \mid \alpha_i = \alpha\}|$.

Trees. The set T_Σ of *(unranked) trees* over the alphabet Σ consists of all mappings $t\colon D \to \Sigma$, where $D \in \mathit{precl}(\mathbb{N})$. The *empty tree*, denoted t_ε, is the unique tree such that $\mathit{dom}(t) = \emptyset$. We henceforth refere to $\mathit{dom}(t)$ as the *nodes of* t (and write $\mathit{nodes}(t)$ rather than $\mathit{dom}(t)$) to help intution.

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

The mapping $ind : [k] \to \mathbb{N}$ is given by $ind(i) = \min_{j \in \mathbb{N}}\{|[j] \cap nodes(t)| = i\}$. We denote a tree t as $f[t_1, \ldots, t_k]$ if $k = |nodes(t) \cap \mathbb{N}|$ and, for every $i \in [k]$, $t_i = t/ind(i)$. 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 shuffle operation, shuffle closure and shuffle expressions from [**?**].

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

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

for every $\alpha_1, \alpha_2 \in \Sigma$, and $u_1, u_2 \in \Sigma^*$.

Operation $\odot$ extends to a mapping $\hat{\odot} : pow(\Sigma^*) \times pow(\Sigma^*) \to pow(\Sigma^*)$ with

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

For readability, we sometimes use infix notation for $\odot$, and, due to the associativity of the shuffle operator, we also write $\odot(\odot(w, w'), w'')$ as $\odot(w', w'', w'')$. From here on, we write the shuffle operation for languages as $\odot$ rather than $\hat{\odot}$.

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

$$\mathcal{L}^{\odot} = \overset{\infty}{\underset{i=0}{\cup}} \mathcal{L}^{\odot_i}, \text{ where } \mathcal{L}^{\odot_0} = \{\varepsilon\} \text{ and } \mathcal{L}^{\odot_i} = \mathcal{L} \odot \mathcal{L}^{\odot_{i-1}} \ .$$

Shuffle expressions are regular expressions that can additionally use the shuffle operators. Formally, the set $\mathrm{Sh}(\Sigma)$ of all shuffle expressions over alphabet Σ is formed as follows. Every $\alpha \in \Sigma$ is a shuffle expression, as well as ε and $\emptyset$. 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 ${s_1}^{\odot}$. *Pure shuffle expressions* are shuffle expressions that do not use concatination or Kleene star. The set of all pure shuffle expressions over Σ is denoted by $\mathrm{PSh}(\Sigma)$. The language $\mathcal{L}(s)$ of a shuffle expression s is defined in the usual way. The *shuffle languages* are the languages defined by shuffle expressions.

3 Concurrent finite-state automata

In this section, we introduce *concurrent finite-state automata* (CFSA). The device is 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, recursive calls can be made in parallel. The latter feature allows for an unbounded number of invocations to be executed simultaneously, although only one invocation at a time is permitted to consume a symbol from the input string. The formal definition reads as follows:

Definition 2 (CFSA) A *Concurrent FSA* is a tuple $M = (Q, \Sigma, \delta, I)$, where
- Q is an alphabet of *states*,
- Σ is an alphabet of *input symbols*,
- δ is a set of *transitions* that is partitioned into $\delta_1 \cup \delta_2 \cup \delta_3$, where
  - $\delta_1 \subseteq Q \times \Sigma_\varepsilon \times Q$ is a set of *horizontal* transitions,
  - $\delta_2 \subseteq Q \times \Sigma_\varepsilon \times \mathrm{PSh}(Q) \times Q$ is a set of *vertical* transitions, and
  - $\delta_3 \subseteq Q$ is a set of *terminal* transitions,
- $I \subseteq Q$ is the set of *initial* states. □

During a run of a CFSA, we maintain a branching call-stack (represented as an unranked tree over the alphabet of states) to track concurrent invocations of the automaton.

