Characterization and Complexity Results on Jumping Finite Automata

Henning Fernau[a,*], Meenakshi Paramasivan[a], Markus L. Schmid[a], Vojtěch Vorel[b]

[a] *Fachbereich 4 – Abteilung Informatik, Universität Trier, D-54286 Trier, Germany*
[b] *Faculty of Mathematics and Physics, Charles University, Malostranské nám. 25, Prague, Czech Republic*

Abstract

In a jumping finite automaton, the input head can jump to an arbitrary position within the remaining input after reading and consuming a symbol. We characterize the corresponding class of languages in terms of special shuffle expressions and survey other equivalent notions from the existing literature. Moreover, we present several results concerning computational hardness and algorithms for parsing and other basic tasks concerning jumping finite automata.

Keywords: Jumping Finite Automata, Shuffle Expressions, Commutative Languages, Semilinear Sets, NP-hard problems, Exponential Time Hypothesis

1. Introduction

Throughout the history of automata theory, the classical finite automaton has been extended in many different ways: two-way automata, multi-head automata, automata with additional resources (counters, stacks, etc.), and so on. However, for all these variants, it is always the case that the input is read in a continuous fashion. On the other hand, there exist models that are closer to the classical model in terms of computational resources, but that differ in how the input is processed (e. g., restarting automata [72] and biautomata [48]). One such model that has drawn comparatively little attention are the jumping finite automata (JFA) introduced by Meduna and Zemek [65, 66], which are like classical finite automata with the only difference that after reading (i. e., consuming) a symbol and changing into a new state, the input head can jump to an arbitrary position of the remaining input.

We provide a characterization of the JFA languages in terms of expressions using shuffle, union, and iterated shuffle, which enables us to put them into the context of classical formal language results. Note that this paper considerably deviates from and adds on to the conference version [19], as JFA languages have many more connections with the classical approaches of formal languages.

The contributions of this paper can be summarized as follows.

*Corresponding author
Email addresses: Fernau@uni-trier.de (Henning Fernau), Paramasivan@uni-trier.de (Meenakshi Paramasivan), MSchmid@uni-trier.de (Markus L. Schmid), vorel@ktiml.mff.cuni.cz (Vojtěch Vorel)

- We introduce a variant of regular-like expressions, called alphabetic shuffle expressions, that characterize JFA languages and put them into the context of earlier literature in the area of shuffle expressions. This also resolves several questions from [66], as we show (this way) that JFA languages are closed under iterated shuffle. This approach also clarifies the closure properties under Boolean operations.

  Moreover, we also investigate the intersection of the JFA languages with the regular languages and consider the problems of deciding for a given regular language or a given JFA language, whether or not it is also a JFA language or a regular language, respectively.

- Alphabetic shuffle expressions are naturally related to semilinear sets, which allows us to derive a star-height one normal form for these expressions.

- We also discuss generalized variants of the two models presented so far, namely, *general jumping finite automata* (*GJFA*) and *SHUF expressions*. In these models, transition labels (or axioms, respectively) are words, not single symbols only. We prove the incomparability of these two language classes and show how they relate to the class of JFA languages.

- Finally, we arrive at several complexity results, in particular, regarding the parsing complexity of the various mechanisms. This is also one of the questions raised in [66]. For instance, it is demonstrated that there are fixed NP-complete GJFA languages. Furthermore, we strengthen the known hardness of the universal word problem for JFA by giving a lower bound based on the exponential time hypothesis.

2. Preliminaries

In this section, we present basic language-theoretical definitions and present the main concepts of this work, i. e., jumping finite automata and special types of shuffle expressions.

2.1. Basic Definitions

We assume the reader to be familiar with the standard terminology in formal language theory and language operations like catenation, union, and iterated catenation, i. e., Kleene star. For a finite alphabet Σ and for a word $w \in \Sigma^*$ and $a \in \Sigma$, by $|w|_a$ we denote the number of occurrences of symbol a in w. By ε we denote the empty word.

First, let us define the language operations of shuffle and permutation, and the notion of semilinearity.

Definition 1. The *shuffle operation*, denoted by $\sqcup\!\sqcup$, is defined by

$$\begin{aligned} u \sqcup\!\sqcup v &= \left\{ x_1y_1x_2y_2\dots x_ny_n : \begin{array}{l} u = x_1x_2\dots x_n, v = y_1y_2\dots y_n, \\ x_i, y_i \in \Sigma^*, 1 \leq i \leq n, n \geq 1 \end{array} \right\}, \\ L_1 \sqcup\!\sqcup L_2 &= \bigcup_{\substack{u \in L_1 \\ v \in L_2}} (u \sqcup\!\sqcup v), \end{aligned}$$

for $u, v \in \Sigma^*$ and $L_1, L_2 \subseteq \Sigma^*$.

Remark 2. The following inductive definition of the shuffle operation, equivalent to Definition 1, shall be useful in some of our proofs. We have

$$\begin{aligned} \varepsilon \sqcup\!\sqcup u &= \{u\}, \\ u \sqcup\!\sqcup \varepsilon &= \{u\}, \\ au \sqcup\!\sqcup bv &= a(u \sqcup\!\sqcup bv) \cup b(au \sqcup\!\sqcup v), \end{aligned}$$

for every $u, v \in \Sigma^*$ and $a, b \in \Sigma$.

The set of permutations of a word can be then conveniently defined using the shuffle operation.

Definition 3. The set $\mathrm{perm}(w)$ of all permutations of w is inductively defined as follows: $\mathrm{perm}(\varepsilon) = \{\varepsilon\}$ and, for every $a \in \Sigma$ and $u \in \Sigma^*$, $\mathrm{perm}(au) = \{a\} \sqcup\!\sqcup \mathrm{perm}(u)$. The permutation operator extends to languages in the natural way. More precisely, $\mathrm{perm}(L) = \bigcup_{w \in L} \mathrm{perm}(w)$ for $L \subseteq \Sigma^*$.

Analogously to the iterated catenation, an iterated shuffle operation can be defined as follows.

Definition 4. For $L \subseteq \Sigma^*$, the *iterated shuffle* of L is

$$L^{\sqcup\!\sqcup,*} = \bigcup_{n=0}^{\infty} L^{\sqcup\!\sqcup,n},$$

where $L^{\sqcup\!\sqcup,0} = \{\varepsilon\}$ and $L^{\sqcup\!\sqcup,i} = L^{\sqcup\!\sqcup,i-1} \sqcup\!\sqcup L$, $i \geq 1$.

Let $\mathbb{N}$ denote the set of nonnegative integers. For any sets A and B, the set of all total functions $f : A \to B$ is denoted B^A. For finite sets A, the number of elements of A is denoted $|A|$. For $n \geq 1$, let $\mathbb{N}^n$ be the n-fold Cartesian product of $\mathbb{N}$ with itself. For $x, y \in \mathbb{N}^n$, i. e., $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$, let $x + y = (x_1 + y_1, \ldots, x_n + y_n)$ and for $c \in \mathbb{N}$, let $cx = (cx_1, \ldots, cx_n)$.

Definition 5. A subset $A \subseteq \mathbb{N}^n$ is said to be *linear* if there are $v, v_1, \ldots, v_m \in \mathbb{N}^n$ such that

$$A = \{v + k_1 v_1 + k_2 v_2 + \cdots + k_m v_m : k_1, k_2, \ldots, k_m \in \mathbb{N}\}.$$

A subset $A \subseteq \mathbb{N}^n$ is said to be *semilinear* if it is a finite union of linear sets.

A permutation of the coordinates in $\mathbb{N}^n$ preserves semilinearity. Let Σ be a finite set of n elements. A *Parikh mapping* ψ from Σ^* into $\mathbb{N}^n$ is a mapping defined by first choosing an enumeration $a_1, \ldots, a_n$ of the elements of Σ and then defining inductively $\psi(\varepsilon) = (0, \ldots, 0)$, $\psi(a_i) = (\delta_{1,i}, \ldots, \delta_{n,i})$, where $\delta_{j,i} = 0$ if $i \neq j$ and $\delta_{j,i} = 1$ if $i = j$, and $\psi(au) = \psi(a) + \psi(u)$ for all $a \in \Sigma$, $u \in \Sigma^*$. For clarity, we sometimes add the alphabet Σ as a subscript to ψ. Any two Parikh mappings from Σ^* into $\mathbb{N}^n$ differ only by a permutation of the coordinates of $\mathbb{N}^n$. Hence, the concept introduced in the following definition is well-defined.

Definition 6. Let Σ be a finite set of n elements. A subset $A \subseteq \Sigma^*$ is said to be a *language with the semilinear property*, or *slip* language for short, if $\psi(A)$ is a semilinear subset of $\mathbb{N}^n$ for a Parikh mapping ψ of Σ^* into $\mathbb{N}^n$. The class of all slip languages is denoted by $\mathcal{PSL}$.

2.2. Jumping Finite Automata and Shuffle Expressions

Following Meduna and Zemek [65, 66], we denote a *general finite machine* as $M = (Q, \Sigma, R, s, F)$, where Q is a finite set of *states*, Σ is the *input alphabet*, $\Sigma \cap Q = \emptyset$, R is a finite set of *rules*[1] of the form $py \to q$, where $p, q \in Q$ and $y \in \Sigma^*$, $s \in Q$ is the *start state* and $F \subseteq Q$ is a set of *final states*. If all rules $py \to q \in R$ satisfy $|y| \leq 1$, then M is a *finite machine*.

We interpret M in two ways.

- As a (general) finite automaton: a *configuration* of M is any string in $Q\Sigma^*$, the binary *move relation* on $Q\Sigma^*$, written as $\Rightarrow$, is defined as follows:

$$pw \Rightarrow qz \iff \exists\, py \to q \in R : w = yz\,.$$

- As a (general) jumping finite automaton: a *configuration* of M is any string in $\Sigma^*Q\Sigma^*$, the binary *jumping relation* on $\Sigma^*Q\Sigma^*$, written as $\curvearrowright$, satisfies:

$$vpw \curvearrowright v'qz' \iff \exists\, py \to q \in R\ \exists\, z \in \Sigma^* : w = yz \ \wedge\ vz = v'z'\,.$$

We hence obtain the following languages from a (general) finite machine M:

$$L_{\mathrm{FA}}(M) = \{w \in \Sigma^* : \exists\, f \in F : sw \Rightarrow^* f\},$$
$$L_{\mathrm{JFA}}(M) = \{w \in \Sigma^* : \exists\, u, v \in \Sigma^*\ \exists\, f \in F : w = uv \wedge usv \curvearrowright^* f\}\,.$$

This defines the language classes $\mathcal{REG}$ (accepted by finite automata), $\mathcal{JFA}$ (accepted by JFAs) and $\mathcal{GJFA}$ (accepted by GJFAs). Moreover, $\mathcal{CFL}$ denotes the class of context-free languages.

Next, we define a special type of expressions that use the shuffle operator. Such shuffle expressions have been an active field of study over decades; we only point the reader to [42], [43] and [45]. We first recall the definition of the SHUF expressions introduced by Jantzen [41], from which we then derive α-SHUF expressions, which are tightly linked to jumping finite automata.

Definition 7. The symbols $\emptyset, \varepsilon$ and each $w \in \Sigma^+$ are (atomic) SHUF expressions. If S_1, S_2 are SHUF expressions, then $(S_1 + S_2), (S_1 \sqcup S_2)$ and ${S_1}^{\sqcup,*}$ are SHUF expressions.

The semantics of SHUF expressions is defined in the expected way, i. e., $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$, $L(w) = \{w\}$, $w \in \Sigma^+$, and, for SHUF expressions S_1 and S_2, $L(S_1 + S_2) = L(S_1) \cup L(S_2), L(S_1 \sqcup S_2) = L(S_1) \sqcup L(S_2)$, and $L({S_1}^{\sqcup,*}) = L(S_1)^{\sqcup,*}$.

A SHUF expression is an *α-SHUF expression*, if its atoms are only $\emptyset, \varepsilon$ or single symbols $a \in \Sigma$. Since α-SHUF expressions are SHUF expressions, the semantics is already defined.

Notice that we could introduce (classical) regular expressions in the very same way (i. e., we only have to substitute the shuffle operation by the catenation and the iterated shuffle by the Kleene star). Clearly, these characterize the regular languages.

Let us illustrate the concepts defined above by two examples.

[1] We also refer to rules as *transitions* with *labels* from Σ^*.

Example 8. Let M be the finite machine depicted in Figure 1, which accepts the regular language $L_{\mathrm{FA}}(M) = L((abc)^*)$. However, if we interpret M as a jumping finite automaton, it accepts the non-context-free language $L_{\mathrm{JFA}}(M) = \{w \in \{a,b,c\}^* : |w|_a = |w|_b = |w|_c\}$. Obviously, $L_{\mathrm{JFA}}(M)$ is also defined by the α-SHUF expression $(a \sqcup\!\sqcup b \sqcup\!\sqcup c)^{\sqcup\!\sqcup,*}$ and, furthermore, $\mathrm{perm}(L_{\mathrm{FA}}(M)) = L_{\mathrm{JFA}}(M)$. As shall be demonstrated later, every JFA language can be expressed by an α-SHUF expression and $\mathrm{perm}(L_{\mathrm{FA}}(M)) = L_{\mathrm{JFA}}(M)$ holds for every finite machine M.

Example 9. The general finite machine M' depicted in Figure 2 accepts the regular language $L_{\mathrm{FA}}(M') = L((abcd)^*)$. However, it is not easy to describe the language $L_{\mathrm{JFA}}(M')$ in a simple way. Obviously, $\mathrm{perm}(L_{\mathrm{FA}}(M')) \neq L_{\mathrm{JFA}}(M')$ since $bacd \notin L_{\mathrm{JFA}}(M')$ and, furthermore, the SHUF expression $(ab \sqcup\!\sqcup cd)^*$ does not describe $L_{\mathrm{JFA}}(M')$ either.

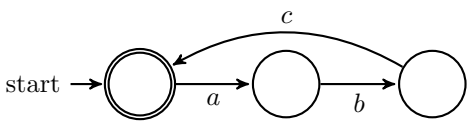

Figure 1: Finite Machine M.

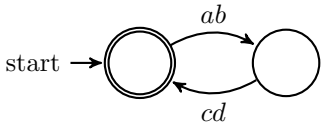

Figure 2: General Finite Machine M'.

2.3. Discussion of Related Concepts

It is hard to trace back all origins and names of the concepts introduced so far. We only mention a few of these sources in this subsection, also to facilitate finding the names of the concepts and understanding the connections to other parts of mathematics and computer science. This subsection is not meant to be a survey on all the neighboring areas. Rather, it should give some impression about the richness of interrelations.

- Shuffle expressions have been introduced and studied to understand the semantics of parallel programs. This was undertaken, as it appears to be, independently by Campbell and Habermann [6], by Mazurkiewicz [64] and by Shaw [78]. These expressions (also known as *flow expressions*) allow for sequential operators (catenation and iterated catenation) as well as for parallel operators (shuffle and iterated shuffle). In view of the results obtained in this paper, let us only highlight one particular result for these flow expressions. The universality problem was shown to be undecidable for such expressions, even when restricted to binary alphabets; see [40].

