On comparing deterministic finite automata and the shuffle of words ⋆ ⋆⋆

Franziska Biegler[1] and Ian McQuillan[2]

[1] Machine Learning Group, TU Berlin
Germany
franziska.biegler@tu-berlin.de
[2] Department of Computer Science, University of Saskatchewan
Saskatoon, SK S7N 5A9, Canada
mcquillan@cs.usask.ca

Abstract. We continue the study of the shuffle of individual words, and the problem of decomposing a finite automaton into the shuffle on words. There is a known polynomial time algorithm to decide whether the shuffle of two words is a subset of the language accepted by a deterministic finite automaton [5]. In this paper, we consider the converse problem of determining whether or not the language accepted by a deterministic finite automaton is a subset of the shuffle of two words. We provide a polynomial time algorithm to decide whether the language accepted by a deterministic finite automaton is a subset of the shuffle of two words, for the special case when the skeletons of the two words are of fixed length. Therefore, for this special case, we can decide equality in polynomial time as well. However, we then show that this problem is coNP-Complete in general, as conjectured in [2].

1 Introduction

The shuffle operation (denoted by $\sqcup\!\sqcup$ here) on words describes the set of all words that can be obtained by interleaving the letters of the operands in all possible ways, such that the order of the letters of each operand is preserved (the operation can then be extended to languages). There have been a number of theoretical results and algorithms involving shuffle such as [10] which showed that the so-called *shuffle languages* obtained from finite languages via union, concatenation, Kleene star, shuffle and shuffle closure, are in P. In [12], it is shown that given a word w, and n other words, it is NP-Complete to decide if w is in the shuffle of the n words.

Despite the length of time since the operator was introduced [7], there remains a number of standard formal language theoretic questions involving shuffle that are unsolved. For example, there is a long-standing open problem as to

⋆ Research supported, in part, by the Natural Sciences and Engineering Research Council of Canada.

⋆⋆ Published in Proceedings of CIAA 2014. The final authenticated version is available online at `https://doi.org/10.1007/978-3-319-08846-4_7`.

whether it is decidable to decompose an arbitrary regular language into the shuffle of two languages. Certain special cases are known to be decidable however, such as for commutative regular languages and locally testable languages, while it is undecidable for context-free languages [6].

Indeed, even the special case of the shuffle of individual words, rather than sets of words, has received considerable attention but there remains a number of yet unsolved problems. In [1], it is shown that the shuffle of individual words (with at least two letters) has a unique shuffle decomposition over words. That result was extended in [3] to show that the shuffle of two words (each with at least two letters) has a unique shuffle decomposition over arbitrary sets.

However, the complexity of taking a language as input, and determining if it has a decomposition into the shuffle of two words, remains an open question (which also depends on the method that the language uses as input). Despite this, in [5], it is shown that if a language accepted by a deterministic finite automaton (DFA) M has a decomposition into words, there is an algorithm that finds the unique decomposition into words in time linear in the lengths of the words (sublinear in the size of the automaton). However, if the input automaton is not decomposable, the algorithm cannot always determine that it is not decomposable, but will instead in those cases output two strings u and v, despite $L(M)$ not having any shuffle decomposition. As the algorithm does not have knowledge regarding whether $L(M)$ has a decomposition, one could take the output strings u and v, and test if their shuffle is equal to $L(M)$, thus testing whether $L(M)$ was itself decomposable. One way to do this would be to construct a DFA accepting $u \sqcup v$, and test equality with $L(M)$, however it was shown in [4] that the size of minimal DFAs accepting the shuffle of two strings can grow exponentially in the length of the strings. Therefore, it still remains an open problem as to whether there is a polynomial time algorithm to test if the language accepted by a DFA has a decomposition into the shuffle of words.

