Jumping Finite Automata: Characterizations and Complexity

Henning Fernau, Meenakshi Paramasivan*, and Markus L. Schmid

Fachbereich 4 – Abteilung Informatik, Universität Trier, D-54286 Trier, Germany,
{Fernau,Paramasivan,MSchmid}@uni-trier.de

Abstract. We characterize the class of languages described by jumping finite automata (i. e., finite automata, for which the input head after reading (and consuming) a symbol, can jump to an arbitrary position of the remaining input) in terms of special shuffle expressions. We can characterize some interesting subclasses of this language class. The complexity of parsing these languages is also investigated.

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 [17] and biautomata [13]). One such model that has drawn comparatively little attention are the jumping finite automata (JFA) introduced by Meduna and Zemek [15, 16], which are like classical finite automata with the only difference that after reading (i. e., consuming) a symbol and changing in 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 from around 1980. This also resolves an open problem in [15]. By showing that any such expression is equivalent to one with a star-height (with respect to iterated shuffle) of at most 1, we obtain a normal form for this language class. If we interpret *general* finite automata, i. e., finite automata the transitions of which can be labeled by words instead of single symbols, as jumping automata, then we obtain a much more powerful model. This is demonstrated by showing that the universal word problem for JFA can be solved in polynomial time (for fixed alphabets), whereas it is NP-complete for general JFA, even for finite languages over a fixed binary alphabet.

Due to space restrictions, results marked with (∗) are not proven here.

* Corresponding author.

Jumping Finite Automata. Following Meduna and Zemek, 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* 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:

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

If M is a (general) finite machine, we can hence obtain the following languages:
$L_{\mathrm{FA}}(M) = \{w \in \Sigma^* : \exists\, f \in F : sw \Rightarrow^* f\}$ and
$L_{\mathrm{JFA}}(M) = \{w \in \Sigma^* : \exists\, u, v \in \Sigma^*\ \exists\, f \in F : w = uv \wedge usv \curvearrowright^* f\}$.
This defines us the language classes $\mathcal{REG}$ (accepted by (generalized) finite automata), $\mathcal{JFA}$ (accepted by jumping finite automata, or JFAs for short) and $\mathcal{GJFA}$ (accepted by general jumping finite automata, or GJFAs for short). $\mathcal{CFL}$ denotes the class of context-free languages.

2 Operations on Languages and Their Properties

The reader is assumed to be familiar with the standard operations on formal languages, like catenation, union and iterated catenation, aka Kleene star.

Definition 1. *Let $u, v \in \Sigma^*$, the* shuffle operation*, denoted by $\sqcup\!\sqcup$, is a binary operation on words, described by $u \sqcup\!\sqcup v = \{x_1y_1x_2y_2\ldots x_ny_n : u = x_1x_2\ldots x_n, v = y_1y_2\ldots y_n, x_i, y_i \in \Sigma^*, 1 \leq i \leq n, n \geq 1\}$. It is extended on languages in the natural way: for $L_1, L_2 \subseteq \Sigma^*$, $L_1 \sqcup\!\sqcup L_2 := \{z : z \in x \sqcup\!\sqcup y, x \in L_1, y \in L_2\}$.*

Definition 2. *For $L \subseteq \Sigma^*$, the* iterated shuffle *of L is defined by:*

$$L^{\sqcup\!\sqcup,*} := \bigcup_{n=0}^{\infty} L^{\sqcup\!\sqcup,n} \quad \textit{where } L^{\sqcup\!\sqcup,0} = \{\varepsilon\} \textit{ and } L^{\sqcup\!\sqcup,i} := L^{\sqcup\!\sqcup,i-1} \sqcup\!\sqcup L.$$

Let us now recall the following computation rules from [8].

Proposition 1. *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 when you consider (iterated) catenation instead of (iterated) shuffle. This is no coincidence, as we will see. The sixth rule will play a crucial rôle in the proof of our main normal form result.

We can deduce from the first three computation rules the following:

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

Definition 3. *The set of all permutations of* w, $\mathrm{perm}(w)$, *is defined as follows:*

$$\mathrm{perm}(w) = \begin{cases} \{\varepsilon\}, & |w| = 0 \\ \{a\} \sqcup\!\sqcup \mathrm{perm}(u), & w = a \cdot u, a \in \Sigma, u \in \Sigma^* \end{cases}$$

For $L \subseteq \Sigma^*$, $\mathrm{perm}(L) = \bigcup_{w \in L} \mathrm{perm}(w)$.