- Further on, different variants of shuffle expressions in general were studied; see, e. g., [23, 43, 44, 62]. We only mention SHUF as a subclass of Shuffle Expressions. Actually, even the α-SHUF expressions that we introduced in [19] have been considered as a special case before; the corresponding class of languages was termed $\mathcal{L}_3$ in [35], possibly a bit confusing given the traditional meaning of the term $\mathcal{L}_3$ for the regular languages.

- Semilinear subsets of $\mathbb{N}^n$ show up quite naturally in many branches of mathematics. In our context of Theoretical Computer Science, the first important ingredient was the paper *Semigroups, Presburger formulas, and*

languages [27] whose title pretty much spells out the connections between logic and formal languages.[2]

- Eilenberg and Schützenberger started in [13] the study of subsets (languages) of the free commutative monoid of some given alphabet, relating this again to earlier studies of Ginsburg and Spanier [26] on bounded languages, which are kind of natural representatives of the permutation equivalence classes. With the notions given in [13] for the definition of rational sets on the level of commutative monoids, if we replace the $+$ by $\sqcup\!\sqcup$ and Kleene star by iterated shuffle, then we basically arrive at the α-SHUF expressions that we introduced in [19]. The studies of [13] were later continued, e. g., by Huynh [36, 37].

- Relations to blind counter automata become obvious if one considers the way that JFAs process the input, basically only counting occurrences of different symbols. The connections to semilinear sets and to Petri nets were already discussed by Greibach [31]; also, see [17, 74]. Even more general yet related structures are studied in [21, 68, 82]. The main formal difference is that with JFA, the input is not processed continuously, while all devices mentioned in this paragraph do process the input in a continuous manner, although the way that the (counter) storage can be operated allows these devices to incorporate some jump-like features.

- Recently, Křivka and Meduna [55] studied *jumping grammars* and also showed two variants of regular grammars that characterize $\mathcal{JFA}$ and $\mathcal{GJFA}$. This type of grammar needs to be further compared to different varieties of commutative grammars. We only mention [15, 37, 69, 70].

3. Basic Algebraic Properties of Shuffle and Permutation

In this section, we state some basic (algebraic) properties of the shuffle and permutation operations. To this end, we first recall the following computation rules for the shuffle operator from [41].

Proposition 10. *Let M_1, M_2, M_3 be arbitrary languages.*

1. $M_1 \sqcup\!\sqcup M_2 = M_2 \sqcup\!\sqcup M_1$ *(commutative law),*
2. $(M_1 \sqcup\!\sqcup M_2) \sqcup\!\sqcup M_3 = M_1 \sqcup\!\sqcup (M_2 \sqcup\!\sqcup M_3)$ *(associative law),*
3. $M_1 \sqcup\!\sqcup (M_2 \cup M_3) = M_1 \sqcup\!\sqcup M_2 \cup M_1 \sqcup\!\sqcup M_3$ *(distributive law),*
4. $(M_1 \cup M_2)^{\sqcup\!\sqcup,*} = (M_1)^{\sqcup\!\sqcup,*} \sqcup\!\sqcup (M_2)^{\sqcup\!\sqcup,*}$,
5. $(M_1{}^{\sqcup\!\sqcup,*})^{\sqcup\!\sqcup,*} = (M_1)^{\sqcup\!\sqcup,*}$,
6. $(M_1 \sqcup\!\sqcup M_2{}^{\sqcup\!\sqcup,*})^{\sqcup\!\sqcup,*} = (M_1 \sqcup\!\sqcup (M_1 \cup M_2)^{\sqcup\!\sqcup,*}) \cup \{\varepsilon\}$.

The second, third and fifth rule are also true for (iterated) catenation instead of (iterated) shuffle. This is no coincidence, as we will see. We can deduce from the first three computation rules the following result.

[2] Interestingly enough, Presburger's original work *Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt* has also a title that basically summarizes the contents of the paper.

Proposition 11. $(2^{\Sigma^*}, \cup, \sqcup\!\sqcup, \emptyset, \{\varepsilon\})$ *is a commutative semiring.*

Proof. Instead of giving a complete formal argument, let us mostly recall what needs to be shown, giving then appropriate hints. First, $(2^{\Sigma^*}, \cup, \emptyset)$ is a commutative monoid; this is a well-known set-theoretic statement. Second, $(2^{\Sigma^*}, \sqcup\!\sqcup, \{\varepsilon\})$ is a commutative monoid; this corresponds to the first two computation rules, plus the fact that $\{\varepsilon\}$ is the neutral element with respect to the shuffle operation. Third, the distributive law was explicitly stated as the third computation rule. □

We are now discussing some special properties of the operator perm from a different (algebraic) viewpoint. Reminiscent of the presentation in [20], there is an alternative way of looking at the permutation operator. Namely, let $w \in \Sigma^n$ be a word of length n, spelled out as $w = a_1 \cdots a_n$ for $a_i \in \Sigma$. Then, $u \in \mathrm{perm}(w)$ if and only if there exists a bijection $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $u = a_{\pi(1)} \cdots a_{\pi(n)}$. In combinatorics, such bijections are also known as permutations. This also shows that $|\mathrm{perm}(w)| \leq (|w|)!$.

Next, we summarize two important properties of the operator perm in the following two lemmas.

Lemma 12. $\mathrm{perm} : 2^{\Sigma^*} \to 2^{\Sigma^*}$ *is a hull operator, i. e., it is extensive (i. e.,* $L \subseteq \mathrm{perm}(L)$*), increasing (i. e., if* $L_1 \subseteq L_2$*, then* $\mathrm{perm}(L_1) \subseteq \mathrm{perm}(L_2)$*), and idempotent (i. e.,* $\mathrm{perm}(\mathrm{perm}(L)) = \mathrm{perm}(L)$*).*

Due to the well-known correspondence between hull operators and (systems of) closed sets, we will also speak about *perm-closed languages* in the following, i. e., languages L satisfying $\mathrm{perm}(L) = L$. Such languages are also called *commutative*, see [56].

Lemma 13. *Let* $\Sigma = \{a_1, \ldots, a_n\}$*. The set* $\{\mathrm{perm}(w) : w \in \Sigma^*\}$ *is a partition of* Σ^**. There is a natural bijection between this partition and* $\mathbb{N}^{|\Sigma|}$*, given by the Parikh mapping* $\psi_\Sigma : \Sigma^* \to \mathbb{N}^{|\Sigma|}, w \mapsto (|w|_{a_1}, \ldots, |w|_{a_n})$*. Namely,* $\mathrm{perm}(w) = \psi_\Sigma^{-1}(\psi_\Sigma(w))$ *for* $w \in \Sigma^*$*.*

Note that there exists a possibly better known semiring in formal language theory, using catenation instead of shuffle; let us make this explicit in the following statement.

Proposition 14. $(2^{\Sigma^*}, \cup, \cdot, \emptyset, \{\varepsilon\})$ *is a semiring that is not commutative whenever* $|\Sigma| > 1$*.*

Another algebraic interpretation can be given as follows.

Proposition 15. *Parikh mappings can be interpreted as semiring morphisms from* $(2^{\Sigma^*}, \cup, \sqcup\!\sqcup, \emptyset, \{\varepsilon\})$ *to* $(2^{\mathbb{N}^{|\Sigma|}}, \cup, +, \emptyset, \{\vec{0}\})$*, where* $\vec{0}$ *is the tuple with* $|\Sigma|$ *zeros.*

Especially, we conclude:

Proposition 16. *For* $u, v \in \Sigma^*$*,* $\mathrm{perm}(u) = \mathrm{perm}(v)$ *if and only if* $\psi_\Sigma(u) = \psi_\Sigma(v)$*. For* $L_1, L_2 \subseteq \Sigma^*$*,* $\mathrm{perm}(L_1) = \mathrm{perm}(L_2)$ *if and only if* $\psi_\Sigma(L_1) = \psi_\Sigma(L_2)$*.*

Due to Proposition 16, we can call $u, v \in \Sigma^*$ (and also $L_1, L_2 \subseteq \Sigma^*$) *permutation-equivalent* or *Parikh-equivalent* if $\psi_\Sigma(u) = \psi_\Sigma(v)$ ($\psi_\Sigma(L_1) = \psi_\Sigma(L_2)$, respectively).

The relation between (iterated) catenation and (iterated) shuffle can now be neatly expressed as follows.

Theorem 17. perm : $2^{\Sigma^*} \to 2^{\Sigma^*}$ *is a semiring morphism from the semiring* $(2^{\Sigma^*}, \cup, \cdot, \emptyset, \{\varepsilon\})$ *to the semiring* $(2^{\Sigma^*}, \cup, \sqcup, \emptyset, \{\varepsilon\})$ *that also respects the iterated catenation resp. iterated shuffle operation.*

Clearly, perm cannot be an isomorphism, as the catenation semiring is not commutative, while the shuffle semiring is, see Proposition 11.

The proof of the previous theorem, broken into several statements that are also interesting in their own right, is presented in the following. Notice that in the terminology of Ésik and Kuich [14], Theorem 17 can also be stated as follows: perm : $2^{\Sigma^*} \to 2^{\Sigma^*}$ is a starsemiring morphism from the starsemiring $(2^{\Sigma^*}, \cup, \cdot, ^*, \emptyset, \{\varepsilon\})$ to the starsemiring $(2^{\Sigma^*}, \cup, \sqcup, ^{\sqcup,*}, \emptyset, \{\varepsilon\})$. Some of the required properties are also listed in [60] (without giving a proof).

Lemma 18. $\forall u, v \in \Sigma^*$*:* $\mathrm{perm}(u \cdot v) = \mathrm{perm}(u) \sqcup \mathrm{perm}(v)$.

Proof. We prove this lemma by induction on $|u|$. Induction Basis: $|u| = 1$. So, $u \in \Sigma$. By Definition 3, $\mathrm{perm}(u \cdot v) = \{u\} \sqcup \mathrm{perm}(v) = \mathrm{perm}(u) \sqcup \mathrm{perm}(v)$. Induction Hypothesis: For $u \in \Sigma^n$, $\mathrm{perm}(u \cdot v) = \mathrm{perm}(u) \sqcup \mathrm{perm}(v)$. Induction Step: Consider $|u| = n + 1$. Let $u = x_1 x_2 \dots x_{n+1},\ x_i \in \Sigma$. We now claim that $\mathrm{perm}(x_1 x_2 \dots x_{n+1} \cdot v) = \mathrm{perm}(x_1 x_2 \dots x_{n+1}) \sqcup \mathrm{perm}(v)$.

$$\begin{aligned}
\mathrm{perm}(x_1 x_2 \dots x_{n+1} \cdot v) &= \{x_1\} \sqcup \mathrm{perm}(x_2 \dots x_{n+1} \cdot v) \text{ (by Definition 3)} \\
&= \{x_1\} \sqcup \mathrm{perm}(x_2 \dots x_{n+1}) \sqcup \mathrm{perm}(v) \text{ (IH)} \\
&= \mathrm{perm}(x_1 x_2 \dots x_{n+1}) \sqcup \mathrm{perm}(v) \text{ (by Definition 3).}
\end{aligned}$$

Therefore, $\mathrm{perm}(u \cdot v) = \mathrm{perm}(u) \sqcup \mathrm{perm}(v)$. □

Lemma 19. $\forall u, v \in \Sigma^*$*:* $u \sqcup v \subseteq \mathrm{perm}(u \cdot v)$.

Proof. By Definition 1, $u \sqcup v = \{x_1 y_1 x_2 y_2 \dots x_n y_n : u = x_1 x_2 \dots x_n, v = y_1 y_2 \dots y_n, x_i, y_i \in \Sigma^*, 1 \le i \le n,\ n \ge 1\}$. It is clear that $u \sqcup v \subseteq \mathrm{perm}(u) \sqcup \mathrm{perm}(v)$, as perm is a hull operator, see Lemma 12. According to Lemma 18 $\mathrm{perm}(u) \sqcup \mathrm{perm}(v) = \mathrm{perm}(u \cdot v)$. Therefore, $u \sqcup v \subseteq \mathrm{perm}(u \cdot v)$. □

As a consequence of Lemma 19 and since perm is a hull operator, we obtain the following lemma.

Lemma 20. $\forall u, v \in \Sigma^*$*:* $\mathrm{perm}(u \sqcup v) = \mathrm{perm}(u \cdot v) = \mathrm{perm}(u) \sqcup \mathrm{perm}(v)$.

Proof. As indicated, from $u \sqcup v \subseteq \mathrm{perm}(u \cdot v)$ we conclude that $\mathrm{perm}(u \sqcup v) \subseteq \mathrm{perm}(\mathrm{perm}(u \cdot v)) = \mathrm{perm}(u \cdot v)$. Conversely, as $\{u \cdot v\} \subseteq u \sqcup v$, $\mathrm{perm}(u \cdot v) \subseteq \mathrm{perm}(u \sqcup v)$. Hence, $\mathrm{perm}(u \sqcup v) = \mathrm{perm}(u \cdot v)$. Now this together with Lemma 18 gives $\mathrm{perm}(u \sqcup v) = \mathrm{perm}(u) \sqcup \mathrm{perm}(v)$. □

Lemma 21. *Let* $L, L_1, L_2 \subseteq \Sigma^*$. *Then*

1. $\mathrm{perm}(L^{n+1}) = \mathrm{perm}(L^n \sqcup L)$,
2. $\mathrm{perm}(L_1) \sqcup \mathrm{perm}(L_2) = \mathrm{perm}(L_1 \sqcup L_2) = \mathrm{perm}(L_1 \cdot L_2)$, *and*
3. $(\mathrm{perm}(L))^{\sqcup,*} = \mathrm{perm}(L^{\sqcup,*}) = \mathrm{perm}(L^*)$.

Proof. We are going to prove three parts separately.