Here, we are interested in testing inclusion between the language accepted by a DFA and the shuffle of two words. One direction of this problem, testing whether the shuffle of two strings is contained in the language accepted by a DFA has a known polynomial time algorithm [5,2]. In this paper, we investigate the complexity of the converse of this problem. It is shown that given a DFA M and words $u, v \in \Sigma^+$, the problem of deciding whether or not $L(M) \subseteq u \sqcup v$ is coNP-Complete, as conjectured in [2]. However, for the special case of the problem on words u, v with fixed-length skeletons (the length of a skeleton is the number of "lettered sections"), we provide a polynomial time algorithm. This also gives a polynomial time algorithm to decide if $L(M) = u \sqcup v$ for this special case. However, the exact complexity of deciding whether $L(M) = u \sqcup v$ in general remains open despite the fact that we know it takes polynomial time to check $u \sqcup v \subseteq L(M)$ and it is coNP-Complete to check $L(M) \subseteq u \sqcup v$.

2 Preliminaries

Let $\mathbb{N}_0$ be the set of non-negative integers. An alphabet Σ is a finite, non-empty set of letters. The set of all words over Σ is denoted by Σ^*, and this set contains the empty word, λ. The set of all non-empty words over Σ is denoted by Σ^+. For $n \in \mathbb{N}_0$, let Σ^n be all words of length n over Σ.

Let Σ be an alphabet. For a word $w \in \Sigma^*$, the length of w is denoted by $|w|$. Let $w(i)$ be the i-th letter of w, let $w[i]$ be the word which is the first i characters of w, and let $w[i,j]$ be the subword between characters i and j where these are undefined if i or j are not in $\{1,\ldots,|w|\}$, or if $j < i$. The skeleton of w is λ if $w = \lambda$, and is $a_1 a_2 \cdots a_n$ where $w = a_1^{\alpha_1} a_2^{\alpha_2} \cdots a_n^{\alpha_n}, n \geq 1, \alpha_i > 0, a_i \in \Sigma, 1 \leq i \leq n, a_j \neq a_{j+1}, 1 \leq j < n$. For example, the skeleton of *aaaaabbbabbbb* is *abab*.

Let $u, v \in \Sigma^*$. The *shuffle* of u and v is defined as $u \sqcup v = \{u_1 v_1 \cdots u_n v_n \mid u = u_1 \cdots u_n, v = v_1 \cdots v_n, u_i, v_i \in \Sigma^*, 1 \leq i \leq n\}$. For example, $aab \sqcup ba = \{aabba, aabab, ababa, abaab, baaba, baaab\}$. We say u is a *prefix* of v, written $u \leq_{\mathrm{p}} v$, if $v = ux$, for some $x \in \Sigma^*$. Let $w, x \in \Sigma^*$. The left quotient of x by w, written $w^{-1}x = x_1$ if $x = wx_1$, and undefined otherwise.

We assume the reader to be familiar with deterministic finite automata (DFAs), nondeterministic finite automata (NFAs), the subset construction commonly used to convert an NFA into an equivalent DFA, and minimal DFAs. See [13,9] for an introduction and more details on finite automata.

3 Fixed-Length Skeleton Polynomial Algorithm

The purpose of this section is to give special cases on an input DFA M and $u, v \in \Sigma^+$ whereby there is a polynomial algorithm to decide whether or not $L(M) \subseteq u \sqcup v$. In particular, the main result is that when u and v have fixed-length skeletons, there is a polynomial time algorithm. We will see in the next section that in general, this problem is coNP-Complete and therefore there likely is not a polynomial time algorithm (unless $P = coNP$).

We will start by examining the complement of the problem. That is, the problem of whether there exists some $w \in L(M)$ such that $w \notin u \sqcup v$ (or whether $L(M) \not\subseteq u \sqcup v$). If we can provide a polynomial time algorithm to solve this problem for some special cases, then we can solve the problem of $L(M) \subseteq u \sqcup v$ for those cases.

Given two words $u, v \in \Sigma^+$, there is an "obvious" NFA accepting $u \sqcup v$ with $(|u| + 1) \cdot (|v| + 1)$ states, where each state is an ordered pair representing the position within both u and v. This NFA is called the *naive NFA* and is defined formally in [4].