Definition 3 (CFSA semantics) The CFSA $M = (Q, \Sigma, \delta, I)$ *accepts* the string $w = \alpha_1 \dots \alpha_k \in \Sigma_\varepsilon^*$ if there are trees $t_0, \dots, t_k \in T_Q$ such that $t_0 \in I$, $t_k = t_\varepsilon$, and for every $i \in [k]$, there is a node $v \in \mathit{leaves}(t_{i-1})$ such that either
- $t_{i-1} = t[\![v \leftarrow q]\!]$ and $t_i = t[\![v \leftarrow q']\!]$ for some $(q, \alpha_i, q') \in \delta_1$;
- $t_{i-1} = t[\![v \leftarrow q]\!]$ and $t_i = t[\![v \leftarrow q'[u]]\!]$ for some $(q, \alpha_i, s, q') \in \delta_2$ and string $u \in \mathcal{L}(s)$; or
- $t_{i-1} = t[\![v \leftarrow q]\!]$, $t_i = t[\![v \leftarrow t_\varepsilon]\!]$, $\alpha_i = \varepsilon$, and $q \in \delta_3$.

The language $\mathcal{L}(M)$ *recognised* by M is the set of strings that M accepts. □

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 given by the shuffle expression $(\{abc\} + \{acb\} + \{bac\} + \{bca\} + \{cab\} + \{cba\})^{\odot}$, but it is not a context-free language. Both $\mathcal{L}_1$ and $\mathcal{L}_2$ can be recognised by a CFSA, and so can $\mathcal{L}_1 \cup \mathcal{L}_2$, which is neither a context-free nor a shuffle language. Thus the class of languages recognized by CFSA properly extend the union of the context-free languages and the shuffle languages.

By supplying an alternative, but equivalent, semantics to that of Definition **??**, we can choose to view a CFSA M as a grammar. When we take this point of view, we refer to M as a *concurrent grammar*.

Definition 4 (Alternative semantics) Let $G = (Q, \Sigma, \delta, I)$ be a concurrent FSA (in this context; a concurrent grammar). The *intermediate sentences* of G, denoted $\Delta(G)$, is the smallest subset of $(\Sigma \cup Q \cup \{f_\odot, (,), , \})^*$, where $f_\odot$ is a special symbol that does not appear in $\Sigma \cup Q$, such that
- $\{\varepsilon\} \cup \Sigma \cup Q \subseteq \Delta(G)$;
- $ww' \in \Delta(G)$, for every $w, w' \in \Delta(G)$; and
- $f_\odot(w_1, \dots, w_k) \in \Delta(G)$, for every $k \in \mathbb{N}^+$ and $w_1, \dots, w_k \in \Delta(G) \setminus \{\varepsilon\}$.

Let $u, v, w, w' \in \Delta(G)$. We say that there is a *transition step* from uwv to $uw'v$ and write $uwv \to uw'v$ if
- $w = q$, $w' = \alpha q'$, and $(q, \alpha, q') \in \delta_1$;
- $w = q$, $w' = \alpha f_\odot(q_1, \dots, q_k)\, q'$, $(q, \alpha, s, q') \in \delta_2$, and $q_1 \cdots q_k \in \mathcal{L}(s)$;
- $w = f_\odot(w_1, \dots, w_k)$, $w' \in \odot_{i \in [k]} w_i$, and $w_i \in \Sigma^*$ for every $i \in [k]$; or
- $w = q$, $w' = \varepsilon$, and $q \in \delta_3$.

As usual, the reflexive and transitive closure of $\rightarrow$ is denoted $\overset{*}{\rightarrow}$. The *language generated by* G is $\mathcal{L}_G(G) = \{w \in \Sigma^* \mid \exists q \in I : q \overset{*}{\rightarrow} w\}$. □

A straight-forward inductive proof yields Theorem **??**.

Theorem 5 (Semantic equivalence). *For every CFSA* M, $\mathcal{L}(M) = \mathcal{L}_G(M)$.

Closure properties. It is known that the context-free languages and the shuffle languages alike are closed under union, concatenation and Kleene-star. Additionally, the context-free languages are closed under intersection with a regular language. Shuffle languages are not, but they are closed under shuffle and shuffle closure. Neither language family is closed under intersection or complementation. As we shall see, the CFSA have comparatively nice closure properties.

Theorem 6 (Closure properties). *The languages recognised 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.*

Proof (Sketch) The remaining closure proofs can be found in the appendix.