1. The inclusion $\mathrm{perm}(L^{n+1}) \subseteq \mathrm{perm}(L^n \sqcup\!\sqcup L)$ is true, since $L^{n+1} \subseteq L^n \sqcup\!\sqcup L$. We now prove the other inclusion $\mathrm{perm}(L^{n+1}) \supseteq \mathrm{perm}(L^n \sqcup\!\sqcup L)$. Let $w \in L^n \sqcup\!\sqcup L$, then $\exists u \in L^n, v \in L : w \in u \sqcup\!\sqcup v$. This implies that $\exists u \in L^n, v \in L : w \in \mathrm{perm}(u \cdot v)$ by Lemma 19. Therefore, $\mathrm{perm}(L^{n+1}) = \mathrm{perm}(L^n \sqcup\!\sqcup L)$.
2. Lemma 20 immediately shows that the second equality holds. Consider $L_1 \subseteq \Sigma^*$, $L_2 \subseteq \Sigma^*$. Let $w \in \mathrm{perm}(L_1) \sqcup\!\sqcup \mathrm{perm}(L_2)$. Let $x' \in \mathrm{perm}(L_1)$, $y' \in \mathrm{perm}(L_2)$ such that $w \in x' \sqcup\!\sqcup y'$. Hence, there exists some $x \in L_1$ such that $x' \in \mathrm{perm}(x)$ (also $x \in \mathrm{perm}(x')$). Likewise, there exists some $y \in L_2$ with $y' \in \mathrm{perm}(y)$. Hence, $w \in \mathrm{perm}(x) \sqcup\!\sqcup \mathrm{perm}(y) = \mathrm{perm}(x \sqcup\!\sqcup y)$ by Lemma 20. Therefore, $w \in \mathrm{perm}(L_1 \sqcup\!\sqcup L_2) \cap \mathrm{perm}(L_1 \cdot L_2)$. Similarly, if $w \in \mathrm{perm}(L_1 \sqcup\!\sqcup L_2)$ then $w \in \mathrm{perm}(L_1) \sqcup\!\sqcup \mathrm{perm}(L_2)$. Hence, $\mathrm{perm}(L_1) \sqcup\!\sqcup \mathrm{perm}(L_2) = \mathrm{perm}(L_1 \sqcup\!\sqcup L_2)$.
3. We will prove $(\mathrm{perm}(L))^{\sqcup\!\sqcup,n} = \mathrm{perm}(L^{\sqcup\!\sqcup,n})$ and $\mathrm{perm}(L^{\sqcup\!\sqcup,n}) = \mathrm{perm}(L^n)$ by induction on n.
   Induction Basis: $(\mathrm{perm}(L))^{\sqcup\!\sqcup,0} = \{\varepsilon\} = \mathrm{perm}(\varepsilon) = \mathrm{perm}(L^{\sqcup\!\sqcup,0})$.
   Induction Hypothesis: $(\mathrm{perm}(L))^{\sqcup\!\sqcup,n} = \mathrm{perm}(L^{\sqcup\!\sqcup,n})$.
   Induction Step: We now claim that $(\mathrm{perm}(L))^{\sqcup\!\sqcup,n+1} = \mathrm{perm}(L^{\sqcup\!\sqcup,n+1})$.

$$\begin{aligned}(\mathrm{perm}(L))^{\sqcup\!\sqcup,n+1} &= (\mathrm{perm}(L))^{\sqcup\!\sqcup,n} \sqcup\!\sqcup \mathrm{perm}(L) \text{ (By Definition 4)}\\ &= \mathrm{perm}(L^{\sqcup\!\sqcup,n}) \sqcup\!\sqcup \mathrm{perm}(L) \text{ (By Induction Hypothesis)}\\ &= \mathrm{perm}(L^{\sqcup\!\sqcup,n} \sqcup\!\sqcup L) \text{ (By (2) in Lemma 21)}\\ &= \mathrm{perm}(L^{\sqcup\!\sqcup,n+1}) \text{ (By Definition 4).}\end{aligned}$$

   We now prove $\mathrm{perm}(L^{\sqcup\!\sqcup,n-1}) = \mathrm{perm}(L^{n-1})$ by induction on n.
   Induction Basis: $\mathrm{perm}(L^{\sqcup\!\sqcup,0}) = \mathrm{perm}(\varepsilon) = \{\varepsilon\} = \mathrm{perm}(\varepsilon) = \mathrm{perm}(L^0)$.
   Induction Hypothesis: $\mathrm{perm}(L^{\sqcup\!\sqcup,n}) = \mathrm{perm}(L^n)$.
   Induction Step: We now claim that $\mathrm{perm}(L^{\sqcup\!\sqcup,n+1}) = \mathrm{perm}(L^{n+1})$.

$$\begin{aligned}\mathrm{perm}(L^{\sqcup\!\sqcup,n+1}) &= \mathrm{perm}(L^{\sqcup\!\sqcup,n} \sqcup\!\sqcup L) \text{ (By Definition 4)}\\ &= \mathrm{perm}(L^{\sqcup\!\sqcup,n}) \sqcup\!\sqcup \mathrm{perm}(L) \text{ (By (2) in Lemma 21)}\\ &= \mathrm{perm}(L^n) \sqcup\!\sqcup \mathrm{perm}(L) \text{ (By Induction Hypothesis)}\\ &= \mathrm{perm}(L^n \sqcup\!\sqcup L) \text{ (By (2) in Lemma 21)}\\ &= \mathrm{perm}(L^{n+1}) \text{ (By (1) Lemma 21).}\end{aligned}$$

   By the very definitions of iterated catenation (Kleene star) and iterated shuffle, the claim of the second part follows. □

Proof of Theorem 17

Proof. Recall that perm, in order to be a semiring morphism, should satisfy the following properties: (i) By an easy standard set-theoretic argument we get $\forall L_1, L_2 \subseteq \Sigma^* : \mathrm{perm}(L_1 \cup L_2) = \mathrm{perm}(L_1) \cup \mathrm{perm}(L_2)$, (ii) By Lemma 21 $\forall L_1, L_2 \subseteq \Sigma^* : \mathrm{perm}(L_1 \cdot L_2) = \mathrm{perm}(L_1) \sqcup\!\sqcup \mathrm{perm}(L_2)$, and (iii) $\mathrm{perm}(\emptyset) = \emptyset$ and $\mathrm{perm}(\{\varepsilon\}) = \{\varepsilon\}$ are trivial claims. Furthermore, we claim an according preservation property for the iterated catenation resp. shuffle, which is explicitly stated and proven in Lemma 21. □

Remark 22. Let us make some further algebraic consequences explicit.
(i) $\mathrm{perm}(L)$ can be seen as the canonical representative of all languages $\tilde{L}$ that

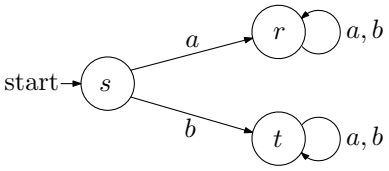

Figure 3: An example JFA, final states not specified.

are permutation-equivalent to L.
(ii) As perm is a morphism, there is in fact a semiring isomorphism between the permutation-closed languages (over Σ) and $\mathbb{N}^{|\Sigma|}$, which is basically a Parikh mapping in this case.
(iii) There is a further natural isomorphism between the monoid $(\mathbb{N}^{|\Sigma|}, +, \vec{0})$ and the free commutative monoid generated by Σ.

4. The Language Class $\mathcal{JFA}$

By the definition of a jumping finite automaton M, it is clear that $w \in L_{\mathrm{JFA}}(M)$ implies that $\mathrm{perm}(w) \subseteq L_{\mathrm{JFA}}(M)$, i. e., $\mathrm{perm}(L_{\mathrm{JFA}}(M)) \subseteq L_{\mathrm{JFA}}(M)$. Since perm is extensive as a hull operator (see Lemma 12), we can conclude:

Corollary 23. *If $L \in \mathcal{JFA}$, then L is perm-closed.*

This also follows from results of [65]. In particular, we mention the following important characterization theorem from [66], that we enrich by combining it with the well-known theorem of Parikh [73] using Proposition 16.

Theorem 24. $\mathcal{JFA} = \mathrm{perm}(\mathcal{RES}) = \mathrm{perm}(\mathcal{CFL}) = \mathrm{perm}(\mathcal{PSL})$.

This theorem also generalizes the main result of [57]. It also indicates that certain properties of this language class have been previously derived under different names; for instance, Latteux [56] writes $\mathcal{JFA}$ as c(RAT), and he mentions yet another characterization for this class in the literature, which is the class of all perm-closed languages whose Parikh image is semilinear; as explained in Section 2.1, the class of languages whose Parikh image is semilinear is also known as *slip languages* [28], or $\mathcal{PSL}$ for short. Due to Lemma 13, there is a natural bijection between $\mathcal{JFA}$ and the recognizable subsets of the monoid $(\mathbb{N}^{|\Sigma|}, +, \vec{0})$.

Let us mention one corollary that can be deduced from these connections; for proofs, we refer to [13, 27].

Corollary 25. *$\mathcal{JFA}$ is closed under intersection and under complementation.*

Notice that the proof given in Theorem 17.4.6 in [66] is wrong, as the nondeterminism inherent in JFAs due to the jumping feature is neglected. For instance, consider the deterministic[3] JFA $M = (\{r, s, t\}, \{a, b\}, R, \{s\}, F)$ with rules according to Figure 3. If $F = \{r\}$, then M accepts all words that contain at least one a. But, if $F = \{s, t\}$, then M accepts ε and all words that contain

[3] According to [66], a JFA is *deterministic* if each state has exactly one outgoing transition for each letter.

at least one b. This clearly shows that the standard state complementation technique does not work for JFAs.

As we will be concerned later also with descriptional and computational complexity issues, let us mention here that, according to the analysis indicated in [16], Parikh's original proof would produce, starting from a context-free grammar G with n variables, a regular expression E of length $\mathcal{O}\left(2^{2^{n^2}}\right)$ such that $\mathrm{perm}(L(G)) = \mathrm{perm}(L(E))$, whose corresponding NFA is even bigger, while the construction of [16] results in an NFA M with only 4^n states, satisfying $\mathrm{perm}(L(G)) = \mathrm{perm}(L_{\mathrm{FA}}(M))$. In the context of this theorem, it is also interesting to note that recently there have been investigations on the descriptional complexity of converting context-free grammars into Parikh-equivalent finite automata, see [58]. The theorem also links JFAs to the literature on "commutative context-free languages", e. g., [3, 50].

Also, a sort of normal forms for language classes $\mathcal{L}$ such that $\mathrm{perm}(\mathcal{L}) = \mathcal{JFA}$ have been studied, for instance, the class $\mathcal{L}$ of letter-bounded languages can be characterized in various ways, see [5, 7, 38] for a kind of survey.

Since finite languages are regular, we can conclude the following corollary of Theorem 24.

Corollary 26. *Let L be a finite language. Then, $L \in \mathcal{JFA}$ if and only if L is perm-closed.*

This also shows that all finite JFA languages are so-called commutative regular languages as studied by Ehrenfeucht, Haussler and Rozenberg in [12]. We will come back to this issue later.

Next, we shall show that $\mathcal{JFA}$ coincides with the class of α-SHUF expressions. To this end, we first observe that a regular expression E can be easily turned into an α-SHUF expression describing $\mathrm{perm}(L(E))$ by replacing catenations and Kleene stars with shuffles and iterated shuffles (this is a direct consequence of the fact that the perm operator is a semiring morphism as stated in Theorem 17).

Lemma 27. *Let R' be a regular expression. Let the α-SHUF expression R be obtained from R' by consequently replacing all $\cdot$ by $\sqcup\!\sqcup$, and all * by $^{\sqcup\!\sqcup,*}$ in R'. Then,* $\mathrm{perm}(L(R')) = L(R)$.

Proof. Let R' be a regular expression. By definition, this means that $L(R') = K$, where K is some expression over the languages $\emptyset$, $\{\varepsilon\}$ and $\{a\}$, $a \in \Sigma$, using only union, catenation and Kleene-star. By Theorem 17, $\mathrm{perm}(K)$ can be transformed into an equivalent expression K' using only union, shuffle and iterated shuffle. Furthermore, in K', the operation perm only applies to languages of the form $\emptyset$, $\{\varepsilon\}$ and $\{a\}$, $a \in \Sigma$, which means that by simply removing all perm operators, we obtain an equivalent expression K'' over languages $\emptyset$, $\{\varepsilon\}$ and $\{a\}$, $a \in \Sigma$, using only union, shuffle and iterated shuffle. This expression directly translates into the α-SHUF expression R with $L(R) = \mathrm{perm}(L(R'))$. □

We are now ready to prove our characterization theorem for $\mathcal{JFA}$.

Theorem 28. *A language $L \subseteq \Sigma^*$ is in $\mathcal{JFA}$ if and only if there is some α-SHUF expression R such that $L = L(R)$.*

Proof. If $L \in \mathcal{JFA}$, then there exists a regular language L' such that $L = \mathrm{perm}(L')$ by Theorem 24. L' can be described by some regular expression R'. By Lemma 27, we find an α-SHUF expression R with $L = \mathrm{perm}(L(R')) = L(R)$.

Conversely, if L is described by some α-SHUF expression R, i. e., $L = L(R)$, then construct the regular expression R' by consequently replacing all $\sqcup\!\sqcup$ by $\cdot$ and all $^{\sqcup\!\sqcup,*}$ by * in R. Clearly, we face the situation described in Lemma 27, so that we conclude that $\mathrm{perm}(L(R')) = L(R) = L$. As $L(R')$ is a regular language, $\mathrm{perm}(L(R')) = L \in \mathcal{JFA}$ by Theorem 24. □

Since α-SHUF languages are closed under iterated shuffle, we obtain the following corollary as a consequence of Theorem 28, adding to the list of closure properties given in [65].

Corollary 29. *$\mathcal{JFA}$ is closed under iterated shuffle.*

We like to point out again the connections to regular expressions over commutative monoids as studied in [13].

Let us finally mention a second characterization of the finite perm-closed sets in terms of α-SHUF expressions (recall that Corollary 26 states the first such characterization).

Proposition 30. *Let L be a language. Then, L is finite and perm-closed if and only if there is an α-SHUF expression R, with $L = L(R)$, that does not contain the iterated shuffle operator.*

Proof. Let L be a finite language with $L = \mathrm{perm}(L)$. Clearly, there is a regular expression R_L, with $L(R_L) = L$, that uses only the catenation and union operations. As L is perm-closed, the α-SHUF expression R obtained from R_L by replacing all catenation by shuffle operators satisfies $L(R) = \mathrm{perm}(L(R_L)) = L$ by Lemma 27 and does not contain the iterated shuffle operator. Conversely, let R be an α-SHUF expression that does not contain the iterated shuffle operator. By combining Theorem 28 with Corollary 23, we know that $L(R)$ is perm-closed. It is rather straightforward that $L(R)$ is also finite. □

5. The Language Classes $\mathcal{GJFA}$ and $\mathcal{SHUF}$

In the last section, we saw that JFA and α-SHUF expressions correspond to each other in a very similar way as classical regular expressions correspond to finite automata. More precisely, in the translation between α-SHUF expressions and JFA, the atoms of the α-SHUF expression will become the labels of the JFA and vice versa.

GJFAs differ from JFAs only in that the labels can be arbitrary words instead of symbols and, similarly, SHUF expressions differ from α-SHUF expressions only in that the atoms can be arbitrary words. This suggests that a similar translation between GJFAs and SHUF expressions exists and, thus, these devices describe the same class of languages. Unfortunately, this is not the case, which can be demonstrated with a simple example: let $M = (\{s\}, \{a, b, c, d\}, \{sab \to s, scd \to s\}, s, \{s\})$ be a GJFA, which naturally translates into the SHUF expression $E = (ab + cd)^{\sqcup\!\sqcup,*}$. It can be easily verified that every word that is accepted by M can also be generated by E, but, as $acbd \in (L(E) \setminus L_{\mathrm{JFA}}(M))$, we have $L_{\mathrm{JFA}}(M) \subsetneq L(E)$.