First notice, that if we could construct a DFA accepting $u \sqcup v$ in polynomial time, then we could build a DFA accepting $(u \sqcup v)^c$ (the complement) in polynomial time, using the standard algorithm to take the complement of a DFA (Theorem 3.2, [9]). Similarly, we could build a DFA accepting $L(M) \cap (u \sqcup v)^c$ in polynomial time, using the standard algorithm for taking the intersection of

two DFAs (Theorem 3.3, [9]). Moreover we could test whether this set is empty in polynomial time (Theorem 3.7, [9]).

Proposition 1. *Let M be any DFA. Let $\mathcal{F}$ be a subset of $\Sigma^+ \times \Sigma^+$ such that there exists a polynomial f from $\mathbb{N}_0 \times \mathbb{N}_0$ to $\mathbb{N}_0$ and an algorithm A converting any pair $(u, v) \in \mathcal{F}$ to a DFA accepting $u \sqcup v$ in time less than or equal to $f(|u|, |v|)$. Then we can test whether or not $L(M) \subseteq u \sqcup v$ in polynomial time, for any $(u, v) \in \mathcal{F}$. Similarly for testing whether $L(M) = u \sqcup v$.*

The ability to test for equality follows from the existing polynomial time algorithm to determine if $u \sqcup v \subseteq L(M)$ [5].

In particular, the algorithm A from this proposition could simply be to construct the naive NFA for $u \sqcup v$ and then to use the standard subset construction algorithm applied to the naive NFA accepting $u \sqcup v$ for $(u, v) \in \mathcal{F}$, which could produce DFAs that are polynomial in size for certain sets $\mathcal{F}$. Therefore, if there is a subset $\mathcal{F}$ of $\Sigma^+ \times \Sigma^+$ such that the subset construction applied to the naive NFAs create DFAs that are polynomial in size, then we have a polynomial time algorithm to decide if $L(M) \subseteq u \sqcup v$, for all $(u, v) \in \mathcal{F}$.

Before establishing the main result of this section, we first need the following definition and three lemmas.

Let $u, v, w \in \Sigma^+$, where $w = a_1^{\alpha_1} a_2^{\alpha_2} \cdots a_n^{\alpha_n}, a_i \neq a_{i+1}, 1 \leq i < n, \alpha_l > 0, a_i \in \Sigma, 1 \leq l \leq n$. For each l, $0 \leq l \leq n$, let

$$g(w, u, v, l) = \{(i, j) \mid a_1^{\alpha_1} a_2^{\alpha_2} \cdots a_l^{\alpha_l} \in u[i] \sqcup v[j] \text{ and either } u(i+1) = a_{l+1} \text{ or } v(j+1) = a_{l+1} \text{ or } (i = |u| \text{ and } j = |v|)\}.$$

Note that if $i = |u|$ then $u(i+1)$ is undefined forcing $u(i+1) = a_{l+1}$ to not be true, as with the case where $j = |v|$.

For example, if $u = aabbaabb, v = aabbaaa$ and $w = aabbaabbaabbaaa$, then $g(w, u, v, 3) = \{(6, 0), (4, 2), (2, 4)\}$.

The definition of g can be rewritten recursively as follows:

Lemma 1. *For all l, $1 \leq l \leq n$,*

$$g(w, u, v, l) = \{(h, m) \mid (i, j) \in g(w, u, v, l-1), u[i+1, h] = a_l^{\gamma}, v[j+1, m] = a_l^{\delta} \text{ for some } \gamma, \delta \geq 0, \text{either } (h, m) = (|u|, |v|) \text{ or } u(h+1) = a_{l+1} \text{ or } v(m+1) = a_{l+1}\}.$$

Next, we will show the following three conditions are equivalent, and then use condition 1 within the decision procedure below.

Lemma 2. *The following conditions are equivalent:*

1. *$g(w, u, v, l) \neq \emptyset$ for all l, $0 \leq l \leq n$, and $(|u|, |v|) \in g(w, u, v, n)$,*
2. *$(|u|, |v|) \in g(w, u, v, n)$,*
3. *$w \in u \sqcup v$.*

This can be seen as conditions 2 and 3 are equivalent from the definition of g. Further, condition 1 implies 2 directly. And condition 3 implies 1 as $w \in u \sqcup\!\sqcup v$ implies that for all l, $0 \leq l < n$, $a_1^{\alpha_1} \cdots a_l^{\alpha_l} a_{l+1} \in u[i] \sqcup\!\sqcup v[j]$ for some i, j.