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 a regular language, so both are recognisable by CFSA. As we shall see, their intersection $\mathcal{L} = \{a^n b^n c^n \mid n \in \mathbb{N}\}$, is not. To obtain a contradiction, assume that $\mathcal{L} = \mathcal{L}(M) = \mathcal{L}_G(M)$ for some CFSA $M = (Q, \Sigma, \delta, I)$. There must the exist a derivation tree t with respect to M of every string in $w \in \mathcal{L}$. Suppose (i) that t contains a node u labelled with a production $(q, \alpha, \odot(q_1 \cdots q_k), q')$, (ii) that the subtrees of t rooted at u are $t_\alpha, t_1, \ldots, t_k, t'$, and (iii) that there are trees t_i and t_j, $i, j \in [k]$, $i < j$, such that $a \in \{t_i(v) \mid v \in \mathit{leaves}(t_i)\}$ and $b \in \{t_j(v) \mid v \in \mathit{leaves}(t_j)\}$. It is then easy to see that the shuffle operation at u may arrange the leaves of t_i and t_j so that a b appears to the left of an a. As the labels in $\mathcal{L}$ are strictly ordered, we draw the conclusion that M can only apply the shuffle operation to nonterminals that generate strings over some singleton alphabet Σ_1. Computing the shuffle of a sequence of strings $w_1, \ldots, w_k \in \Sigma_1^*$ yields the singleton set $\{w_1 \cdots w_k\}$, so the shuffle operations can be rewritten as regular concatenation. However, without shuffle operations M is a context-free grammar, and it is well known that $\mathcal{L}$ is not a context-free language. Consequently, $\mathcal{L}$ is not recognisable by a CFSA.

Complementation. Since the CFSA languages are closed under union, but not under intersection, we can conclude that they are not closed under complementation, since $(L_1 \cap L_2) = \overline{(\overline{L_1} \cup \overline{L_2})}$. □

Corollary 7. *Neither the CFSA languages nor the shuffle languages are closed under intersection with a regular language.*

Restrictions and expressive power. Our motivation for introducing CFSA is to provide a formal device for plan recognition, in which the balance between

expressive power and computational tractability can be tuned through natural restrictions on the model. The restrictions considered here are as follows:

A CFSA M is

- *horizontal* if the set of vertical transitions is empty;
- *non-branching* if the vertical transitions form a subset of $(Q \times \Sigma \times Q \times Q)$, i.e., if the shuffle expressions in the definition are all singleton state symbols;
- *finitely branching* if all the shuffle expressions in the definitions of vertical transitions define finite languages, i.e., if no expression uses shuffle closure;
- *acyclic* if there is no configuration tree t of M such that some state q appears twice on a path from the root to a leaf of t.

The above restrictions affect the expressive power of CFSA as follows:

Observation 8 (Regular languages) A language is regular if and only if it is recognised by a horizontal CFSA.

Theorem 9 (Context-free languages). *A language is context-free if and only if it is recognised by a non-branching CFSA.*

Theorem 10 (Shuffle languages). *A language is a shuffle language if and only if it is recognised by an acyclic CFSA.*

Corollary 11 (Shuffle languages w.o. shuffle closure). *A language is a closure-free shuffle language if and only if it is recognised by an acyclic and finitely branching CFSA.*

Corollary 12. *Closure-free shuffle languages are regular.*

Corollary **??** follows from Corollary **??** and the fact that an acyclic and finitely branching CFSA admits only a finite number of distinct configuration trees.

Theorem 13 (Mildly context-sensitive). *The languages recognised by CFSA are properly contained in the context-sensitive languages.*

4 Decision problems

We continue to address the decision problems associated with CFSA. Let us first consider emptiness testing, which is useful, e.g., for sanity-checking the automaton specification.

Theorem 14 (Emptiness). *The emptiness problem for CFSA is decidable in polynomial time.*

We now turn to the membership problem, beginning with acyclic CFSA, i.e., the restriction of CFSA that recognises the shuffle languages.

Corollary 15. *For acyclic CFSA*

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

2. the uniform membership problem is NP-complete.