We summarize two important properties of perm in the following two lemmas.

Lemma 1. $\mathrm{perm} : 2^{\Sigma^*} \to 2^{\Sigma^*}$ *is a hull operator, i.e., it is extensive, (monotone) increasing and idempotent.*

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

Lemma 2. *The set* $\{\mathrm{perm}(w) : w \in \Sigma^*\}$ *is a partition of* Σ^*. *There is a natural bijection between this partition and the set of functions* $\mathbb{N}^\Sigma$, *given by the* Parikh mapping $\pi_\Sigma : \Sigma^* \to \mathbb{N}^\Sigma, w \mapsto (a \mapsto |w|_a)$, *where* $|w|_a$ *is the number of occurrences of* a *in* w. *Namely,* $\{\mathrm{perm}(w) : w \in \Sigma^*\} = \pi_\Sigma^-(\pi_\Sigma(w))$.

Due to Lemma 2, we conclude:

Proposition 3. *For* $L_1, L_2 \subseteq \Sigma^*$, $\mathrm{perm}(L_1) = \mathrm{perm}(L_2)$ *iff* $\pi_\Sigma(L_1) = \pi_\Sigma(L_2)$.

By the definition of the work 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, we can conclude:

Corollary 1. *If* $L \in \mathcal{JFA}$, *then* L *is perm-closed.*

This also follows by results in [15]. In particular, we mention the following important characterization theorem from [16], that we enrich by combining it with the well-known theorem of Parikh [18], using Proposition 3.

Theorem 1. $\mathcal{JFA} = \mathrm{perm}(\mathcal{RES}) = \mathrm{perm}(\mathcal{CFL})$.

This theorem also generalizes the main result of [14]. According to the analysis indicated in [5], Parikh's original proof would produce, starting from a context-free grammar G with n variables, a regular expression E of length $O\left(2^{2^{n^2}}\right)$ such that $\mathrm{perm}(L(G)) = \mathrm{perm}(L(E))$, whose corresponding NFA is even bigger, while the construction of [5] results in an NFA A with only 4^n states, satisfying $\mathrm{perm}(L(G)) = \mathrm{perm}(L(A))$.

Corollary 2. *Let L be a finite language. Then, $L \in \mathcal{JFA}$ iff L is perm-closed.*

This also shows that all finite $\mathcal{JFA}$ languages are so-called commutative regular languages as studied by Ehrenfeucht, Haussler and Rozenberg in [4]. We will come back to this issue later.

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

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

Clearly, perm cannot be an isomorphism, as the catenation semiring is not commutative, while the shuffle semiring is, see Proposition 2.

3 Alphabetic Shuffle Expressions

Shuffle expressions and variants thereof have been an active field of study over decades; we only point the reader to [9], [10] and [11]. Here, we describe one special variant tightly linked to jumping finite automata. We hence give an inductive definition of what we call *alphabetic shuffle expressions*, or α-SHUF expressions for short, in the following.

Definition 4. *The symbols $\emptyset, \varepsilon$ and each $a \in \Sigma$ are* α-SHUF *expressions (base case). If S_1, S_2 are* α-SHUF *expressions, then $(S_1 + S_2), (S_1 \sqcup\!\sqcup S_2)$ and ${S_1}^{\sqcup\!\sqcup,*}$ are* α-SHUF *expressions.*

The semantics of α-SHUF expressions is defined in the expected way. For instance, $L((a+b)^{\sqcup\!\sqcup,*}) = \{a,b\}^{\sqcup\!\sqcup,*}$. The corresponding class of languages was termed $\mathcal{L}_3$ in [7]. If S_1, S_2 are two expressions, then $S_1 \equiv S_2$ means that they are equivalent, i.e., they describe the same language, or, more formally, $L(S_1) = L(S_2)$. Sometimes, to avoid confusion with arithmetics, we also write $\cup$ in expressions instead of $+$.

Notice that we could introduce (classical) regular expressions in the very same way. Clearly, these characterize the regular languages.