Next, we see that as long as the g function is of size bounded by a constant for each $w \in L(M)$, there is a polynomial algorithm.

Lemma 3. *Let k be a constant, M a DFA and $u, v \in \Sigma^+$. If, for every $w \in L(M)$ where $w = a_1^{\alpha_1} \cdots a_n^{\alpha_n}, a_i \neq a_{i+1}, 1 \leq i < n, \alpha_j > 0, a_j \in \Sigma, 1 \leq j \leq n$, and for every $l, 0 \leq l \leq n$, the inequality $|g(w, u, v, l)| \leq k$ is true, then there is a polynomial time algorithm for deciding whether or not $L(M) \subseteq u \sqcup\!\sqcup v$, and for deciding whether $L(M) = u \sqcup\!\sqcup v$.*

Proof. We will construct a logspace bounded nondeterministic Turing machine for the algorithm, and use the fact that NLOGSPACE is a subset of P (Corollary to Theorem 7.4 in [11]). It will decide if $L(M) \not\subseteq u \sqcup\!\sqcup v$; that is, if there exists w such that $w \in L(M)$ but $w \notin u \sqcup\!\sqcup v$. We will use condition 1 of Lemma 2 combined with Lemma 1. The Turing Machine will guess a word $w \in L(M)$, where $w = a_1^{\alpha_1} \cdots a_n^{\alpha_n}, a_i \neq a_{i+1}, 1 \leq i < n, \alpha_l > 0, a_l \in \Sigma, 1 \leq l \leq n$, and for every l from 0 to n, writes out the list of all (i, j) such that $a_1^{\alpha_1} \cdots a_l^{\alpha_l} \in u[i] \sqcup\!\sqcup v[j]$, where either $(i, j) = (|u|, |v|)$, or $u(i+1) = a_{l+1}$, or $v(j+1) = a_{l+1}$, which must be of size at most k. Indeed this can be done by at first writing out the elements of $g(w, u, v, 0)$ ($(0, 0)$ if either $u(1)$ or $v(1)$ is equal to a_1, and $\emptyset$ otherwise). Then, if after writing out the list $\{(i_1, j_1), \ldots, (i_q, j_q)\}$ after reading $a_1^{\alpha_1} \cdots a_l^{\alpha_l}, l < n$, then for $a_{l+1}^{\alpha_{l+1}}$, the new list is obtained as follows: by taking each $(i_p, j_p), 1 \leq p \leq q$, removing it from the list and adding $(i_p{+}\gamma, j_p{+}\delta)$, where $\gamma{+}\delta = \alpha_{l+1}$, such that the subword of u starting at position $i_p{+}1$ is a_{p+1}^{γ}, and the subword of v starting at position $j_p{+}1$ is a_{p+1}^{δ} and either $(i_p{+}\gamma, j_p{+}\delta) = (|u|, |v|)$, or $a_{p+1}^{\gamma} a_{p+2}$ is a subword of u starting at position $i_p{+}1$, or $a_{p+1}^{\delta} a_{p+2}$ is a subword of v starting at position $v_p{+}1$. This will accurately calculate each set $g(w, u, v, l)$ by Lemma 1. If we do this for all (i_p, j_p), the resulting list must be of size less than or equal to k after each step l by the assumption and therefore we can store the numbers in logspace. Moreover, $(|u|, |v|)$ does not appear in the final list if and only if $w \notin u \sqcup\!\sqcup v$ by Lemma 2. □

These three lemmas allow to prove the main result of this section.