In the following, we shall see that not only this naive translation between GJFA and SHUF expressions fails, but the language classes $\mathcal{GJFA}$ and $\mathcal{SHUF}$ are incomparable.

Lemma 31. *Let* $M = (\{s\}, \{a, b, c, d\}, \{sab \to s, scd \to s\}, s, \{s\})$. *Then,* $L(M)$ *is not a* SHUF *language.*

Proof. For the sake of contradiction, let E be a SHUF expression with $L(E) = L(M)$. As the number of occurrences of both a and d are unbounded in words from $L(M)$, one of the two cases must hold:

Case 1: E contains a subexpression $(R)^{\sqcup\!\sqcup,*}$ such that there exists a $w \in L(R)$ with $|w|_a \geq 1$ and $|w|_d \geq 1$.

Case 2: E contains a subexpression $R_1 \sqcup\!\sqcup R_2$ such that there exists a $w \in L(R_1)$ with $|w|_a \geq 1$ and a $w' \in L(R_2)$ with $|w|_d \geq 1$.

Both cases imply that $L(E)$ contains a word with factor ad. This is a contradiction, since such words are not in $L_{\mathrm{JFA}}(M)$. □

Lemma 32. *Let* $L = L(ac \sqcup\!\sqcup (bd)^{\sqcup\!\sqcup,*})$. L *is not accepted by any GJFA.*

Proof. For the sake of contradiction, assume that L is accepted by a GJFA M. Let n be greater than the maximum length of a transition label in M and let $w = ab^n cd^n$. The accepting computation of M on w uses exactly one transition with a label u that contains c.

- If $u = b^i cd^j$ for $i, j \geq 0$, all earlier transitions only consume factors that are completely contained in the prefix ab^{n-i} or the suffix d^{n-j} of $w = ab^{n-i}(b^i cd^j)d^{n-j}$. This implies that, by using the same sequence of transitions, M can accept $w' = b^i cd^j ab^{n-i}d^{n-j}$.
- Otherwise, $u = ab^r cd^s$ for $r, s \geq 0$, i. e., it contains both a and c. By the choice of n, an earlier transition labeled with b^k with $k > 0$ was used. However, this implies that also $w'' = ab^{n-k}cd^n b^k$ is accepted by M.

The case of $w' \in L(M)$ violates the condition that the symbol a precedes c in words from L, while the case of $w'' \in L_{\mathrm{JFA}}(M)$ contradicts the fact that the words in L do not end with b. □

Lemma 33. $\{ab\}^{\sqcup\!\sqcup,*} \in (\mathcal{GJFA} \cap \mathcal{SHUF}) \setminus \mathcal{JFA}$.

Proof. Obviously, $\{ab\}^{\sqcup\!\sqcup,*} = L((ab)^{\sqcup\!\sqcup,*})$. Furthermore, $\{ab\}^{\sqcup\!\sqcup,*} = L_{\mathrm{JFA}}(M)$, where M is the GJFA with a single state s, which is both initial and final, and a single rule $sab \to s$. As $ab \in \{ab\}^{\sqcup\!\sqcup,*}$, but $ba \notin \{ab\}^{\sqcup\!\sqcup,*}$, $\{ab\}^{\sqcup\!\sqcup,*}$ is not perm-closed, and hence not a JFA language. □

It is interesting to note that if we take the permutation closures of the separating languages from Lemmas 31 and 32, then we get JFA languages. As shall be demonstrated next (see Theorem 34), this property holds for all SHUF and GJFA languages.

Theorem 34. $\mathrm{perm}(\mathcal{GJFA}) = \mathrm{perm}(\mathcal{SHUF}) = \mathrm{perm}(\mathcal{PSL}) = \mathcal{JFA}$.

Proof. Let us prove that $\mathrm{perm}(\mathcal{GJFA}) \cup \mathrm{perm}(\mathcal{SHUF}) = \mathcal{JFA}$. Clearly $\mathcal{JFA} \subseteq \mathrm{perm}(\mathcal{GJFA}) \cup \mathrm{perm}(\mathcal{SHUF})$ and we only have to show that $\mathrm{perm}(\mathcal{GJFA}) \cup \mathrm{perm}(\mathcal{SHUF}) \subseteq \mathcal{JFA}$.

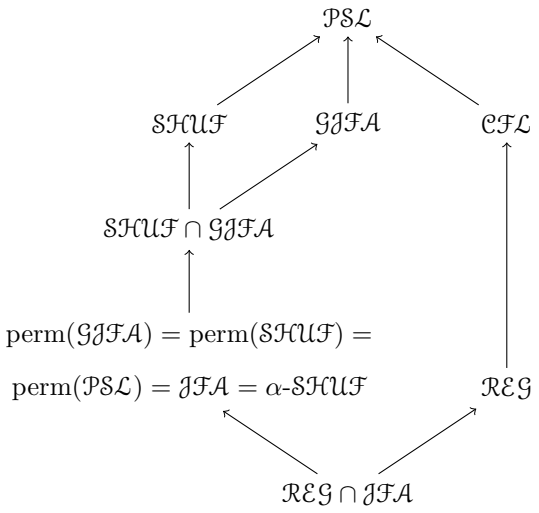

Figure 4: Inclusion diagram of our language families.

Case 1: Let $L \in \mathcal{SHUF}$ be described by a SHUF expression X. Then $\mathrm{perm}(L)$ is described by the α-SHUF expression X' that is obtained from X by replacing each atomic word $a_1 \cdots a_n \in \Sigma^*$ of length $n \geq 2$ by the α-SHUF subexpression $a_1 \sqcup\!\sqcup \cdots \sqcup\!\sqcup a_n$. The fact that $\mathrm{perm}(L) = \mathrm{perm}(L(X)) = L(X')$ follows by an easy induction argument using Theorem 17.

Case 2: Let $L \in \mathcal{GJFA}$. The well-known construction of a finite automaton that simulates a given general finite automaton can be applied to obtain, from a given GJFA M, a JFA M' with the property $\mathrm{perm}(L_{\mathrm{JFA}}(M)) = L_{\mathrm{JFA}}(M')$. The correctness of this method immediately follows from our reasoning towards Theorem 24. In both the cases we conclude that $\mathrm{perm}(L)$ lies in $\mathcal{JFA}$.

Let $L \subseteq \Sigma^*$. Then, the following three claims are equivalent: (i) $L \in \mathcal{JFA}$, (ii) L is perm-closed and $L \in \mathcal{GJFA}$, and (iii) L is perm-closed and $L \in \mathcal{SHUF}$.

As each $L \in \mathcal{JFA}$ is perm-closed and in $\mathcal{GJFA} \cap \mathcal{SHUF}$, we only have to show the upward implications. If $L \in \mathcal{GJFA}$, then by Case 2, $\mathrm{perm}(L) \in \mathcal{JFA}$. If, in addition, L is perm-closed, then $\mathrm{perm}(L) = L$, which shows the claim. Similarly, we can show that, if L is perm-closed and $L \in \mathcal{SHUF}$, then $L \in \mathcal{JFA}$. □

We summarize the inclusion relations between the language families considered in this paper in Figure 4. In this figure, an arrow from class A to B represents the strict inclusion $A \subsetneq B$. A missing connection between a pair of language families means incomparability.

Theorem 35. *The inclusion and incomparability relations displayed in Figure 4 are correct.*

Proof. We first show the correctness of the subset relations. The class $\mathcal{REG} \cap \mathcal{JFA}$ is obviously included in both $\mathcal{REG}$ and $\mathcal{JFA}$, and any non-commutative regular language and the non-regular JFA language $L = \{w \in \{a, b\}^* : |w|_a = |w|_b\}$ show these subset relations to be proper. That $\mathcal{JFA} \subsetneq \mathcal{SHUF} \cap \mathcal{GJFA}$ follows by definition and Lemma 33. Similarly, both $\mathcal{SHUF} \cap \mathcal{GJFA} \subsetneq \mathcal{GJFA}$ and $\mathcal{SHUF} \cap \mathcal{GJFA} \subsetneq \mathcal{SHUF}$ follows by definition and Lemmas 31 and 32, respectively.

Theorem 34 together with Lemma 12 (as the operator perm is extensive) shows that the classes $\mathcal{SHUF}$ and $\mathcal{GJFA}$ are contained in $\mathcal{PSL}$. Namely, if this would not be the case, then there should be a language L, say, in $\mathcal{GJFA} \setminus \mathcal{PSL}$. Now, $\mathrm{perm}(L) \in \mathrm{perm}(\mathcal{GJFA}) \subseteq \mathcal{PSL}$, but $L \in \mathcal{PSL}$ if and only if $\mathrm{perm}(L) \in \mathcal{PSL}$ by definition of $\mathcal{PSL}$, yielding a contradiction. Similarly, there should be a language L, say, in $\mathcal{SHUF} \setminus \mathcal{PSL}$. Now, $\mathrm{perm}(L) \in \mathrm{perm}(\mathcal{SHUF}) \subseteq \mathcal{PSL}$, but $L \in \mathcal{PSL}$ if and only if $\mathrm{perm}(L) \in \mathcal{PSL}$, again yielding a contradiction. Since $\mathcal{SHUF}$ and $\mathcal{GJFA}$ are incomparable, these two inclusions are proper.

By Parikh's theorem [73] and as the context-free languages do not contain the language studied in Example 8, $\mathcal{CFL} \subsetneq \mathcal{PSL}$. Finally, $\mathcal{REG} \subsetneq \mathcal{CFL}$ is well-known; thus, all the claimed proper subset relations hold.

Since $\mathcal{JFA}$ is a proper superclass of $\mathcal{REG} \cap \mathcal{JFA}$, it contains a language not in $\mathcal{REG}$. Furthermore, according to [66, Lemma 17.3.2], the regular language $\{a\}^*\{b\}^*$ is not in $\mathcal{GJFA}$. By a similar argument as used in the proof of Lemma 31, it can also be shown that $\{a\}^*\{b\}^* \notin \mathcal{SHUF}$ (more precisely, since this language is infinite, either a subexpression that contains both a and b is subject to an iterated shuffle operation or two subexpressions that produce only a and b, respectively, are connected by a shuffle operation). Hence, $\mathcal{REG}$ is incomparable with all the classes on the left side of the diagram. The language of Example 8 is in $\mathcal{JFA}$, but not in $\mathcal{CFL}$. Furthermore, $\{a\}^*\{b\}^*$ is a context-free language, which implies that $\mathcal{CFL}$ is also incomparable with all the classes on the left side of the diagram. Finally, the incomparability of the classes $\mathcal{SHUF}$ and $\mathcal{GJFA}$ is established by Lemmas 31, 32 and 33. This concludes the proof. □

The inclusion diagram also motivates to study problems that can be expressed as follows. If $\mathcal{X}$ and $\mathcal{Y}$ are two language families with $\mathcal{X} \subsetneq \mathcal{Y}$ and if $\mathcal{Y}$ can be described by $\mathcal{Y}$-devices, what is the complexity (or even decidability) status of the problem, given some $\mathcal{Y}$-device Y, to determine if the language $L(Y)$ belongs to $\mathcal{X}$? In Section 7, we shall see that this type of problem is NP-hard for $\mathcal{X} = \mathcal{REG} \cap \mathcal{JFA}$ and $\mathcal{Y} = \mathcal{REG}$ (Theorem 46) or $\mathcal{Y} = \mathcal{JFA}$ (Theorem 45). For example, it is even undecidable whether or not a given context-free grammar generates a regular language; see [30].

6. Representations and Normal Forms

The representation theorem in [19] was proved on the level of α-SHUF expressions, so that we got a normal form theorem for these expressions. However, the connections to Parikh's theorem yields a different reasoning in the following proof.

Theorem 36 (Representation Theorem)**.** *Let $L \subseteq \Sigma^*$. Then, $L \in \mathcal{JFA}$ if and only if there exists a number $n \geq 1$ and finite sets $M_i \subseteq \Sigma^*$, $N_i \subseteq \Sigma^*$ for $1 \leq i \leq n$, so that the following representation is valid.*

$$L = \bigcup_{i=1}^{n} \mathrm{perm}(M_i) \sqcup\!\sqcup (\mathrm{perm}(N_i))^{\sqcup\!\sqcup,*}$$

Proof. Consider $L \subseteq \Sigma^*$ with $L \in \mathcal{JFA}$. By Theorem 24, $\psi_\Sigma(L)$ is semilinear. Hence,

$$\psi_\Sigma(L) = \bigcup_{i=1}^{n} S_i,$$

where the sets S_i are linear sets, which means that there are vectors $v^i, v^i_1, \ldots, v^i_{\ell_i}$ such that

$$S_i = \{x \in \mathbb{N}^{|\Sigma|} : \exists k_1, \ldots, k_{\ell_i} : x = v^i + \sum_{j=1}^{\ell_i} k_i v^i_j\}.$$

Let $M_i = \psi_\Sigma^{-1}(v^i)$ and $N_{i,j} = \psi_\Sigma^{-1}(v^i_j)$. Then,

$$S_i = \psi_\Sigma\left(M_i \sqcup\!\sqcup \coprod_{j=1}^{\ell_i} N_{i,j}^{\sqcup\!\sqcup,*}\right) = \psi_\Sigma\left(M_i \sqcup\!\sqcup \left(\bigcup_{j=1}^{\ell_i} N_{i,j}\right)^{\sqcup\!\sqcup,*}\right),$$

as ψ_Σ acts as a morphism. Let $N_i = \bigcup_{j=1}^{\ell_i} N_{i,j}$. Observe that by our definition, M_i, $N_{i,j}$ and hence N_i are all perm-closed. Hence, L can be represented as required.

As the required representation can be easily interpreted as some α-SHUF expression, any L that can be represented as in the theorem is in $\mathcal{JFA}$ according to Theorem 28. □

Recall that the star height was quite an important notion for the classical regular expressions; see [8, 9, 32]. The proof in [19] of the preceding theorem was based on an inductive argument involving the star height of the expressions. Omitting details, we only mention the following interesting consequences from the Representation Theorem that can be obtained by combining Theorem 36 with Theorem 28, Lemma 27 and Theorem 24.

Corollary 37. *$L \in \mathcal{JFA}$ if and only if there is a regular language R of star height at most one such that $L = \mathrm{perm}(R)$.*

From Proposition 30, we can immediately deduce:

Corollary 38. *A language is finite and perm-closed if and only if it can be described by some α-SHUF expression of height zero.*

Combining Corollary 38 with Theorem 24 and the well-known fact that finiteness of regular expressions can be decided, we immediately obtain the following consequence, as Theorem 36 guarantees that the height of $\mathcal{JFA}$ languages is zero or one.