Definition 5. *The symbols $\emptyset, \varepsilon$ and each $a \in \Sigma$ are regular expressions. If S_1, S_2 are regular expressions, then $(S_1 + S_2), (S_1 \cdot S_2)$ and ${S_1}^*$ are regular expressions.*

Lemma 3. *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 2, $\mathrm{perm}(K)$ 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'' of 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 3. *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 1. L' can be described by some regular expression R'. By Lemma 3, we find an α-SHUF expression R such that $L = \mathrm{perm}(L(R')) = L(R)$.

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 3, 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 1. □

Since α-SHUF languages are closed under iterated shuffle, we obtain the following corollary as a consequence of Theorem 3, adding to the list of closure properties given in [15].

Corollary 3. *$\mathcal{JFA}$ is closed under iterated shuffle.*

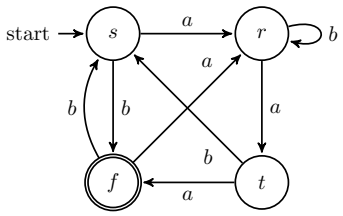
Let us finally mention a second characterization (recall the first characterization from Corollary 2) of the finite perm-closed sets.

Proposition 4. *Let L be some 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 3 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 3 with Corollary 1, we know that $L(R)$ is perm-closed. It is rather straightforward that $L(R)$ is also finite. □

Let us now see an example for the class $\mathcal{JFA}$.

Example 1. The finite machine $M = (\{s, r, t, f\}, \{a, b\}, R, s, \{f\})$ with $R = \{sa \to r, sb \to f, ra \to t, rb \to r, ta \to f, tb \to s, fa \to r, fb \to s\}$ accepts (in terms of traditional regular expressions) $L = L_{\mathrm{FA}}(M)$ with $L = L\left(((ab^*ab)^*((ab^*aa) + b)(ab^*aa)^*((ab^*ab) + b))^*(ab^*ab)^*((ab^*aa) + b)(ab^*aa)^*\right)$. The same M accepts (in terms of α-SHUF expressions) $L = L_{\mathrm{JFA}}(M)$ with $L = L(((a \sqcup b^{\sqcup,*} \sqcup a \sqcup b)^{\sqcup,*} \sqcup ((a \sqcup b^{\sqcup,*} \sqcup a \sqcup a) + b) \sqcup (a \sqcup b^{\sqcup,*} \sqcup a \sqcup a)^{\sqcup,*} \sqcup ((a \sqcup b^{\sqcup,*} \sqcup a \sqcup b) + b))^{\sqcup,*} \sqcup (a \sqcup b^{\sqcup,*} \sqcup a \sqcup b)^{\sqcup,*} \sqcup ((a \sqcup b^{\sqcup,*} \sqcup a \sqcup a) + b) \sqcup (a \sqcup b^{\sqcup,*} \sqcup a \sqcup a)^{\sqcup,*})$.



4 Representations and Normal Forms

Our desired representation theorem can be stated as follows.

Theorem 4. *Let $L \in \mathcal{JFA}$. Then there exists a number $n \geq 1$ and finite sets M_i, N_i for $1 \leq i \leq n$, so that the following representation is valid.*

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

We will prove this representation theorem on the level of α-SHUF expressions, so that we actually get a normal form theorem for these. A central tool in the proofs of this normal form theorem is the following notion that corresponds to the well-known star-height of regular expressions.

Definition 6. *We can inductively associate the (*shuffle iteration*)* height *h to any α-*SHUF *expression S as follows.*

- *If S is a base case, then $h(S) = 0$.*
- *If $S = (S_1 + S_2)$ or $S = (S_1 \sqcup S_2)$, then $h(S) = \max\{h(S_1), h(S_2)\}$.*
- *If $S = S_1^{\sqcup,*}$, then $h(S) = h(S_1) + 1$.*

The shuffle iteration height of a $\mathcal{JFA}$-language L is then the smallest shuffle iteration height of any α-SHUF expression S describing L.

Let us mention the following interesting consequence obtained by combining Theorem 4 with Theorem 3, Lemma 3 and Theorem 1.