Theorem 1. *Let M be a DFA, and let $u, v \in \Sigma^+$ with fixed-length skeletons. Then, there is a polynomial time algorithm to decide whether or not $L(M) \subseteq u \sqcup\!\sqcup v$, and to decide whether $L(M) = u \sqcup\!\sqcup v$.*

Proof. Let $u = b_1^{\beta_1} b_2^{\beta_2} \cdots b_p^{\beta_p}, b_j \neq b_{j+1}, 1 \leq j < p, \beta_i > 0, b_i \in \Sigma, 1 \leq i \leq p$, and let $v = c_1^{\gamma_1} c_2^{\gamma_2} \cdots c_q^{\gamma_q}, c_j \neq c_{j+1}, 1 \leq j < q, \gamma_i > 0, c_i \in \Sigma, 1 \leq i \leq q$, where p and q are fixed. Let k be the maximum of p and q. If, for all $w \in L(M)$ and every l from 1 to the length of the skeleton of w, it is true that $|g(w, u, v, l)|$ is less than or equal to some constant, then the result follows from Lemma 3.

Let $w = a_1^{\alpha_1} a_2^{\alpha_2} \cdots a_n^{\alpha_n}, a_j \neq a_{j+1}, 1 \leq j < n, \alpha_i > 0, a_i \in \Sigma, 1 \leq i \leq n$. Then we can build a directed acyclic graph $G = (V, E)$ such that $V =$

$\bigcup_{0 \leq l \leq n} g(w, u, v, l)$ and

$$E = \{(x, y) \mid x = (i, j) \in g(w, u, v, l) \text{ for some } l, 0 \leq l < n, y = (h, m) \in g(w, u, v, l+1) \text{ s. t. } a_{l+1}^{\alpha_{l+1}} \in u[i+1, h] \sqcup v[j+1, m] \text{ and either } u(h+1) = a_{l+2} \text{ or } v(m+1) = a_{l+2} \text{ or } (h = |u| \text{ and } m = |v|)\}.$$

It is clear that $w \in u \sqcup v$ if and only if there is a path from $(0, 0)$ to $(|u|, |v|)$ in G from Lemma 1 and Lemma 2. Also, every $x \in V$ is reachable from $(0, 0)$ via at least one path.

Let l be such that $0 \leq l < n$, and let $(i, j) \in g(w, u, v, l)$. Consider $a_{l+1}^{\alpha_{l+1}}$. Assume $a_{l+1} \neq a(i+1)$ and $a_{l+1} \neq v(j+1)$. Then there is no outgoing edge from (i, j). Assume that $a_{l+1} = u(i+1)$ but $a_{l+1} \neq v(j+1)$. If $u[i+1, i+\alpha_{l+1}] = a_{l+1}^{\alpha_{l+1}}$ and either $u(i + \alpha_{l+1} + 1) = a_{l+2}$ or $v(j + 1) = a_{l+2}$ or $(i + \alpha_{l+1} = |u|$ and $j = |v|)$, then (i, j) has one outgoing edge. Otherwise (i, j) has no outgoing edges. Similarly if $a_{l+1} \neq u(i + 1)$ and $v_{l+1} = v(j + 1)$. Then the only way of having more than one outgoing edge from (i, j) is if $a_{l+1} = u(i + 1) = v(j + 1)$. But in this situation, from (i, j), there are at most two outgoing edges since all copies of a_{l+1} must be consumed from either u or v by the definition of g.

Let $(i_0, j_0) = (0, 0), (i_1, j_1), \ldots, (i_x, j_x)$ (with $x \leq n$ and potentially $x = n$) be a sequence of vertices such that e_α connects (i_α, j_α) to $(i_{\alpha+1}, j_{\alpha+1})$ for all α, $0 \leq \alpha < x$. Let $(p_0, q_0), \ldots, (p_m, q_m)$ be the subsequence of $(i_0, j_0), (i_1, j_1), \ldots, (i_x, j_x)$ such that there are two outgoing edges in G from $(p_\alpha, q_\alpha), 0 \leq \alpha \leq m$. This list is of size at most $2k$ since each one consumes one section of the skeleton of u or v. Further, it can be shown (omitted for reasons of space) that, if every such path has at most $2k$ branching points, the number of elements in $g(w, u, v, l)$ is at most $2^{2k+1} - 1$ for every l, which is a constant since k is a constant. □

4 General coNP-Completeness

The purpose of this chapter is to show that given a DFA M and words $u, v \in \Sigma^+$, the problem of determining whether or not $L(M) \subseteq u \sqcup v$ is coNP-Complete.