Corollary 39. *It is decidable, given some JFA and some integer k, whether or not this JFA describes a language of height at most k.*

Notice that we have formulated, in this corollary, the shuffle analogue of the famous star height problem, which has been a major open problem for regular languages [33]. Recall that Eggan's Theorem [11] relates the star height of a regular language to its so-called cycle rank, which formalizes loop-nesting in NFAs. Again, the characterization theorems that we derived allow us to conclude that, in short, for any $L \in \mathcal{JFA}$ there exists some finite machine M of cycle rank at most one such that $L_{\mathrm{JFA}}(M) = L$.

The Representation Theorem can also be derived in yet another method, as derived in [13], where the connection to the definition of semilinear sets is also drawn, although with a different method and background. As mentioned earlier, regular expressions over free commutative monoids can be re-interpreted as regular expressions dealing with Parikh vectors.

7. Comparing $\mathcal{JFA}$ and $\mathcal{RELG}$

By the results of Meduna and Zemek, we know that $\mathcal{JFA}$ and $\mathcal{RELG}$ are two incomparable families of languages. Above, we already derived several characterizations of $\mathcal{JFA} \cap \mathcal{FIN} \subseteq \mathcal{RELG}$. Let us first explicitly state a characterization of $\mathcal{JFA} \cap \mathcal{RELG}$ that can be easily deduced from our previous results.

Proposition 40. *$L \in \mathcal{JFA} \cap \mathcal{RELG}$ if and only if $L \in \mathcal{RELG}$ and L is perm-closed.*

We mention this, as the class $\mathcal{JFA} \cap \mathcal{RELG}$ can be also characterized as follows according to Ehrenfeucht, Haussler and Rozenberg [12]. Namely, they describe this class of (what they call) commutative regular languages as finite unions of periodic languages.

According to Ehrenfeucht, Haussler and Rozenberg [12], a sequence of vectors

$$\rho = v_0, v_1, \ldots, v_{|\Sigma|} \in \mathbb{N}^{|\Sigma|}$$

is called a *base* (with respect to Σ) if and only if, for all $i, j \in \{1, \ldots, |\Sigma|\}$, $v_i(j) = 0$ if $i \neq j$. The *ρ-set*, written $\Theta(\rho)$, of a base ρ is defined by

$$\Theta(\rho) = \{v \in \mathbb{N}^{|\Sigma|} : \exists \ell_1, \ldots, \ell_{|\Sigma|} \in \mathbb{N} : v = v_0 + \sum_{i=1}^{|\Sigma|} \ell_i \cdot v_i\}.$$

ρ-sets are linear sets, and they are in one-to-one correspondence with their bases in the following sense:

Lemma 41 ([12])**.** *Let ρ, ρ' be bases with respect to Σ. Then, $\rho = \rho'$ if and only if $\Theta(\rho) = \Theta(\rho')$.*

Now, a language $L \subseteq \Sigma^*$ is called *periodic* if and only if it is perm-closed and there is a base ρ with respect to Σ such that $\psi_\Sigma(L) = \Theta(\rho)$.

Proposition 42. *Let $L \subseteq \Sigma^*$. L is periodic if and only if, for some word $w \in \Sigma^*$ and some function $n : \Sigma \to \mathbb{N}$,*

$$L = \mathrm{perm}(w) \sqcup \left(\bigcup_{a \in \Sigma} a^{n(a)} \right)^{\sqcup,*}.$$

Proof. Let $\Sigma = \{a_1, \ldots, a_{|\Sigma|}\}$. If L is periodic, then $L = \psi_\Sigma^{-1}(\Theta(\rho))$ for some base $\rho = v_0, v_1, \ldots, v_{|\Sigma|}$. Select $w \in \psi_\Sigma^{-1}(v_0)$ and set $n(a_i) = v_i(i)$. Then, $L = \left(\mathrm{perm}(w) \sqcup \left(\bigcup_{a \in \Sigma} a^{n(a)}\right)^{\sqcup,*}\right)$. Conversely, given $L = \mathrm{perm}(w) \sqcup \left(\bigcup_{a \in \Sigma} a^{n(a)}\right)^{\sqcup,*}$, one can see that $v_0 = \psi_\Sigma(w)$, $v_i(i) = n(a_i)$ and $v_i(j) = 0$ for $i \neq j$ defines a base ρ such that $\psi_\Sigma(L) = \Theta(\rho)$. □

Ehrenfeucht, Haussler and Rozenberg [12] have shown a characterization theorem that easily yields the following result.

Corollary 43. *A language L is regular and perm-closed if and only if L is the finite union of periodic languages.*

Proof. If L is regular and perm-closed, then L is the finite union of periodic languages according to [12, Theorem 6.5]. Conversely, as the finite union of perm-closed languages is perm-closed, we can conclude from [12, Theorem 6.5] that the finite union of periodic languages is regular and perm-closed. □

The last two results immediately yield Theorem 44.

Theorem 44. *Let $L \subseteq \Sigma^*$. Then, $L \in \mathcal{JFA} \cap \mathcal{RELG}$ if and only if there exists a number $n \geq 1$, words w_i and finite sets N_i for $1 \leq i \leq n$, where each N_i is given as $\bigcup_{a \in \Sigma_i} a^{n_i(a)}$ for some $\Sigma_i \subseteq \Sigma$ and some $n_i : \Sigma_i \to \mathbb{N}$, so that the following representation is valid.*

$$L = \bigcup_{i=1}^{n} \operatorname{perm}(w_i) \sqcup\!\sqcup (\operatorname{perm}(N_i))^{\sqcup\!\sqcup,*}$$

Let us finally mention that yet another characterization of $\mathcal{JFA} \cap \mathcal{RELG}$ was derived in [57, Theorem 3]. Moreover, a relaxed version of the notion of commutativity (of languages) allows a characterization of $\mathcal{RELG}$, as shown by Reutenauer [76]. We would also like to point the reader to [29], where not only learnability questions of this class of languages were discussed, but also two further normal form representations of $\mathcal{JFA} \cap \mathcal{RELG}$ were mentioned. Further algebraic properties of $\mathcal{JFA} \cap \mathcal{RELG}$ were presented by Mateescu [60]. A proper subclass of $\mathcal{JFA} \cap \mathcal{RELG}$ (star-free commutative languages) was characterized in [4] with the help of shuffle expressions in a certain normal form.

Next, we consider the problem to decide, for a given JFA or NFA, whether it accepts a language from $\mathcal{JFA} \cap \mathcal{RELG}$. This is equivalent to the task of deciding whether a given JFA accepts a regular language[4] or whether a given NFA accepts a commutative language. We shall show that both these problems are co-NP-hard, even if restricted to automata with binary alphabets. Note that for a language over a one-letter alphabet, regularity is equivalent to membership in $\mathcal{JFA}$, so the problem becomes trivial.

We will actually use nearly the same construction for both hardness results, which is based on [79], in which Stockmeyer and Meyer showed how to construct for each 3-SAT formula ϕ a regular expression E_ϕ over the unary alphabet $\Sigma = \{a\}$ such that

$$\phi \text{ is satisfiable if and only if } L(E_\phi) \neq \Sigma^* .$$

We will also make use of another property

$$\phi \text{ is satisfiable if and only if } \Sigma^* \setminus L(E_\phi) \text{ is infinite} .$$

Theorem 45. *The non-regularity problem for JFA is NP-hard, even for binary alphabets.*

[4] This question was explicitly asked in the Summary of Open Problems section (ii) of [66].

Proof. We present a reduction from 3-SAT. Let ϕ be a 3-SAT formula and let E_ϕ be the regular expression over $\{a\}$ with the properties described above. Let

$$L_\phi = (\{b\}^{\sqcup\!\sqcup,*} \sqcup\!\sqcup L(E_\phi)) \cup (\{a\} \sqcup\!\sqcup \{b\})^{\sqcup\!\sqcup,*}.$$

Note that a finite machine M_ϕ with $L_{\mathrm{JFA}}(M_\phi) = L_\phi$ can be easily obtained by transforming $(b^{\sqcup\!\sqcup,*} \sqcup\!\sqcup \widehat{E}_\phi) \cup (a \sqcup\!\sqcup b)^{\sqcup\!\sqcup,*}$ (where $\widehat{E}_\phi$ is obtained from E_ϕ by replacing all catenations and Kleene stars by shuffles and iterated shuffles, respectively) into a finite machine (treating shuffle and iterated shuffle as catenation and Kleene star, respectively) by a standard construction, e. g., the Thompson NFA construction.

Clearly, if ϕ is unsatisfiable, then

$$L_\phi = L((\{b\}^{\sqcup\!\sqcup,*} \sqcup\!\sqcup \{a\}^{\sqcup\!\sqcup,*}) \cup (\{a\} \sqcup\!\sqcup \{b\})^{\sqcup\!\sqcup,*}) = \{a, b\}^*;$$

and thus L_ϕ is regular. However, if ϕ is satisfiable, then $L' = \{a\}^* \setminus L(E_\phi)$ is an infinite regular set. Assume, for the sake of contradiction, that L_ϕ is regular. Then, also

$$\begin{aligned} & L_\phi \cap (\{b\}^{\sqcup\!\sqcup,*} \sqcup\!\sqcup L') \\ = {} & L((\{a\} \sqcup\!\sqcup \{b\})^{\sqcup\!\sqcup,*}) \cap (\{b\}^{\sqcup\!\sqcup,*} \sqcup\!\sqcup L') \\ = {} & \{w \in \{a, b\}^* : |w|_a = |w|_b \wedge a^{|w|_a} \notin L(E_\phi)\} \end{aligned}$$

would be regular. However, for every $k, k' \in \mathbb{N}$ with $k \neq k'$ and $a^k, a^{k'} \notin L(E_\phi)$, the words a^k and $a^{k'}$ are not Nerode-equivalent with respect to $L_\phi \cap (\{b\}^{\sqcup\!\sqcup,*} \sqcup\!\sqcup L')$. Thus, there are infinitely many equivalence classes of this Nerode relation, which is a contradiction. □

Notice that the regularity problem is decidable, as shown in a far more general context by Sakarovitch [77], referring to older papers of Ginsburg and Spanier [26, 27]. It would be also interesting to better understand the precise complexities for the regularity problems mentioned by Sakarovitch in the context of trace theory.

It would be of course interesting to determine the exact complexity status of this problem. Currently, we only know about the mentioned decidability result, which in itself is not completely natural, as similar problems like the universality for flow expressions is known to be undecidable [40]. Next, we deal with the problem of deciding whether a given NFA accepts a commutative language, i. e., a language from $\mathcal{JFA} \cap \mathcal{RES}$.

Theorem 46. *It is NP-hard to decide, for a given NFA M, whether $L_{\mathrm{FA}}(M)$ is noncommutative, even for binary languages.*

Proof. We use the fact that regular languages are closed under shuffle (see [22] or [25, p. 108]). Let ϕ be a 3-CNF formula and let E_ϕ be the regular expression over $\{a\}$ with the properties described above. We define the language

$$L'_\phi = (\{b\}^* \sqcup\!\sqcup L(E_\phi)) \cup \{a\}^*\{b\}.$$

Obviously, there is a finite automaton M'_ϕ, which can be easily constructed, that accepts L'_ϕ. If ϕ is unsatisfiable, then

$$L'_\phi = (\{b\}^* \sqcup\!\sqcup \{a\}^*) \cup \{a\}^*\{b\} = \{a, b\}^*$$

is commutative. If ϕ is satisfiable, then, for some $a^k \notin L(E_\phi)$, we have $a^k b \in L'_\phi$ and $ba^k \notin L'_\phi$; thus L'_ϕ is not commutative. □

Again, we are not aware of a matching upper bound. At least, decidability can be shown as follows. In [48], an explicit construction of a *biautomaton*[5] accepting $L_{\mathrm{FA}}(M)$ for a given DFA M is shown, though there is an exponential blowup in the number of states. Moreover, in [34] the authors present an algorithm for turning a biautomaton into the *canonical biautomaton* and show that commutativity of a language can be deduced from basic properties of the corresponding canonical biautomaton.

Notice that Theorem 56 from the Appendix allows us to deduce the following two corollaries. Indeed, because the number of states of the constructed automata is only a constant off from the number of states of the automaton M_G obtained in the proof of Theorem 56, this unary NFA could replace the regular expression E_ϕ used above. In fact, that proof (not delivered in the appendix) was from 3-Coloring, and we inherit the following property: G is 3-colorable if and only if $\{a\}^* \setminus L(M_G)$ is infinite. This property is important in the reasonings of the hardness proofs shown above.

Corollary 47. *There is no algorithm that solves the regularity problem for q-state JFAs on binary input alphabets in time $\mathcal{O}^*(2^{o(q^{1/3})})$, unless ETH fails.*

Corollary 48. *There is no algorithm that solves the commutativity problem for q-state NFAs on binary input alphabets in time $\mathcal{O}^*(2^{o(q^{1/3})})$, unless ETH fails.*

If we used the construction of Stockmeyer and Meyer (directly), we would only get bounds worse than $\mathcal{O}^*(2^{o(q^{1/4})})$ (for more details, we refer to the Appendix).

8. Complexity Issues

In this section, we investigate the complexity of parsing of JFA and GJFA languages as well as the complexity of the non-disjointness and non-universality problem. A summary of our results and existing results can be found in Table 1.

8.1. Parsing

For a fixed JFA M and a given word $w \in \Sigma^*$, we can decide whether $w \in L(M)$ in the following way[6]. Scan over w and construct the Parikh mapping $\psi_\Sigma(w)$ of w. Observe that $\psi_\Sigma(w)$ can be stored on the working tape of a nondeterministic logspace machine, as Σ is fixed. Simulate a computation of M on w by repeated nondeterministic choice of an outgoing transition, passing to the target state, and decrementing the component of $\psi_\Sigma(w)$ that corresponds to the label $x \in \Sigma$ of the chosen transition. If an accepting state is reached and all the components of $\psi_\Sigma(w)$ are 0, then $w \in L(M)$. In this procedure, we only have to store the current state and the Parikh mapping, which only requires logarithmic space. Thus, this shows $\mathcal{JFA} \subseteq \mathsf{NL} \subseteq \mathsf{P}$.[7]

[5] We do not introduce biautomata in this paper.

[6] For the sake of convenience, in the whole Section 8, we use $L(M)$ instead of $L_{\mathrm{JFA}}(M)$.

[7] We wish to point out that this also follows from results in [10], where containment in NL is shown for a superclass of $\mathcal{JFA}$.