Corollary 4. *$L \in \mathcal{JFA}$ if and only if there is a regular language R of star height at most one such that $L = \mathrm{perm}(R)$.*

Immediately from the Definition 6, we obtain from Proposition 4:

Corollary 5. *A language is finite and perm-closed if and only if it can be described by some α-*SHUF *expression of shuffle iteration height zero.*

Combining Corollary 5 with Theorem 1 and the well-known fact that finiteness of regular expressions can be decided, we immediately obtain the following, as Theorem 4 guarantees that the height of $\mathcal{JFA}$ languages is zero or one:

Corollary 6. *It is decidable, given some JFA and some integer k, whether or not this JFA describes a language of shuffle iteration 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 [6]. Recall that Eggan's Theorem [3] relates the star height of a regular language to its so-called cycle rank, which formalizes loop-nesting in NFA's. 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 $L_{\mathrm{JFA}}(M) = L$.

Corollary 5 means that, in order to show Theorem 4, it is sufficient (and in a sense stronger) to prove the following normal form theorem for α-SHUF expressions. The proof resembles the one given by Jantzen [8] for a different variant of shuffle expressions, but we keep it here, as it shows several technicalities with these notions.

Theorem 5. *For any α-SHUF expression R, an equivalent α-SHUF expression S with $h(S) = 1$ can be constructed that is the union of n α-SHUF expressions $S_1, \ldots, S_n$ such that $S_i = F_i \sqcup\!\sqcup G_i^{\sqcup\!\sqcup,*}$, where $h(F_i) = h(G_i) = 0$, $1 \leq i \leq n$. Moreover, we can assume that $F_i = \bigcup_{j=1}^{n(i)} u_j$ and $G_i = \bigcup_{j=1}^{m(i)} v_j$, where all u_j and v_j are α-SHUF expressions with $\sqcup\!\sqcup$ as their only operators.*

Proof. We show the claim by induction on the height of R. If $h(R) = 0$, then $S = R \sqcup\!\sqcup \emptyset^{\sqcup\!\sqcup,*}$ is an equivalent expression in the desired normal form. Let $h > 0$. Assume now that the result is true for all α-SHUF expressions of height less than h and consider some α-SHUF expression R with $h(R) = h$. By repeatedly applying the distributive law, we can obtain an equivalent α-SHUF expression R' that is of the following form:

$$R' = \bigcup_{j=1}^{m} \bigsqcup_{k=1}^{k(j)} S_{j,k}\,,$$

where each expression $S_{j,k}$ contains only the operators shuffle and iterated shuffle. In a first step, by applying the commutative law of the shuffle, we can order the $S_{j,k}$ such that, slightly abusing notation, $S_{j,1}, \ldots, S_{j,b(j)}$ are base cases, and $S_{j,b(j)+1}, \ldots, S_{j,k(j)}$ are of the form $S_{j,i} = (T_{j,i})^{\sqcup\!\sqcup,*}$. To simplify the further discussions, we can assume that none of the base cases $S_{j,1}, \ldots, S_{j,b(j)}$ is $\emptyset$, as this would mean that the language $L(\bigsqcup_{k=1}^{k(j)} S_{j,k})$ is empty, and we can omit this part immediately from the union. In the next step, we form $F'_j := \bigsqcup_{k=1}^{b(j)} S_{j,k}$. Notice that, by Corollary 5, each F'_j represents a finite perm-closed set. Moreover, we define α-SHUF expressions G'_j of iteration height less than h as follows.

If $b(j) = k(j)$, then $G'_j := \emptyset$. Otherwise, $G'_j := \bigcup_{i=b(j)+1}^{k(j)} T_{j,i}$. By using Rule 4 from Proposition 1, one can see that

$$R'' := \bigcup_{j=1}^{m} F'_j \sqcup\!\sqcup (G'_j)^{\sqcup\!\sqcup,*}$$

is equivalent to R'. As all G'_j have iteration height less than h, we can apply the induction hypothesis to them and replace G'_j by equivalent expressions

$$\bigcup_{i=1}^{n(j)} F_{j,i} \sqcup\!\sqcup G_{j,i}^{\sqcup\!\sqcup,*} ,$$

where each $F_{j,i}$ and each $G_{j,i}$ are α-SHUF expressions of height zero. Rule 4 now yields the following equivalent expression:

$$R''' := \bigcup_{j=1}^{m} F'_j \sqcup\!\sqcup \bigsqcup_{i=1}^{n(j)} \left(F_{j,i} \sqcup\!\sqcup G_{j,i}^{\sqcup\!\sqcup,*}\right)^{\sqcup\!\sqcup,*}$$