To show in general (for any u, v and M), this problem is not solvable in polynomial time, we need to examine pairs of words u, v whereby the DFA created from the subset construction accepting $u \sqcup v$ is not polynomial in size by Proposition 1. Indeed, we know that such automata exist, as in [4], an infinite subset of $\Sigma^+ \times \Sigma^+$ is demonstrated such that for each (u, v) in the subset, the minimal DFA accepting $u \sqcup v$ requires an exponential number of states. These word pairs are quite similar to those constructed in Theorem 2 below. The existence of minimal DFAs accepting the shuffle of two words that requires an exponential number of states is not enough information on its own though to show that testing $L(M) \subseteq u \sqcup v$ cannot be done in polynomial time, as there could in principle be an algorithm that tests this fact without first constructing the minimal DFA accepting $u \sqcup v$ (as is the case for the converse problem, which can be tested in polynomial time).

We will examine the complement of that problem; that is, the problem of whether there exists some $w \in L(M)$ such that $w \notin u \sqcup v$. We will show that this

problem is NP-Complete. This implies that the problem of determining whether or not $L(M) \subseteq u \sqcup v$ is coNP-Complete, by Proposition 10.1 of [11], which states that the complement of an NP-Complete language is coNP-Complete.

Throughout the proof, we will refer to Example 1 for the purposes of intuition. It is helpful to follow the example together with the proof.

Theorem 2. *Let M be a DFA and let $u, v \in \Sigma^+$ where Σ has at least two letters. The problem of determining whether there exists $w \in L(M)$ such that $w \notin u \sqcup v$ is NP-Complete.*

Proof. It is clear that this problem is in NP since we can construct a nondeterministic Turing Machine that guesses a word in $L(M)$ and then verifies that $w \notin u \sqcup v$ (we can test membership in the naive NFA accepting $u \sqcup v$, and NFA membership testing can be done in polynomial time [8]).

Thus, we need to show that the problem is NP-hard. Let F be an instance of the satisfiability problem with $X = \{x_1, \ldots, x_p\}$ the set of Boolean variables, and $C = \{c_1, \ldots c_q\}$ the set of clauses over X where each clause in C has three literals. This problem is known as 3SAT and it is NP-Complete (Proposition 9.2 of [11]). We will also assume without loss of generality that $q \geq 2$. For a variable x, let x^+ be the literal obtained from the variable x as true (simply the variable x), and x^- be the literal obtained from the variable as false (the negation of x). If d is a truth assignment, then d is a function from X to $\{+, -\}$ (true or false). We extend d to a function on clauses, where $d(c) = +$ if c contains at least one literal that matches the sign of d applied to its variable, and $d(c) = -$ if all literals have differing signs than its variables on d.

We will first provide the construction. Although technical, we will refer throughout the proof to Example 1, located after the proof, for intuition. Next, we construct the two words u, v, and the DFA accepting the language L. The words u and v depend only on the number of variables and clauses. The language L accepts a different string for every possible truth assignment to the variables X. Each such string contains a substring for each variable consecutively, and within each such substring, another sequence of substrings for each clause consecutively.

Let $f(c_i, x_j, \alpha)$ be a function from $C \times X \times \{+, -\}$ such that

$$f(c_i, x_j, \alpha) = \begin{cases} bbbabaaa & \text{if } c_i \text{ does not contain literal } x_j^\alpha, \\ bbbbaaaa & \text{otherwise (if } c_i \text{ does contain literal } x_j^\alpha). \end{cases}$$

Then, let

$$\begin{aligned} u &= (aabb)^{q-1}(aababab(aabb)^{q-2})^p(aabb), \\ v &= (aabb)^{q-1}(aabbaabb(aabb)^{q-2})^p(aabb), \\ g(x_j, \alpha) &= f(c_1, x_j, \alpha) f(c_2, x_j, \alpha) \cdots f(c_q, x_j, \alpha), x_j \in X, \alpha \in \{+, -\}, \\ y(d) &= g(x_1, d(x_1)) \cdots g(x_p, d(x_p)), d \text{ a function from } X \text{ to } \{+, -\}, \\ Y &= \{y(d) \mid d \text{ a function from } X \text{ to } \{+, -\}\}, \\ L &= a(aabb)^{q-1}aaa \cdot Y \cdot bbbb(aabb)^{q-1}. \end{aligned}$$

Below, we will show that F is satisfiable if and only if there exists a word in L that is not in $u \sqcup v$. First, notice that we can build u and v in polynomial time, and they depend only on the number of variables and clauses in F.