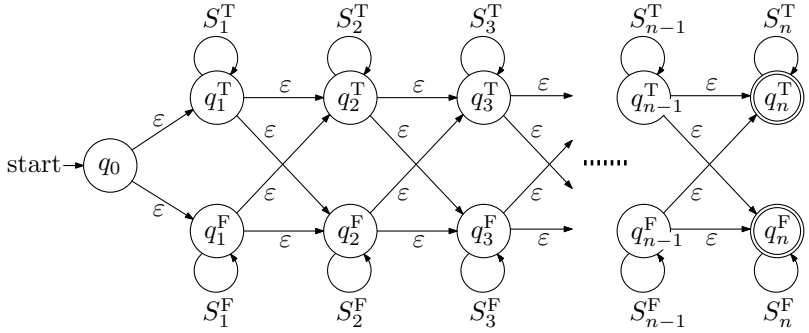

Figure 5: The JFA M representing a formula ϕ.

These considerations show that the *fixed* word problem can be solved in polynomial time. In contrast to the fixed word problem, the *universal* word problem is to decide, for a given automaton M and a given word w, whether $w \in L(M)$. The universal word problem for JFA is known to be solvable in polynomial time for fixed alphabets (see, e. g., [45]), but NP-complete in general. The hardness follows from our Theorem 49 or, e. g., from [62, Theorem 5.1], which gives a proof of the NP-hardness (concerning expressions using only union and shuffle) by a reduction from the problem of 3-dimensional matching. Alternatively, a very simple reduction from the Hamiltonian circle problem was given in [49]. The membership of this problem in NP is shown in our Theorem 51 or, e. g., in [49] again. See also [37] for generalizations towards commutative context-free grammars.

We shall improve the hardness result by giving a reduction from 3-SAT. This allows us to conclude a lower bound for the complexity of an algorithm solving the universal word problem for JFA, assuming the exponential time hypothesis (ETH), which is reproduced and further commented in the Appendix.

Theorem 49. *Unless ETH fails, there is no algorithm that, for a given JFA M with state set Q and a given word w, decides whether $w \in L(M)$ and runs in time $\mathcal{O}^*(2^{o(|Q|)})$.*

Proof. Fix a 3-CNF formula $\phi = \bigwedge_{j=1}^{m} C_j$, where each C_j is a disjunction of three literals over variables $x_1, x_2, \ldots, x_n$. Let $\Sigma = \{c_1, \ldots, c_m\}$. For each $i \in \{1, \ldots, n\}$, let $S_i^{\mathrm{T}} = \{c_j : x_i \in C_j\}$ and $S_i^{\mathrm{F}} = \{c_j : \neg x_i \in C_j\}$. We claim that the JFA $M = (Q, \Sigma, R, q_0, F)$ with

$$\begin{aligned} Q &= \{q_0\} \cup \{q_1^{\mathrm{T}}, \ldots, q_n^{\mathrm{T}}\} \cup \{q_1^{\mathrm{F}}, \ldots, q_n^{\mathrm{F}}\}, \\ F &= \{q_n^{\mathrm{T}}, q_n^{\mathrm{F}}\}, \end{aligned}$$

and transitions according to Figure 5 accepts the word $w = c_1 c_2 \ldots c_m$ if and only if ϕ is satisfiable.

- First, let $(\xi_1, \ldots, \xi_n) \in \{\mathrm{T}, \mathrm{F}\}^n$ be an assignment of $x_1, \ldots, x_n$ that satisfies ϕ. Consider the path in M that uses ε-transitions to visit the states

  $$q_0, q_1^{\xi_1}, q_2^{\xi_2}, \ldots, q_n^{\xi_n}$$

  and, moreover, in each state uses all possible loops (i. e., loops labeled by letters that still appear within the input). Because each clause contains

some x_i with $\xi_i = \mathrm{T}$ or $\neg x_i$ with $\xi_i = \mathrm{F}$, it follows that each letter c_j of the word w lies in $S_i^{\xi_i}$ for some i and thus is consumed by the loop on $q_i^{\xi_i}$.

- Second, assume that there is an accepting computation of M on w. For each $i \in \{1, \ldots, n\}$ the corresponding path in M must visit exactly one of the vertices $q_i^{\mathrm{T}}, q_i^{\mathrm{F}}$; let $\xi_i = \mathrm{T}$ or $\xi_i = \mathrm{F}$, respectively. For each $j \in \{1, \ldots, m\}$, the letter c_j is consumed from w and thus lies in $S_i^{\xi_i}$ for some i. It follows that each clause contains a satisfied literal.

As $|Q| = 2n + 1$ and the construction of M works in linear time, any algorithm that solves universal word problem for JFA in time $\mathcal{O}^*(2^{o(|Q|)})$ violates ETH. □

For *general* jumping finite automata the complexity of word problems increase considerably. In fact, there is a fixed general jumping finite automaton that accepts an NP-complete language, i. e., for GJFA even the fixed word problem is NP-complete.

Before proving that, let us give the following simple lemma, which is later used for reducing alphabet sizes of GJFAs.

Lemma 50. *Let M be a GJFA over $\Sigma = \{x_1, \ldots, x_k\}$. Then, there exists a homomorphism $h : \Sigma \to \{0,1\}^*$ and a GJFA M' over $\{0,1\}$ such that, for each $w \in \Sigma^*$, $w \in L(M)$ if and only if $h(w) \in L(M')$.*

Proof. For each $1 \leq i \leq k$, let $h(x_i) = 10^i1$. Let M' be obtained from M by replacing each rule $(q, u, r) \in R$ with $(q, h(u), r)$. Clearly, if $w \in L(M)$, then $h(w) \in L(M')$. On the other hand, the definition of $h(x_1), \ldots, h(x_k)$ implies that a computation of M on $h(w)$ can only consume factors of the form $h(x)$ corresponding to particular occurrences of x in w. □

Theorem 51. $\mathcal{GJFA} \subseteq$ NP.

Proof. For each GJFA $M = (Q, \Sigma, R, s, F)$ and $w \in L(M)$, there exists a computation of M that consists of at most $|Q|\,|w|$ steps (at most $|Q|$ transitions labeled by ε are taken between any two steps that shorten the current word). Such a witness for accepting w can be easily checked in polynomial time. □

Theorem 52. *There exists a GJFA M over a binary alphabet such that $L(M)$ is NP-complete.*

Proof. Let $M = (Q, \{0, 1, \overline{0}, \overline{1}, \star\}, R, q_{\mathrm{C}}, \{q_{\mathrm{C}}\})$ with $Q = \{q_{\mathrm{C}}, q_{\mathrm{D}}, q_0, q_1\}$ be defined according to Figure 6. For $u \in \{0,1\}^*$, $\overline{u}$ is obtained from u by replacing 0 with $\overline{0}$ and 1 with $\overline{1}$.

Let $u_1, u_2, \ldots, u_n, v \in \{0,1\}^*$ and $t_i = \star^{|u_i|+2} c \overline{u}_i \overline{c}$, $1 \leq i \leq n$. First, we prove the equivalence of the following two statements.

1. On input $w = \star^{|v|} v t_1 t_2 \ldots t_n$, M can reach state q_{C} from state q_{C} with remaining input w' and without visiting q_{C} in between.
2. $w' = \star^{|v'|} v' t_1 t_2 \ldots t_{i-1} t_{i+1} \ldots t_n$ with $v = u_i v'$, for some i, $1 \leq i \leq n$.

Assume that M starts in q_{C} with input $w = \star^{|v|} v t_1 t_2 \ldots t_n$. In the first step, while changing into state q_{D}, M consumes a factor $\star c$ from w. After this step, the remaining input is $\star^{|v|} v t_1 t_2 \ldots t_{i-1} \star^{|u_i|+1} \overline{u}_i \overline{c} t_{i+1} \ldots t_n$, for some i, $1 \leq i \leq n$. Now, by using states q_0 and q_1, a sequence of factors $\star\overline{y}_1$, $\star y_1$, $\star\overline{y}_2$, $\star y_2$, …, $\star\overline{y}_m$, $\star y_m$ is consumed, where $y_i \in \{0,1\}$ for $1 \leq i \leq m$. All these factors only

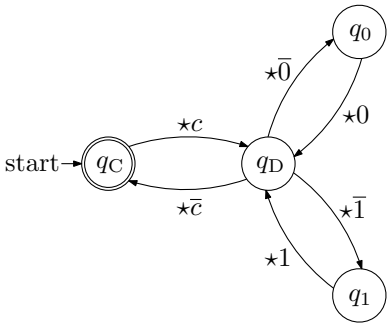

Figure 6: A GJFA M that can solve the problem EBC_2.

occur in the middle of the factor $\star^{|u_i|}\overline{u}_i$. Furthermore, M can only change into state q_C again if there exists a factor $\star\overline{c}$, which is only the case when the whole factor $\star^{|u_i|}\overline{u}_i$ is consumed. This implies the second statement.

If the second statement holds, then the transitions described above (each y_i being chosen such that $y_1 y_2 \dots y_m = u_i$) will lead M on input $\star^{|v|} v t_1 t_2 \dots t_n$ from state q_C into state q_C without visiting q_C in between.

Next, consider the following computational problem, which was shown to be NP-complete in [46]:

Binary Exact Block Cover (EBC_2)
Instance: Words $u_1, u_2, \dots, u_k$, and v over $\{0,1\}$.
Question: Does there exist a permutation $\pi : \{1,2,\dots,k\} \to \{1,2,\dots,k\}$ such that $v = u_{\pi(1)} u_{\pi(2)} \dots u_{\pi(k)}$?

Let $(u_1, u_2, \dots, u_k, v)$ be an instance of EBC_2. If we apply the claim from above inductively, it follows immediately that $\star^{|v|} v t_1 t_2 \dots t_n \in L(M)$ if and only if there exists a permutation π with $v = u_{\pi(1)} u_{\pi(2)} \dots u_{\pi(k)}$. The permutation π corresponds to the order in which the factors t_i are consumed by M.

Finally, Lemma 50 says that M can be easily turned into a binary GJFA M', while the corresponding homomorphism h serves as a polynomial-time reduction from $L(M)$ to $L(M')$. □

Obviously, Theorem 52 implies that the universal word problem for GJFA is NP-complete as well, which has been shown by a separate reduction in the conference version of this paper [19]. We wish to point out, however, that the hardness result for the universal word problem given in [19] is stronger in the sense that it also holds under the restriction to GJFA that accept finite languages.

Theorem 53 ([19])**.** *The universal word problem is NP-complete for GJFAs accepting finite languages over binary alphabets.*

A benefit of Theorem 52 is that the employed GJFA is rather simple; thus, we obtain a simple proof. However, by choosing a more complicated GJFA, we can obtain a reduction from 3-SAT to the fixed word problem for GJFA, which allows us to conclude a stronger lower bound that relies on the exponential time hypothesis.

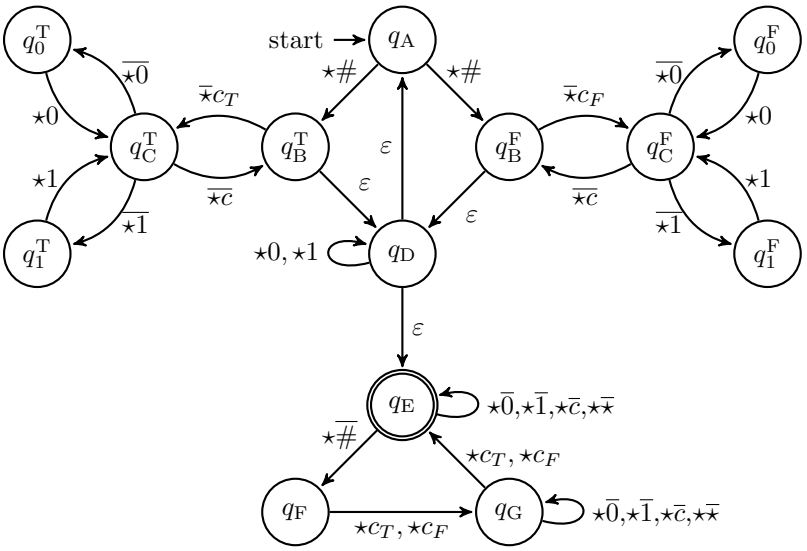

Figure 7: A GJFA M that can solve 3-SAT.

Theorem 54. *There exists a GJFA M over a binary alphabet such that, unless ETH fails, there is no algorithm that, for a given word w, decides whether $w \in L(M)$ and runs in time $\mathcal{O}^*\left(2^{o\left(\frac{|w|}{\log|w|}\right)}\right)$.*

Proof. Consider the GJFA $M = (Q, \Sigma, R, q_{\mathrm{A}}, \{q_{\mathrm{E}}\})$, where

$$\begin{aligned} \Sigma &= \{0, 1, \overline{0}, \overline{1}, c_T, c_F, \overline{c}, \star, \#, \overline{\star}, \overline{\#}\}, \\ Q &= \{q_{\mathrm{A}}, q_{\mathrm{B}}^{\mathrm{T}}, q_{\mathrm{B}}^{\mathrm{F}}, q_{\mathrm{C}}^{\mathrm{T}}, q_{\mathrm{C}}^{\mathrm{F}}, q_0^{\mathrm{T}}, q_0^{\mathrm{F}}, q_1^{\mathrm{T}}, q_1^{\mathrm{F}}, q_{\mathrm{D}}, q_{\mathrm{E}}, q_{\mathrm{F}}, q_{\mathrm{G}}\}, \end{aligned}$$

as defined in Figure 7. Fix a 3-CNF formula $\phi = \bigwedge_{j=1}^{m} C_j$, where

$$C_j = \lambda_{j,1} \vee \lambda_{j,2} \vee \lambda_{j,3}$$

and $\lambda_{j,1}, \lambda_{j,2}, \lambda_{j,3}$ are literals over variables $x_1, x_2, \ldots, x_n$. Suppose that for each $i \in \{1, \ldots, n\}$, the variable x_i occurs p_i times in ϕ. Note that $\sum_{i=1}^{n} p_i = 3m$. Let $S = \lceil \log_2(n) \rceil$ and fix distinct codes $u_1, \ldots, u_n \in \{0,1\}^S$ for the variables. Let

$$\begin{aligned} w_{\mathrm{aux}} &= \star^{n+3mS} \# u_1^{p_1} \# u_2^{p_2} \cdots \# u_n^{p_n}, \\ w_\phi &= \star^{m+m\cdot 6(S+2)} \overline{\#} t_1 \overline{\#} t_2 \cdots \overline{\#} t_m, \end{aligned}$$

where

$$\begin{aligned} t_{j,r} &= \begin{cases} c_{\mathrm{T}} \overline{u}_i \overline{c} & \text{if } \lambda_{j,r} = x_i, \\ c_{\mathrm{F}} \overline{u}_i \overline{c} & \text{if } \lambda_{j,r} = \neg x_i, \end{cases} \\ t_j &= \overline{\star}^{S+2} t_{j,1} \overline{\star}^{S+2} t_{j,2} \overline{\star}^{S+2} t_{j,3} \end{aligned}$$

for each $j \in \{1, \ldots, m\}$ and $r \in \{1, 2, 3\}$. Finally, let $w = w_{\mathrm{aux}} w_\phi$ and let us claim that M accepts the word w if and only if ϕ is satisfiable.