Now, we can apply Rule 6 to avoid nesting of the iterated shuffle. Hence, the following expression is again equivalent:

$$R^{iv} := \bigcup_{j=1}^{m} F'_j \sqcup\!\sqcup \bigsqcup_{i=1}^{n(j)} (F_{j,i} \sqcup\!\sqcup (F_{j,i} \cup G_{j,i})^{\sqcup\!\sqcup,*} \cup \{\varepsilon\})$$

Finally, setting $F_{j,I} := F'_j \sqcup\!\sqcup \bigsqcup_{i\in I} F'_{j,i}$ and $G_{j,I} := \bigcup_{i\in I}(F_{j,i} \cup G_{j,i})$ for $I \subseteq I(j) := \{1, \ldots, n(j)\}$, with $F_{j,\emptyset} = F'_j$ and $G_{j,\emptyset} = \emptyset$, and observing that also these α-SHUF expressions are of height zero, we define

$$S := \bigcup_{j=1}^{m} \bigcup_{I\subseteq I(j)} F_{j,I} \sqcup\!\sqcup G_{j,I}^{\sqcup\!\sqcup,*} .$$

By the commutative and distributive laws and by Rule 4, S is equivalent to R^{iv} and satisfies all the properties of the theorem, possibly apart from the last sentence, which can be enforced by exhaustively applying the distributive law.

□

Unfortunately, the construction of Theorem 5 could blow up the size of the resulting expression exponentially. This does not harm the statement of the theorem, and also Theorem 4 follows immediately. For algorithmic purposes, this is indeed a drawback, because this also means that the running time of an algorithm (derived from the proof of Theorem 5) would be exponential in the length of the input expression.

Therefore, we establish the following weaker normal form result that can be, however, obtained in time that can be described within the framework of parameterized complexity [2]. In this framework, certain parts of the input are singled

out as so-called parameters. In our case, it will be the number of iterated shuffle operator occurrences, as well as the shuffle iteration height of the expression. We will then present an algorithm whose only exponential-time dependencies is on these two parts of the input. In other words, if both are fixed (or if we consider only expressions with a certain upper bound on these parameters as inputs), we obtain a polynomial-time transformation algorithm.

This is an interesting fact in itself, as it also raises the descriptional complexity question if the blow-up formally described below is indeed necessary. We are not aware of any work that can be considered as "parameterized descriptional complexity", which might be therefore an interesting (new) subject on its own, motivated by the construction below.

Let us first describe the idea and some of the details of the construction that we have in mind here. As we are aiming at obtaining some equivalent α-SHUF expression of shuffle iteration height at most one, we can assume that the expression that we start with has a height of at least two. When we want to measure the size of an α-SHUF expression E, we simply count the number of all occurrences of operators in the expression, and we denote this by $s(E)$. Clearly, if we consider E as a word over Σ (plus operator symbols and parentheses), then the length of E is bounded by a linear function in $s(E)$. First of all, observe that each iterated shuffle operator occurrence in some α-SHUF expression E can be viewed as the outermost operator of a subexpression F of E that is of a certain shuffle iteration height $h(F)$. For the sake of convenience, we can hence associate a shuffle iteration height also to occurrences of shuffle operators. Let $ISO(E)$ collect all iterated shuffle operator occurrences of expression E and $ISO_h(E)$ those of shuffle iteration height h. Hence,

$$ISO(E) = \bigcup_{h=1}^{h(E)} ISO_h(E)\,.$$

If E is an α-SHUF expresssion over the alphabet Σ, then let $\Sigma_1, \ldots, \Sigma_{h(E)}$ be fresh alphabets containing new letters, with $|\Sigma_i| = |ISO_i(E)|$ and hence a natural bijection $\psi_i : ISO_i(E) \to \Sigma_i$. Now, consider the α-SHUF expresssion E' obtained from E by replacing, for $h = 1, \ldots, h(E) - 2$, the subexpression whose outermost operator is some iterated shuffle occurrence $j \in ISO_h(E)$ by the letter $\psi_h(j)$, for all occurrences in $ISO_h(E)$. As by our assumption $h(E) \geq 2$, $h(E') = 2$. So, $ISO_{h(E)}(E) = ISO_2(E')$ and $ISO_{h(E)-1}(E) = ISO_1(E')$. In the following, we are considering all $2^{|ISO_1(E')|}$ many subsets of $ISO_1(E')$. We will convert accordingly derived expressions into equivalent ones of star height one. Proceeding inductively, we can finally show:

Theorem 6. *(*) For any* α-SHUF *expression* R*, an equivalent* α-SHUF *expression* S *with* $h(S) \leq 1$ *can be constructed in time* $O^*(2^{|ISO(R)|}2^{h(R)})$*; the resulting expression could be as big as this.*

Notice that the O^*-notation suppresses polynomial factors, which is a very suitable notation in the area of Parameterized Complexity. This shows that the transformation of R into normal form is in FPT, with parameter $|ISO(R)|$.

5 Comparing $\mathcal{JFA}$ and $\mathcal{RES}$

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

Proposition 5. *$L \in \mathcal{JFA} \cap \mathcal{RES}$ iff $L \in \mathcal{RES}$ and L is perm-closed.*

We mention this, as the class $\mathcal{JFA} \cap \mathcal{RES}$ can be also characterized as follows according to Ehrenfeucht, Haussler and Rozenberg [4]. Namely, they describe this class of (what they call) commutative regular languages as finite unions of periodic languages. We are not giving a definition of this notion here, but rather state an immediate consequence of their characterization in our terminology.

Theorem 7. *Let $L \subseteq \Sigma^*$. Then, $L \in \mathcal{JFA} \cap \mathcal{RES}$ 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 $\mathcal{JFA} \cap \mathcal{RES}$ was derived in [14, Theorem 3].

6 Complexity of Parsing

For a fixed JFA M, we can decide, for a given word w, whether $w \in L(M) \subseteq \Sigma^*$ in the following way. We scan over w and construct its Parikh mapping $\pi_\Sigma(w)$. Then we simulate a computation of M on w by nondeterministically choosing in every state the transition labelled by some symbol and decrementing the corresponding component of $\pi_\Sigma(w)$. If we reach an accepting state with all components of $\pi_\Sigma(w)$ being 0, then we conclude $w \in L(M)$. In this procedure, we only have to store the Parikh mapping, which only requires logarithmic space; thus, this shows $\mathcal{JFA} \subseteq \mathsf{NL} \subseteq \mathsf{P}$.[1]

These considerations show that the *fixed* word problem can be solved in polynomial time. In the following, we look at the *universal* word problem for (generalized) jumping finite automata, which is to decide for a given (general) finite machine M with input alphabet Σ and a word $w \in \Sigma^*$, whether or not $w \in L_{\mathrm{JFA}}(M)$. The study of this problem was explicitly suggested in [15], where only the mere decidability status was resolved.

We first show that the universal word problem for jumping finite automata can be solved in polynomial time, provided that the alphabet is fixed.

[1] We wish to point out that this also follows from results in [1], where containment in NL is shown for a superclass of $\mathcal{JFA}$.

Theorem 8. *For any fixed alphabet, the universal word problem for jumping finite automata is polynomial-time solvable.*