The uniform membership problem is NP-complete already for acyclic and finitely branching CFSA, which only recognise regular languages (see Corollary **??**). The explanation is that for some languages, CFSA offer a more succinct form of representation than nondeterministic finite automata.

Corollary **??** tells us that the problem is polynomial for a fixed automaton but NP-hard if the automaton is considered input. This is however not the whole story. Exactly how does the automaton contribute to the complexity? Does a change in the automaton only change the coefficients of the polynomial? Or does it even affect the degree of the polynomial? We partially answer this question by showing that when parameterized by the maximal branching of the automaton, the uniform memebership problem is *not fixed-parameter tractable* (unless FPT = W[1], which is considered very unlikely and would have far-reaching complexity-theoretic implications).[1] 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 16 An instance of the parameterized uniform shuffle 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 word in Σ^*. The parameter is the maximal branching of M, i.e., the maximal length of a word in one of the shuffle languages in the definition of M:s vertical transitions. The question is whether $w \in \mathcal{L}(M)$. □

If the problem were fixed-parameter tractable, there would be an algorithm for it with running time $f(k) \cdot n^c$, where f is a computable function, k is the parameter (the maximal branching) of the instance, n is the instance size (i.e., $|M| + |w|$), and c is a constant. Theorem **??** gives strong evidence to the contrary.

Theorem 17. *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. We start by defining the latter problem.

Definition 18 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. □

In terms of parameterized complexity, k is the parameter and the problem is known to be $W[1]$-complete [**?**].

*Proof (of Theorem **??**).* 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 $|\Sigma| = O(n + m)$, $|r| = O(k \cdot n^2 + k^2 \cdot m)$,

[1] For more on parameterized complexity theory, see, e.g., [**?**,**?**].

$|w| = O(k \cdot n + m)$, the shuffle operator appears $O(k^2)$ times in r, and $w \in L(r)$ 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 $v_1^k \cdot v_2^k \cdots v_n^k \cdot \mathit{edges}$, where *edges* is any enumeration of the edges in E. We define the regular expressions s, t, u by

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

Finally, we define

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

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. In other word, the rest of the expression can only use k distinct vertex names.
- Each instance of expression u matches one sequence $v_i \cdot v_j \cdot e_{i,j}$. Thus, in total, 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 vertex names. Thus the $k(k-1)/2$ edge names, which are distinct since *edges* is an enumeration of E, represent edges that all have both their endpoints in a set of vertices of size k.
- The expression t matches any extra vertex and edge names that are left over.
- 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. Since the reduction is polynomial and the size of the new parameter (the number of shuffle operators in the expression) depends only on the old parameter (k), it is a fixed-parameter reduction. We conclude that uniform membership for closure-free shuffle expressions, parameterized by the number of shuffle operators, is W[1]-hard.

Using Corollary **??** (and the construction from the proof of Theorem **??**) it is straightforward to construct 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 r, and the maximal branching of M_r is the same as the number of shuffle operators in r. Thus there is a fixed-parameter reduction from k-Clique to parameterized membership for acyclic and finitely branching CFSA, and the latter problem is W[1]-hard. □

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

Theorem 19. *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.*

Since acyclic and finitely branching CFSA only contribute a more compact representation, Theorem **??** extends, via Corollary **??**, to non-uniform membership for the shuffle of a context-free language and a closure-free shuffle language:

Corollary 20. *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.*

5 Conclusions and Future Work

In concurrent finite-state automata, the expressive power of context-free and shuffle languages combine. The synthesis of nested and freely ordered dependencies is particularly convenient when modelling parallel activities in plan recognition. On the algorithmic side, 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 sanity-checked in polynomial time.

To be of practical use in plan recognition, the non-uniform membership problem needs to be efficiently decidable. We know that the uniform membership problem for unrestricted CFSA is NP-complete, and the non-uniform membership problem is likely to be computationally expensive without additional restrictions. Future work should therefore strive to determine the complexity of the non-uniform membership problem for restricted and unrestricted CFSA. If even very sparse use of shuffling has a large negative impact on the complexity, one could consider replacing the shuffle operator with a weaker alternative, e.g., a permutation operator that rearranges the symbols of a string in *any* order.

A Proofs from Section ??