It is clear that the machine works in two phases:

1. In the *first phase*, which ends once the transition from q_{D} to q_{E} is taken, some parts of w_ϕ and (as explained below) the entire word w_{aux} are consumed.
2. In the *second phase*, only the transitions between the states q_{E}, q_{F}, and q_{G} can be used. Observe:
   - Each of the transition labels contains c_{T}, c_{F}, or a letter with bar, which implies that only factors from w_ϕ are consumed in the second phase; therefore, in order to fully consume the input, w_{aux} must be consumed in the first phase.
   - Each of the transition labels starts with $\star$, which implies that the remainder of $\overline{\#}t_1\overline{\#}t_2\cdots\overline{\#}t_m$ is consumed left-to-right only.
   - The occurrences of $\overline{\#}$, c_{T}, and c_{F} in transition labels imply that the second phase is successful only if c_{T} and c_{F} together occur exactly twice between successive occurrences of $\overline{\#}$.

   It follows that before the second phase starts, exactly one of the three occurrences of c_{T} and c_{F} must be consumed from each of the segments $t_1, t_2, \ldots, t_m$. This corresponds to at least one literal of each clause being satisfied.

It remains to check that:

- A run of the first phase must follow some fixed assignment of variables while consuming parts of $t_1, t_2, \ldots, t_m$ (i. e., only factors $t_{j,r}$ standing for satisfied literals are consumed).
- Vice versa, for each assignment there is a run of the first phase that only deletes factors $t_{j,r}$ standing for satisfied literals.

Suppose that M is in the state q_{A}. It must use a transition labeled with $\star\#$ leading to $q_{\mathrm{B}}^{\mathrm{X}}$ for $\mathrm{X} \in \{\mathrm{T}, \mathrm{F}\}$, which implies that the remainder of w_{aux} is then of the form $\star\cdots\star\, u_i^{p_i}\#u_{i+1}^{p_{i+1}}\cdots\#u_n^{p_n}$. Then, before passing to q_{D} it can repeat the following process up to p_i times:

- *Open* a literal in any clause by consuming $\overline{\star}c_{\mathrm{X}}$ from $\overline{\star}^{S+2}t_{j,r}$. If $\mathrm{X} = \mathrm{T}$ (or $\mathrm{X} = \mathrm{F}$), only a positive (negative, respectively) literal can be open.
- Use the transitions between q_0^{X}, q_1^{X} and $q_{\mathrm{C}}^{\mathrm{X}}$ to consume $\overline{\star}^S\overline{u}_i$ from $\overline{\star}^{S+2}t_{j,r}$ together with consuming $\star^S u_i$ from w_{aux}. This is necessary because passing back to $q_{\mathrm{B}}^{\mathrm{X}}$ is not possible until only $\overline{\star c}$ remains from $\overline{\star}^{S+2}t_{j,r}$.
- *Close* the literal, i. e., finish consuming $\overline{\star}^{S+2}t_{j,r}$ with passing back to $q_{\mathrm{B}}^{\mathrm{X}}$.

Then, M passes to q_{D} and must use the loops on q_{D} to consume a possible reminder of $u_i^{p_i}$ from w_{aux}. After that, if w_{aux} is not fully consumed, the first phase must continue, i. e., M passes back to q_{A}.

Together, for each $1 \leq i \leq n$ the automaton chooses $\mathrm{X} \in \{\mathrm{T}, \mathrm{F}\}$ and then deletes from w_ϕ an arbitrary number of factors that stand for occurrences of the exact literal x_i or $\neg x_i$, respectively.

Finally, with Lemma 50, we can convert M to a binary GJFA M', such that $h(w) \in L(M')$ if and only if ϕ is satisfiable. Note that $|h(w)| = \mathcal{O}(m \log n)$. We assume that there is an algorithm deciding, for a given binary word v, whether

$v \in L(M')$ in time $\mathcal{O}^*\left(2^{o\left(\frac{|v|}{\log|v|}\right)}\right)$. Then we can obtain an algorithm for 3-SAT as follows. We first transform the 3-CNF formula ϕ with n variables and m clauses into $h(w)$ as described above, which can be done in linear time, then we check whether $h(w) \in L(M')$ in time $\mathcal{O}^*\left(2^{o\left(\frac{|h(w)|}{\log|h(w)|}\right)}\right) \subseteq \mathcal{O}^*\left(2^{o\left(\frac{m \log n}{\log n}\right)}\right) = \mathcal{O}^*\left(2^{o(m)}\right)$. This is a contradiction to ETH. □

A special feature of the two particular GJFAs used in the proofs of Theorems 52 and 54 (see Figures 6 and 7) is that the length of transition labels is at most 2 (note that for JFAs, i. e., machines with labels of length at most 1, the fixed word problem lies in P). However, Lemma 50, converting the GJFAs to binary ones, increases the lengths of labels. It is open whether the fixed word problem remains NP-hard for GJFAs with binary alphabets and with words of length at most 2 in the transitions.

The hardness results presented here point out that the difference between finite machines and general finite machines is crucial if we interpret them as jumping finite automata. In contrast to this, the universal word problem for classical finite automata on the one hand and classical general finite automata on the other is very similar in terms of complexity, i. e., in both cases it can be solved in polynomial time.

The fact that descriptional mechanisms could yield NP-hard universal word problems if both shuffle and concatenation operations are somehow involved was already known. For instance, in [71] it is remarked at the end that expressions formed like ordinary regular expressions, but with shuffle as an additional operator, lead to a type of expressions with an NP-complete universal word problem. However, unlike in the case of GJFA, the class of languages that can be described is just $\mathcal{RES}$ and hence at least the fixed word problem lies in NL for this type of expressions.

Let us comment on one more aspect of parsing JFA. Although this mechanism was presented as a device to accept words (elements of the free monoid), the very nature of the acceptance mode brings along the idea to present input words as tuples of integers. This makes no difference as long as the integers are encoded in unary, but it does make a difference if they are encoded in binary. This is the standard encoding for the commutative grammars / semilinear sets as studied by Huynh [36, 37] and also explains why his complexity results seemingly deviate from ours. More precisely, he has shown that the universal word problem for JFA (when they are considered as processing tuples of numbers encoded in binary) is indeed NP-complete.

8.2. Non-Disjointness and Non-Universality

The *non-disjointness problem* is the task to decide, for given automata M_1 and M_2, whether there exists a word w that is accepted by both M_1 and M_2, i. e., whether it holds that $L(M_1) \cap L(M_2) \neq \emptyset$. In the case of JFA, we encounter a similar situation as for the universal word problem, i. e., it can be decided in polynomial time for fixed alphabets, while it becomes NP-complete in general [49].

The NP-hardness of non-disjointness also follows easily from the proof of Theorem 49. Moreover, the complexity lower bound depending on ETH applies to this problem as well:

	JFA	**JFA** $\lvert\Sigma\rvert=k$	**GJFA**	**GJFA** $\lvert\Sigma\rvert=k$
fixed word problem	P [45]	P	NPC	NPC if $k\geq 2$ ♦
universal word problem	NPC [62]	P [45]	NPC	NPC if $k\geq 2$ ♦
non-disjointness	NPC [49]	P [49]	UND	UND if $k\geq 8$ [80]
non-universality	NPH [67]	NPC if $k\geq 1$ [49]	UND [81]	NPH if $k\geq 1$ [49]

Table 1: Lower and upper bounds on complexity of basic problems. Legend: NPC — NP-complete; NPH — NP-hard, membership in NP unknown; UND — undecidable; ♦ — present results

Theorem 55. *There is no algorithm deciding, for given JFAs M_1, M_2 with state sets Q_1, Q_2, whether $L(M_1) \cap L(M_2) \neq \emptyset$ in time $\mathcal{O}^*\big(2^{o(|Q_1|+|Q_2|)}\big)$, unless ETH fails.*

Proof. The construction from the proof of Theorem 49 produces, for given formula ϕ with n variables and m clauses, a JFA M_1 with $\mathcal{O}(n)$ states and a word w of length m such that ϕ is satisfiable if and only if $w \in L(M_1)$. We can trivially construct a JFA M_2 with $\mathcal{O}(m)$ states such that $L(M_2) = \mathrm{perm}(w)$. Then $w \in L(M_1)$ if and only if $L(M_1) \cap L(M_2) \neq \emptyset$.

Thus, any algorithm answering $L(M_1) \cap L(M_2) \neq \emptyset$ in time $\mathcal{O}^*\big(2^{o(|Q_1|+|Q_2|)}\big)$ can solve 3-SAT in time $\mathcal{O}^*(2^{o(n+m)})$, which violates ETH. □

Another basic decision problem is the *non-universality problem*, where the task is to decide, for a given automaton M, whether $L(M) \neq \Sigma^*$. Results of [67] and the fact that, on unary alphabets, classical nondeterministic machines coincide with JFAs, imply that the non-universality problem for JFA is NP-hard even if restricted to JFA with unary alphabets. On the other hand, [49] shows (in terms of a more general model) that non-universality lies in NP for any fixed alphabet size. For the unrestricted variant of non-universality, which is trivially NP-hard as well, no close upper bound of the complexity is known [49].

9. Discussions and Prospects

We have related the concept of jumping finite automata to the, actually quite well-studied, area of expressions involving shuffle operators. This immediately opens up further questions, and it also shows some limitations for this type of research programme.

- Is there a characterization of the class of languages accepted by general jumping finite automata in terms of expressions?[8] We are currently work-

[8] We claimed to have found such a characterization at the German Formal Language community meeting in 2014, but this claim turned out to be flawed.

ing on this question and other ones related to GJFAs.

- The original motivation for introducing variants of expressions involving shuffle operators was to model parallel features from programming languages; see, e. g., [6, 64, 78]. It is well-known that adding all according features immediately lead to expressions that are computationally complete, i. e., they characterize the recursively enumerable languages [2]. Notably, expressions with limited nesting of iterated concatenation and iterated shuffle operators (as provided by our main normal form results for α-SHUF expressions) have a descriptive power limited by Petri nets (without inhibitor arcs), so that in particular the non-emptiness problem for such limited expressions is decidable (in contrast to the general situation), confer [1, 17, 51, 63]. Yet, decidability questions for Petri nets are quite hard, so that in any case the study of restricted versions of shuffle expressions or related devices is of considerable practical interest.

- The inductive definition of α-SHUF expressions starts with single letters (plus symbols for the empty set and the empty word). This is contrasting the definition of SHUF expressions, which starts with any finite language as a basis. As it is well-known, for classical regular expressions this difference vanishes. Hierarchies as the one explained in [23] should inspire similar research for α-SHUF expressions as introduced in this paper.

- As there is a number of variations and restrictions of the shuffle operation itself [47, 52, 53, 61], it would be also interesting to study expressions that contain some of these. We plan to deal with this topic in the near future.

- The whole area seems to be related to *membrane systems*, also known as *P systems*. The reason is that membrane computing often reduces to *multiset computing*, which is just another name for dealing with subsets of $\mathbb{N}^{|\Sigma|}$. These connections are explained by Kudlek and Mitrana in [54].

- We have somehow initiated the study of complexity aspects of JFA and related models under ETH. Many other automata problems can be investigated in this paradigm (as also indicated in the Appendix), and more importantly from an algorithmic point of view, it would be interesting to know of procedures that match the proven lower bounds.

Summarizing, the study of expressions involving the shuffle operation, as well as of variants of jumping automata, still offers a lot of interesting questions, as it is also indicated in the recent survey of Restivo [75].

Acknowledgements

Meenakshi Paramasivan — Supported by the DAAD PhD Funding Programme - Research Grants for Doctoral Candidates and Young Academics and Scientists - Programme ID: 57076385.
Vojtěch Vorel — Supported by the Czech Science Foundation grant GA14-10799S and the GAUK grant No. 52215.

References

[1] T. Araki, T. Kagimasa, and N. Tokura. Relations of flow languages to Petri net languages. *Theoretical Computer Science*, 15:51–75, 1981.

[2] T. Araki and N. Tokura. Flow languages equal recursively enumerable languages. *Acta Informatica*, 15:209–217, 1981.

[3] J. Beauquier, M. Blattner, and M. Latteux. On commutative context-free languages. *Journal of Computer and System Sciences*, 35(3):311–320, 1987.

[4] J. Berstel, L. Boasson, O. Carton, J.-E. Pin, and A. Restivo. The expressive power of the shuffle product. *Information and Computation*, 208(11):1258–1272, 2010.

[5] M. Cadilhac, A. Finkel, and P. McKenzie. Bounded Parikh automata. *International Journal of Foundations of Computer Science*, 23(8):1691–1710, 2012.

[6] R. H. Campbell and A. N. Habermann. The specification of process synchronization by path expressions. In E. Gelenbe and C. Kaiser, editors, *Operating Systems OS*, volume 16 of *LNCS*, pages 89–102. Springer, 1974.

[7] C. Choffrut, A. Malcher, C. Mereghetti, and B. Palano. First-order logics: some characterizations and closure properties. *Acta Informatica*, 49(4):225–248, 2012.

[8] R. S. Cohen. Star height of certain families of regular events. *Journal of Computer and System Sciences*, 4:281–297, 1970.

[9] R. S. Cohen and J. A. Brzozowski. General properties of star height of regular events. *Journal of Computer and System Sciences*, 4:260–280, 1970.

[10] S. Crespi-Reghizzi and P. San Pietro. Commutative languages and their composition by consensual methods. In Z. Ésik and Z. Fülöp, editors, *Proceedings 14th International Conference on Automata and Formal Languages, AFL*, volume 151 of *EPTCS*, pages 216–230, 2014.

[11] L. C. Eggan. Transition graphs and the star-height of regular events. *The Michigan Mathematical Journal*, 10(4):385–397, December 1963.

[12] A. Ehrenfeucht, D. Haussler, and G. Rozenberg. On regularity of context-free languages. *Theoretical Computer Science*, 27:311–332, 1983.

[13] S. Eilenberg and M. P. Schützenberger. Rational sets in commutative monoids. *Journal of Algebra*, 13:173–191, 1969.

[14] Z. Ésik and W. Kuich. *Modern Automata Theory.* 2012.

[15] J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundamenta Informaticae*, 31:13–26, 1997.

[16] J. Esparza, P. Ganty, S. Kiefer, and M. Luttenberger. Parikh's theorem: A simple and direct automaton construction. *Information Processing Letters*, 111(12):614–619, 2011.

[17] J. Esparza and M. Nielsen. Decidability issues for Petri nets – a survey. *EATCS Bulletin*, 52:244–262, 1994.

[18] H. Fernau and A. Krebs. Problems on finite automata and the exponential time hypothesis. In Y.-S. Han and K. Salomaa, editors, *Implementation and Application of Automata - 21st International Conference, CIAA 2016*, volume 9705 of *LNCS*, pages 89–100. Springer, 2016.

[19] H. Fernau, M. Paramasivan, and M. L. Schmid. Jumping finite automata: Characterizations and complexity. In F. Drewes, editor, *Implementation and Application of Automata - 20th International Conference, CIAA*, volume 9223 of *LNCS*, pages 89–101. Springer, 2015.