Proof. Let $M = (Q, \Sigma, R, s, F)$ be a finite machine over $\Sigma = \{a_1, a_2, \ldots, a_k\}$ and let $w \in \Sigma^*$. We define a directed graph $\mathcal{G}_M = (V_M, E_M)$, where V_M contains all elements $(p, (\ell_1, \ell_2, \ldots, \ell_k))$ with $p \in Q$ and, for every i, $1 \leq i \leq k$, $0 \leq \ell_i \leq |w|_{a_i}$, and $E_M \subseteq V_M \times V_M$ contains all pairs $((p, (\ell_1, \ell_2, \ldots, \ell_k)), (p', (\ell'_1, \ell'_2, \ldots, \ell'_k)))$ such that there is a rule $pa_i \to p' \in R$, $\ell'_i = \ell_i - 1$ and, for every j, $1 \leq j \leq k$, with $i \neq j$, $\ell'_j = \ell_j$. We note that $|\mathcal{G}_M| \leq (|w|^k|Q|)^2$ and that $\mathcal{G}_M$ can be constructed in time $\mathcal{O}(|\mathcal{G}_M|)$. The graph $\mathcal{G}_M$ corresponds to the computation of M on input w: a vertex is a configuration consisting of the current state and the Parikh mapping of the remaining input and there is an edge between two configurations if it is possible to reach one from the other by the application of a rule. Hence, $w \in L_{\mathrm{JFA}}(M)$ if and only if there exists a path in $\mathcal{G}_M$ from $(s, \pi_\Sigma(w))$ to some vertex $(q, (0, 0, \ldots, 0))$ with $q \in F$. This property can be decided in time $\mathcal{O}(|\mathcal{G}_M|)$. □

The decision procedure of Theorem 8 is only polynomial if the alphabet size is a constant, which for most real-world applications is the case. From a theoretical point of view, it would nevertheless be interesting to know whether a polynomial time procedure for unbounded alphabets is possible.

Next, we show that if the JFA-language is given as an α-SHUF expression in the normal form of Theorem 5, then the universal word problem can be solved in polynomial time also for unbounded alphabets.

Theorem 9. *The universal word problem is polynomial-time solvable for* α-SHUF *expressions in normal form.*

Proof. Let $R = \bigcup_{i=1}^{n} R_i \sqcup\!\sqcup (R'_i)^{\sqcup\!\sqcup,*}$ be the α-SHUF expression in the normal form of Theorem 5. Moreover, we define $M_i = L(\widehat{R_i})$ and $N_i = L(\widehat{R'_i})$, where $\widehat{R_i}$ and $\widehat{R'_i}$ are the regular expressions obtained from R_i and R'_i by replacing every shuffle operation by a catenation operation. We can convert each of the n parts of the union into linear equations as follows. Let $w \in \Sigma^*$ be the input word. Then, $w \in L(R_i \sqcup\!\sqcup (R'_i)^{\sqcup\!\sqcup,*})$ if and only if there is a non-negative integer solution of one of the linear equations

$$\pi_\Sigma(w) = \pi_\Sigma(u) + \sum_{v \in N_i} x_v \pi_\Sigma(v)\,,$$

where $u \in M_i$. Since the expressions R_i, R'_i are unions of shuffles of single symbols, $\sum_{u \in M_i} |u|$ and $\sum_{v \in N_i} |v|$ are linear in $|R_i|$ and $|R'_i|$, respectively. Thus, each of the equations is of polynomial size in terms of the size of R. Each of these linear equations can be analyzed by Gaussian elimination in polynomial time. Altogether, this proves the claim. □

If the input finite machine is allowed to be a *general* finite machine, then the complexity of the universal word problem increases considerably, i. e., it becomes NP-complete even for general finite machines accepting *finite* language over *binary* alphabets.

Theorem 10. *The universal word problem is NP-complete for generalized jumping finite automata (even for finite languages over binary alphabets).*

We can simulate a given generalized jumping finite automaton on a word by guessing where to jump and which rules to apply. Since the number of guesses is cleary bounded by the length of the input word, this shows that the universal word problem is in NP.

It remains to prove the NP-hardness of this problem, which can be done by a reduction from the following problem.

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

By EBC_2, we denote the restricted version of EBC, where Σ is a fixed binary alphabet. It has recently been shown in [12] that EBC_2 is NP-complete.