Theorem ??. *The languages recognised 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.*

Proof. Let $M = (Q, \Sigma, \delta, I)$ and $M' = (Q', \Sigma, \delta', I')$ be CFSA. We assume without loss of generality that $Q \cap Q' = \emptyset$.

Union. To construct a CFSA for the union of the languages of M and M' we just have to add a unique new initial state with horizontal ε-transitions to all initial states of M and M' and use the union of the two automata.

Concatination. For concatination, we again take the union of the states and transitions of M and M', and additionally add a new, unique, initial state q_I. We then add a vertical transition $(q_I, \varepsilon, q_1, q_2)$ for every pair $(q_1, q_2) \in (I \times I')$.

Kleene closure. Next, we construct a CFSA for the Kleene closure of M. All we have to do is add a new, unique, initial state q_I to Q along with the terminal transition q_I and the vertical transitions $(q_I, \varepsilon, q, q_I)$ for every $q \in I$. 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 a unique inital state q_I and a state q_F to the union of the two automata. We also add the terminal tranisition q_F and the vertical transition $(q_I, \varepsilon, (q_1 \odot q_2), q_F)$ for each pair $(q_1, q_2) \in (I \times I')$.

Shuffle closure. To construct the shuffle closure of the language of M, we again add a unique inital state q_I along with a state q_F. Additionally, we add the terminal transition q_F and the vertical transition $(q_I, \varepsilon, I^{\odot}, q_F)$. This means that the new automaton can spawn any number of copies of M that will run in parallel over the string.

The proofs for non-closedness under intersection and complementation can be found in the body of the paper. □

Theorem ??. *A language is context-free if and only if it is recognised by a non-branching CFSA.*

Proof (Sketch). It is easy to turn a context-free grammar $G = (N, \Sigma, \gamma, S)$ on 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 transition $(q, \alpha, \overline{q})$ in δ_1 and a transition $\overline{q}$ in δ_3.
- For every rule $q \to pp'$ in γ, there is a transition (q, ε, 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. □

Theorem ??. *A language is a shuffle language if and only if it is recognised by an acyclic CFSA.*

Proof. The only-if direction follows directly from the proof of Theorem **??** since the constructions there preserve acyclicity.

Given a CFSA $M = (Q, \Sigma, \delta, I)$ we show how to construct a shuffle expression s recognizing $\mathcal{L}(M)$. We consider two states $q, q' \in Q$ to be *connected* if there is a transition $(q, \alpha, q') \in \delta_1$ or a transition $(q, \alpha, r, q') \in \delta_2$, for some $\alpha \in \Sigma_\varepsilon$ and $r \in \mathrm{PSh}(Q)$. With this notion of connectivity, let $C_1, \ldots, C_k$ 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 are states $q, q' \in C_i$, a transition $(q, \alpha, r, q') \in \delta_2$, and a state $p \in C_j$ such that p appears in some word in $\mathcal{L}(r)$. Since M is acyclic, it follows that G_M is acyclic.

Next, we create a set Δ with one unique new alphabet symbol for each vertical transtition. Let $h : \delta_2 \to \Delta$ be the bijection mapping each $d \in \delta_2$ to the corresponding alphabet symbol. Also, for each $d \in \delta_2$, let q_d be a new state. Define M_h to be the CFSA obtained from M by replacing each vertical transition $d = (q, \alpha, r, q')$ with the horizontal transitions (q, α, q_d) and $(q_d, h(d), q')$. Notice that the connected components of M_h are the same as the connected components of M and that M_h is a finite automaton recognizing a regular language.

For each $q \in Q$, let the *regular* expression $r(q)$ be such that $\mathcal{L}(r(q))$ is exactly the language recognized by M_h when starting from q. Such a regular expression can be computed from M_h 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 $M[q]$, i.e., the CFSA obtained from M by replacing I by $\{q\}$. We do this by induction on the structure of G_M.

Suppose q belongs to a leaf of G_M. This means that there are no vertical transitions in the connected component q belongs to. This, in turn, means that $s(q) = r(q)$.