[20] H. Fernau and J. M. Sempere. Permutations and control sets for learning non-regular language families. In A. L. Oliveira, editor, *Grammatical Inference: Algorithms and Applications, 5th International Colloquium ICGI 2000*, volume 1891 of *LNCS/LNAI*, pages 75–88. Springer, 2000.

[21] H. Fernau and R. Stiebe. Sequential grammars and automata with valences. *Theoretical Computer Science*, 276:377–405, 2002.

[22] N. E. Flick and M. Kudlek. A hierarchy of languages with catenation and shuffle. In L. Popova-Zeugmann, editor, *Proceedings of the 21th International Workshop on Concurrency, Specification and Programming, CS&P*, volume 928 of *CEUR Workshop Proceedings*, pages 103–114. CEUR-WS.org, 2012.

[23] N. E. Flick and M. Kudlek. On a hierarchy of languages with catenation and shuffle. In H.-C. Yen and O. H. Ibarra, editors, *Developments in Language Theory, DLT*, volume 7410 of *LNCS*, pages 452–458. Springer, 2012.

[24] M. R. Garey and D. S. Johnson. *Computers and Intractability.* New York: Freeman, 1979.

[25] S. Ginsburg. *The Mathematical Theory of Context-Free Languages.* McGraw-Hill, 1966.

[26] S. Ginsburg and E. H. Spanier. Bounded ALGOL-like languages. *Transactions of the American Mathematical Society*, 113:333–368, 1964.

[27] S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.

[28] S. Ginsburg and E. H. Spanier. AFL with the semilinear property. *Journal of Computer and System Sciences*, 5:365–396, 1971.

[29] A. C. Gómez and G. I. Álvarez. Learning commutative regular languages. In A. Clark, F. Coste, and L. Miclet, editors, *Grammatical Inference: Algorithms and Applications, 9th International Colloquium, ICGI*, volume 5278 of *LNCS*, pages 71–83. Springer, 2008.

[30] S. Greibach. A note on undecidable properties of formal languages. *Mathematical Systems Theory*, 2:1–6, 1968.

[31] S. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7:311–324, 1978.

[32] K. Hashiguchi. Regular languages of star height one. *Information and Control (now Information and Computation)*, 53(3):199–210, 1982.

[33] K. Hashiguchi. Algorithms for determining relative star height and star height. *Information and Computation*, 78(2):124–169, 1988.

[34] M. Holzer and S. Jakobi. Minimization and characterizations for biautomata. *Fundamenta Informaticae*, 136(1–2):113–137, 2015.

[35] M. Höpner and M. Opp. About three equations classes of languages built up by shuffle operations. In A. W. Mazurkiewicz, editor, *Mathematical Foundations of Computer Science 1976, 5th Symposium, MFCS*, volume 45 of *LNCS*, pages 337–344. Springer, 1976.

[36] D. T. Huynh. The complexity of semilinear sets. *Elektronische Informationsverarbeitung und Kybernetik (J. Inf. Process. Cybern. EIK)*, 18:291–338, 1982.

[37] D. T. Huynh. Commutative grammars: The complexity of uniform word problems. *Information and Control (now Information and Computation)*, 57(1):21–39, 1983.

[38] O. H. Ibarra and S. Seki. Characterizations of bounded semilinear languages by one-way and two-way deterministic machines. *International Journal of Foundations of Computer Science*, 23(6):1291–1306, 2012.

[39] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[40] K. Iwama. The universe problem for unrestricted flow languages. *Acta Informatica*, 19(1):85–96, 1983.

[41] M. Jantzen. Eigenschaften von Petrinetzsprachen. Technical Report IFI-HH-B-64, Fachbereich Informatik, Universität Hamburg, Germany, 1979.

[42] M. Jantzen. The power of synchronizing operations on strings. *Theoretical Computer Science*, 14:127–154, 1981.

[43] M. Jantzen. Extending regular expressions with iterated shuffle. *Theoretical Computer Science*, 38:223–247, 1985.

[44] J. Jedrzejowicz. Infinite hierarchy of shuffle expressions over a finite alphabet. *Information Processing Letters*, 36(1):13–17, 1990.

[45] J. Jedrzejowicz and A. Szepietowski. Shuffle languages are in P. *Theoretical Computer Science*, 250(1–2):31–53, 2001.

[46] H. Jiang, B. Su, M. Xiao, Y. Xu, F. Zhong, and B. Zhu. On the exact block cover problem. In Q. Gu, P. Hell, and B. Yang, editors, *Algorithmic Aspects in Information and Management - 10th International Conference, AAIM*, volume 8546 of *LNCS*, pages 13–22. Springer, 2014.

[47] L. Kari and P. Sosík. Aspects of shuffle and deletion on trajectories. *Theoretical Computer Science*, 332(1–3):47–61, 2005.

[48] O. Klíma and L. Polák. On biautomata. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 46:573–592, 2012.

[49] E. Kopczyński. Complexity of problems of commutative grammars. *Logical Methods in Computer Science*, 11(1:9), 2015.

[50] J. Kortelainen. Remarks about commutative context-free languages. *Journal of Computer and System Sciences*, 56(1):125–129, 1998.

[51] S. R. Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In H. R. Lewis, B. B. Simons, W. A. Burkhard, and L. H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, STOC*, pages 267–281. ACM, 1982.

[52] M. Kudlek. On mix operation. In Gh. Păun and A. Salomaa, editors, *New Trends in Formal Languages*, volume 1218 of *LNCS*, pages 430–439. Berlin: Springer, 1997.

[53] M. Kudlek and A. Mateescu. On distributed catenation. *Theoretical Computer Science*, 180(1–2):341–352, 1997.

[54] M. Kudlek and V. Mitrana. Considerations on a multiset model for membrane computing. In Gh. Păun, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *Membrane Computing, International Workshop, WMC-CdeA 2002*, volume 2597 of *LNCS*, pages 352–359. Springer, 2003.

[55] Z. Křivka and A. Meduna. Jumping grammars. *International Journal of Foundations of Computer Science*, 26(6):709–732, 2015.

[56] M. Latteux. Cônes rationnels commutatifs. *Journal of Computer and System Sciences*, 18(3):307–333, 1979.

[57] M. Latteux and G. Rozenberg. Commutative one-counter languages are regular. *Journal of Computer and System Sciences*, 1:54–57, 1984.

[58] G. J. Lavado, G. Pighizzini, and S. Seki. Converting nondeterministic automata and context-free grammars into Parikh equivalent one-way and two-way deterministic automata. *Information and Computation*, 228:1–15, 2013.

[59] D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *EATCS Bulletin*, 105:41–72, 2011.

[60] A. Mateescu. Scattered deletion and commutativity. *Theoretical Computer Science*, 125(2):361–371, 1994.

[61] A. Mateescu, G. Rozenberg, and A. Salomaa. Shuffle on trajectories: Syntactic constraints. *Theoretical Computer Science*, 197(1–2):1–56, 1998.

[62] A. J. Mayer and L. J. Stockmeyer. The complexity of word problems–this time with interleaving. *Information and Computation*, 115(2):293–311, 1994.

[63] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing*, 13(3):441–460, 1984.

[64] A. W. Mazurkiewicz. Parallel recursive program schemes. In J. Becvár, editor, *Mathematical Foundations of Computer Science 1975, MFCS*, volume 32 of *LNCS*, pages 75–87. Springer, 1975.

[65] A. Meduna and P. Zemek. Jumping finite automata. *International Journal of Foundations of Computer Science*, 23(7):1555–1578, 2012.

[66] A. Meduna and P. Zemek. Chapter 17: Jumping finite automata. In *Regulated Grammars and Automata*, pages 567–585. Springer, New York, 2014.

[67] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *13th Annual Symposium on Switching and Automata Theory, SWAT / FOCS*, pages 125–129. IEEE Computer Society, 1972.

[68] V. Mitrana and R. Stiebe. Extended finite automata over groups. *Discrete Applied Mathematics*, 108(3):287–300, 2001.

[69] B. Nagy. Languages generated by context-free grammars extended by type $AB \rightarrow BA$ rules. *Journal of Automata, Languages and Combinatorics*, 14(2):175–186, 2009.

[70] B. Nagy. On a hierarchy of permutation languages. In M. Ito, Y. Kobayashi, and K. Shoji, editors, *Automata, Formal Languages and Algebraic Systems (AFLAS 2008)*, pages 163–178. World Scientific, 2010.

[71] W. F. Ogden, W. E. Riddle, and W. C. Rounds. Complexity of expressions allowing concurrency. In A. V. Aho, S. N. Zilles, and T. G. Szymanski, editors, *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, POPL*, pages 185–194. ACM Press, 1978.

[72] F. Otto. Restarting automata. In Z. Ésik, C. Martín-Vide, and V. Mitrana, editors, *Recent Advances in Formal Languages and Applications*, volume 25 of *Studies in Computational Intelligence*, pages 269–303. Springer, 2006.

[73] R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

[74] K. Reinhardt. *Counting as Method, Model and Task in Theoretical Computer Science*. Habilitationsschrift, Universität Tübingen, 2005.

[75] A. Restivo. The shuffle product: New research directions. In A. Horia Dediu, E. Formenti, C. Martín-Vide, and B. Truthe, editors, *Language and Automata Theory and Applications - 9th International Conference, LATA*, volume 8977 of *LNCS*, pages 70–81. Springer, 2015.

[76] C. Reutenauer. A new characterization of the regular languages. In S. Even and O. Kariv, editors, *Automata, Languages and Programming, 8th Colloquium, ICALP*, volume 115 of *LNCS*, pages 177–183. Springer, 1981.

[77] J. Sakarovitch. The 'last' decision problem for rational trace languages. In I. Simon, editor, *LATIN '92, 1st Latin American Symposium on Theoretical Informatics*, volume 583 of *LNCS*, pages 460–473. Springer, 1992.

[78] A. C. Shaw. Software descriptions with flow expressions. *IEEE Transactions on Software Engineering*, 4(3):242–254, 1978.

[79] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In A. V. Aho, A. Borodin, R. L. Constable, R. W. Floyd, M. A. Harrison, R. M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, STOC*, pages 1–9. ACM, 1973.

[80] V. Vorel. On basic properties of jumping finite automata. *CoRR*, abs/1511.08396, 2015.

[81] V. Vorel. Two results on discontinuous input processing. In C. Câmpeanu, F. Manea, and J. O. Shallit, editors, *Descriptional Complexity of Formal Systems, 18th International Conference, DCFS*, volume 9777 of *LNCS*, pages 205–216. Springer, 2016.

[82] G. Zetzsche. The emptiness problem for valence automata or: Another decidable extension of Petri nets. In M. Bojanczyk, S. Lasota, and I. Potapov, editors, *Reachability Problems - 9th International Workshop, RP*, volume 9328 of *LNCS*, pages 166–178. Springer, 2015.

10. Appendix: The Exponential Time Hypothesis

The Exponential Time Hypothesis (ETH) was formulated by Impagliazzo, Paturi and Zane in [39]:

> There is a positive real s such that 3-SAT with n variables and m clauses cannot be solved in time $2^{sn}(n+m)^{\mathcal{O}(1)}$.

ETH considerably strengthens the well-known and broadly accepted hypothesis that $\mathsf{P} \neq \mathsf{NP}$. A slightly weaker but more compact formulation of ETH (which we will hence adopt in this paper) is the following hypothesis:

> There is no algorithm that solves 3-SAT with n variables and m clauses in time $\mathcal{O}^*(2^{o(n)})$.

The famous *sparsification lemma* of Impagliazzo, Paturi and Zane [39] can hence be stated as follows:

> Assuming ETH, there is no algorithm that solves 3-SAT with n variables and m clauses in time $\mathcal{O}^*(2^{o(n+m)})$.

Notice that this seemingly minor modification gives in fact a tremendous advantage to everybody aiming at proving complexity statements that hold, unless ETH fails. Observe that, in order to prove such complexity results, one also needs a special type of reductions called SERF reductions, see [39], or also the survey [59]. This type of reduction is essentially a subexponential-time

Turing reduction, and the very power of such a type of reduction is exploited in the sparsification lemma. However, the easiest case of such a reduction is in fact a polynomial-time many-one reduction from 3-SAT such that there is a linear dependence of the size measure of the reduced instance on the number of variables and clauses of the given 3-SAT instance. Several (but not all) textbook reductions enjoy this kind of *linearity property*.

It is worth mentioning that not all textbook reductions enjoy this linearity property. For instance, out of the five problems reduced (directly or indirectly) from 3-SAT in Sec. 3.1 of [24], only Vertex Cover and Clique enjoy this property. Conversely, the given reduction from 3-SAT to 3-Dimensional Matching (3-DM) produces $\mathcal{O}(n^2m^2)$ many triples from a given 3-SAT instance with m clauses and n variables. Hence, under ETH we can only rule out algorithms running in time $\mathcal{O}^*(2^{o(\sqrt[4]{t})})$ for 3-Dimensional Matching instances with t triples. So, we might need new (and possibly also more complicated) reductions to make proper use of ETH. This venue is also exemplified by reductions presented in this paper (see Theorems 49 and 54).

Notice that the lower bounds that can be obtained by using published proofs that only go back to 3-SAT by a chain of reductions could be really weak. We make this statement clearer by one concrete example. In the following, we denote the 'loss' that is incurred by a reduction by given the 'root term'; for instance, we have the following losses:

- From 3-SAT to 3-DM: $\sqrt[4]{\cdot}$ [24].
- From 3-DM to 4-Packing: $\sqrt[4]{\cdot}$ [24].
- From 4-Packing to 3-Packing: $\sqrt[2]{\cdot}$ [24].
- From 3-Packing to Exact Block Cover EBC_2: linear [46].
- From EBC_2 to the fixed word problem of a GJFA: linear (Theorem 49).

This chain of reduction would hence incur a loss of $\sqrt[32]{\cdot}$, which also shows the need to exhibit yet another reduction for the purpose of making proper use of ETH (Theorem 54).

Another example for a reduction from 3-SAT that only gives a weak-looking bound is offered by the reduction of Stockmeyer and Meyer (mentioned several time throughout this paper) that shows that the question whether any word is not accepted by a given unary regular language is NP-hard. As can be seen by analyzing that proof, under ETH only the existence of an $\mathcal{O}^*(2^{\sqrt[4]{q-\varepsilon}})$-time algorithm is ruled out (for q-state unary NFA). As presented in [18, Theorem 6], this can be improved to the following statement:

Theorem 56. *Unless ETH fails, there is no $\mathcal{O}^*(2^{o(q^{1/3})})$-time algorithm for deciding, given a unary NFA M on q states, whether $L(M) \neq \{a\}^*$.*

As this seems to be the currently best bound of its kind, we make use of it in Corollaries 47 and 48. Stronger reductions are known for the case of binary input alphabets, as shown in the same paper.