Let $u_1, u_2, \ldots, u_k, v \in \Sigma^*$ be an instance of EBC_2, where $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. For the sake of convenience, we define $u_i = s_{i,1} s_{i,2} \ldots s_{i,\ell_i}$, $s_{i,j} \in \Sigma$, $1 \leq i \leq k$, $1 \leq j \leq \ell_i$, and $v = t_1 t_2 \ldots t_m$, $t_j \in \Sigma$, $1 \leq j \leq m$. Furthermore, for every j, $1 \leq j \leq 2m$, we define the j^{th} *separator* $\star_j = \mathtt{a}\,\mathtt{b}^{j+m}\,\mathtt{a}$. For every i, $1 \leq i \leq k$, u_i is transformed into $A_i = \{\star_j\, s_{i,1} \star_{j+1} s_{i,2} \ldots \star_{j+\ell_i-1} s_{i,\ell_i} : 1 \leq j \leq m\}$ and v is transformed into $\widehat{v} = \star_1 t_1 \star_2 t_2 \ldots \star_m t_m$. We note that, for every j, $1 \leq j \leq m$, there is exactly one unique occurrence of the j^{th} separator in $\widehat{v}$ and all these occurrences of separators are non-overlapping. Finally, we define a general finite machine $M = (Q, \Sigma, R, q_0, F)$ by $Q = \{q_0, q_1, q_2, \ldots, q_k\}$, $R = \bigcup_{i=1}^{k} \{q_{i-1} w \to q_i : w \in A_i\}$ and $F = \{q_k\}$. This reduction is obviously polynomial.

We give a proof sketch for the correctness of this reduction. To this end, let $(u_1, u_2, \ldots, u_k, v)$ be a positive instance of EBC_2. Then $v = u_{\pi(1)} \ldots u_{\pi(k)}$ for some permutation $\pi : \{1, 2, \ldots, k\} \to \{1, 2, \ldots, k\}$. If we insert the j^{th} separator after the j^{th} symbol of $u_{\pi(1)} \ldots u_{\pi(k)}$, then we obtain $\widehat{v} = w_{\pi(1)} w_{\pi(2)} \ldots w_{\pi(k)}$ with $w_{\pi(i)} \in A_{\pi(i)}$, $1 \leq i \leq k$; thus, $\widehat{v} \in L_{\mathrm{JFA}}(M)$, which yields the following.

Lemma 4. *(*) If $(u_1, u_2, \ldots, u_k, v) \in \mathrm{EBC}_2$, then $\widehat{v} \in L_{JFA}(M)$.*

If, on the other hand, $\widehat{v} \in L_{\mathrm{JFA}}(M)$, then $v = u_{\pi(1)} u_{\pi(2)} \ldots u_{\pi(k)}$ can only be concluded if M never erases a factor that does not correspond to an original factor of $\widehat{v}$ (or, equivalently, if M never erases a factor that contains consecutive symbols that do not correspond to consecutive symbols of $\widehat{v}$). This property is enforced by the separator words; thus, we can conlude the following.

Lemma 5. *(*) If $\widehat{v} \in L_{JFA}(M)$, then $(u_1, u_2, \ldots, u_k, v) \in \mathrm{EBC}_2$.*

Theorems 8 and 10 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.

References

1. 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.
2. R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.
3. L. C. Eggan. Transition graphs and the star-height of regular events. *The Michigan Mathematical Journal*, 10(4):385–397, 12 1963.
4. A. Ehrenfeucht, D. Haussler, and G. Rozenberg. On regularity of context-free languages. *Theoretical Computer Science*, 27:311–332, 1983.
5. 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.
6. K. Hashiguchi. Algorithms for determining relative star height and star height. *Information and Computation*, 78(2):124–169, 1988.
7. 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.
8. M. Jantzen. Eigenschaften von Petrinetzsprachen. Technical Report IFI-HH-B-64, Fachbereich Informatik, Universität Hamburg, Germany, 1979.
9. M. Jantzen. The power of synchronizing operations on strings. *Theoretical Computer Science*, 14:127–154, 1981.
10. M. Jantzen. Extending regular expressions with iterated shuffle. *Theoretical Computer Science*, 38:223–247, 1985.
11. J. Jedrzejowicz and A. Szepietowski. Shuffle languages are in P. *Theoretical Computer Science*, 250(1-2):31–53, 2001.
12. 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.
13. O. Klíma and L. Polák. On biautomata. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 46:573–592, 2012.
14. M. Latteux and G. Rozenberg. Commutative one-counter languages are regular. *Journal of Computer and System Sciences*, 1:54–57, 1984.
15. A. Meduna and P. Zemek. Jumping finite automata. *International Journal of Foundations of Computer Science*, 23(7):1555–1578, 2012.
16. A. Meduna and P. Zemek. Chapter 17: Jumping finite automata. In *Regulated Grammars and Automata*, pages 567–585. Springer, New York, 2014.
17. 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.
18. R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.