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 Δ by appropriate shuffle expressions. In particular, consider symbol $h(d) \in \Delta$ that corresponds to $d = (p, \alpha, r, p') \in \delta_2$. The shuffle expression for $h(d)$ is obtained from r by replacing each occurance of a state q' in r by $s(q')$. Finally, the shuffle expression for M is the union of the shuffle expressions for states in I, i.e.,

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

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

Theorem ??. *The languages recognised by CFSA are properly contained in the context-sensitive languages.*

Proof. Let $M = (Q, \Sigma, \delta, I)$ be a CFSA and w an input string. If there is an accepting run of M on w from some initial state p, then a non-deterministic Turing machine can guess and verify this run in linear space by proceeding as follows. (1) The TM simulates a run of M on w starting in p, but every time a vertical transition (q, α, s, q') is used on the top level, the TM guesses what part of the subsequent string is to be consumed by s, marks this segment off with brackets and a pointer to s, and continues in state q' after the closing bracket until it has read all of w. If it accepts what it has seen so far, then it goes on to verify each of the bracketed segments. Let w' be such a segment, annotated with the expression s. The automaton guesses a number $n \in [|w'|]$, a way to partition w' into n subsequences $w_1, \ldots, w_n$ (i.e., $w' \in w_1 \odot \cdots \odot w_n$), a sequence $q_1, \ldots, q_n \in Q$, and a string $u \in \mathcal{L}(s) \cap Q^* q_1 Q^* \cdots Q^* q_n Q^*$ s.t. every state $p \notin \{q_1, \ldots, q_n\}$ that appears in u accepts the empty string (i.e., $\varepsilon \in \mathcal{L}(M[p])$) and $|u| \leq c \cdot n$, where $c \in \mathbb{N}$ is a constant depending on M.[2] For each $i \in [n]$, the TM repeats the procedure starting at (1) for the string w_i and the initial state q_i. This process continues recursively until no unprocessed bracketed segment has non-zero length. We note that the total amount of information that was recorded in the process is linear in $|w|$, so the non-uniform membership problem for CFSA can be decided by a linearly bounded nondeterministic TM. □

B Proofs from Section ??

Theorem ??. *The emptiness problem for CFTA is decidable in polynomial time.*

Proof. Let $M = (Q, \Sigma, \delta, I)$ be a CFTA. For any state q of M, let $M[q]$ be the automaton obtained by replacing I by $\{q\}$ in M. We say that a state q of M is *live* if $\mathcal{L}(M[q])$ is nonempty.

Given M, let $\mathcal{F} \subseteq Q$ be the smallest set satisfying the following conditions.

1. $F_0 = \delta_3$
2. if $q \in F_i$, then $q \in F_{i+1}$
3. if $(q, \alpha, q') \in \delta_1$ for some $\alpha \in \Sigma_\epsilon$ and some $q' \in F_i$, then $q \in F_{i+1}$
4. if $(q, \alpha, s, q') \in \delta_2$ for some $\alpha \in \Sigma_\epsilon$, some $q' \in F_i$ and some s such that the set $\mathcal{L}(s) \cap F_i^*$ is nonempty, then $q \in F_{i+1}$
5. $\mathcal{F} = \cup_{i=0}^{\infty} F_i$

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 $q \in \delta_3$, and thus $M[q]$ accepts the string ε.

[2] A shrinking argument yields that if $\mathcal{L}(s) \cap Q^* q_1 Q^* \cdots Q^* q_n Q^*$ is non-empty, then it contains a string of length less than $c \cdot n$.

Assume that every state in F_i is live, and consider a $q \in F_{i+1} \setminus F_i$. If there is a rule $(q, \alpha, q') \in \delta_1$, 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 $(q, \alpha, s, q') \in \delta_2$ such that $q' \in F_i$ and there is a word $u = q_1 q_2 \cdots q_m \in \mathcal{L}(s) \cap F_i^*$. If this is the case, then there is a word $w_{q'}$ accepted by $M[q']$ and for each position q_i of u, there is a word w_{q_i} that is accepted by $M[q_i]$. This, in turn, means that $\alpha \cdot w_{q_1} \cdot w_{q_2} \cdot \cdots \cdot w_{q_m} \cdot w_{q'}$ is accepted by $M[q]$. Thus q is live.

For the other direction, assume that q is live, i.e., there is a word $w = w_1 \cdots w_m$ that is accepted by $M[q]$. Let $\rho = t_0 \cdots t_m$ be an accepting run of $M[q]$ on w. We show by induction that every state that appears in a tree in ρ belongs to $\mathcal{F}$. In particular, this means that q belongs to $\mathcal{F}$, since $t_0 = q$. Since ρ is accepting, we have $t_m = t_\varepsilon$. Thus 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 node v).

1. $t_{i-1} = t[\![v \leftarrow q]\!]$, $t_i = t[\![v \leftarrow q']\!]$, and there is a transition $(q, \alpha_i, q') \in \delta_1$. If this is the case, $q \in \mathcal{F}$ and thus all states of t_{i-1} belong to $\mathcal{F}$.
2. $t_{i-1} = t[\![v \leftarrow q]\!]$, $t_i = t[\![v \leftarrow q'[u]]\!]$, and there is a transition $(q, \alpha_i, s, q') \in \delta_2$ such that $u \in \mathcal{L}(s)$. Again, $q \in \mathcal{F}$ 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 CFTA can be decided in polynomial time. □

Corollary ??. *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 **??** and the fact, proved in [**?**], that non-uniform parsing for shuffle expressions is polynomial.

For the uniform membership problem, membership in NP is obvious – just guess and verify a run of the automaton. NP-hardness follows by an easy adaptation of a result by Mayer and Stockmeyer [**?**]. □

Theorem ??. *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. Let $G = (N, \Sigma, \delta, S)$ and $M = (Q, \Sigma, \gamma, I, F)$ be a context-free grammar on Chomsky normal form and an NFA, respectively.

A *parse triple* for G and M over a string $w = w_1 \cdots w_m$ is a triple (A, q_1, q_2) in $((N \cup \{\varepsilon\}) \times Q \times Q)$ such that w can be partitioned into subsequences w_1 and w_2, where w_1 is in the language of (N, Σ, δ, A) (or w_1 is empty, if $A = \varepsilon$),

and w_2 is in the language of $(Q, \Sigma, \gamma, \{q_1\}, \{q_2\})$. There are at most $(|N|+1)\cdot|Q|^2$ distinct parse triples.

The idea of our algorithm is to compute all parse triples for all substrings of w, in order of increasing length. In the end, $w \in \mathcal{L}(G) \odot \mathcal{L}(M)$ if and only if there is a parse triple (S, q_I, q_F) for the whole of w such that S is the start symbol of G, $q_I \in I$, and $q_F \in F$. Since w has $O(m^2)$ substrings we will compute at most $O(m^2 \cdot |N| \cdot |Q|^2)$ parse triples.

For substrings of length one, computing the triples is trivial. Assume that we have computed all the parse triples for all substrings of length $k-1$. We show how to compute the parse triples for a substring of length k. Let $v = v_1 \cdots v_k$ be such a substring. To find out whether (ε, q_1, q_2) is a parse triple for v, we proceed as follows. We check whether there is an $i \in [k-1]$ and a state q such that (ε, q_1, q) is a parse triple for $v_1 \cdots v_i$, and (ε, q, q_2) is a parse triple for $v_{i+1} \cdots v_k$. If this is the case, (ε, q_1, q_2) is a parse triple for v.

To determine whether (A, q_1, q_2), $A \in N$, is a parse triple for v, we proceed in two steps. First, if there is a rule $A \to a$ in δ, for some $a \in \Sigma$, we check whether there is an $i \in [k]$ and a $q \in Q$ such that $v_i = a$, (ε, q_1, q) is a parse triple for $v_1 \cdots v_{i-1}$, and (ε, q, q_2) is a parse triple for $v_{i+1} \cdots v_k$. If this is the case, (A, q_1, q_2) is a parse triple for v. Second, we check, for each rule $A \to BB'$ whether there is an $i \in [k]$ and a $q \in Q$ such that (B, q_1, q) is a parse triple for $v_1 \cdot v_i$ and (B', q, q_2) is a parse triple for $v_{i+1} \cdot v_k$. In this case too, (A, q_1, q_2) is a parse triple for v. □