We will build a DFA accepting L in polynomial time in several steps. First, we can build a DFA accepting each $f(c_i, x_j, \alpha)$ which only has 9 states since each accepts only one word of length 8. We can also build all $f(c_i, x_j, \alpha)$ in $O(pq)$ time. Then, we can accept each $g(x_j, \alpha)$ in polynomial time. We can then build a DFA accepting $\{g(x_j, +), g(x_j, -)\}$ for every j. Indeed, if x_j is a variable, and l is the length of the longest common prefix of $g(x_j, +)$ and $g(x_j, -)$, then we can build a DFA that reads this common prefix and then switches to one of two states based on whether the next letter is from $g(x_j, +)$ or $g(x_j, -)$. Then, from the two different states, it reads what remains of either $g(x_j, +)$ or $g(x_j, -)$. Upon reading the last symbol of either, the DFA switches to a common final state. Thus, we can build a DFA accepting $\{g(x_j, +), g(x_j, -)\}$ in polynomial time. Next, note that $Y = \{g(x_1, +), g(x_1, -)\} \cdot \{g(x_2, +), g(x_2, -)\} \cdots \{g(x_p, +), g(x_p, -)\}$. Hence, we can also build an automaton accepting Y in polynomial time by concatenating the DFA for $\{g(x_1, +), g(x_1, -)\}$ to that of $\{g(x_2, +), g(x_2, -)\}$, and so on, for all p variables. Then we can transform the DFA accepting Y into one accepting L in polynomial time. In Example 1, we provide an instance of the 3SAT problem and show its DFA (accepting Y) created by this construction in Figure 1. Intuitively, notice that every path through the automaton corresponds to a different truth assignment. For each variable, consecutively, taking the upper path corresponds to setting that variable to be true, and the lower path corresponds to setting that variable to be false.

For a prefix w of a word in L, we let

$$h(w) = \{(i, j) \mid w \in u[i] \sqcup v[j]\}.$$

We will show next that d is a satisfying truth assignment if and only if $h(a(aabb)^{q-1}aaa \cdot y(d)) = \emptyset$.

Each word of Y is composed of $y(d)$ where d is any assignment of the variables. That is, each word is of the form $g(x_1, \alpha_1) \cdots g(x_p, \alpha_p)$ where $\alpha_1, \ldots, \alpha_p$ can be any assignment of each variable to $+$ (true) or $-$ (false). Each $g(x_j, \alpha)$ is a string where we concatenate for each clause *bbbabaaa* when x_j^α is not in the clause (either the variable is not in the clause at all, or only the negation of x_j^α is in the clause), and *bbbbaaaa* when the literal is in the clause.

Every word in L starts with $a(aabb)^{q-1}$, and indeed, for any $q \geq 2$,

$$h(a(aabb)^{q-1}) = \{(4q - 4l, 4l - 3), (4q - 4l - 3, 4l) \mid 1 \leq l < q\},$$

(proof omitted due to space constraints). This part is essentially identical to a claim in Theorem 13 of [4]. Intuitively, in Figure 2, this can be seen visually as the set of points at the bottom diagonal of the "duplication section", where $l = 1$ occurs for the first two points, followed by $l = 2$ for the next two points, etc. Then,

$$h(a(aabb)^{q-1}aaa) = \{(4q - 4l + 2, 4l - 2) \mid 1 \leq l \leq q\}$$

(this is diagonal below the previous diagonal marked on the diagram as β_0), as the point $(4q-4l, 4l-3)$ gives one point two rows down and one column to the right, $(4q-4l+2, 4l-2)$, while the point $(4q-4l-3, 4l)$ gives one point one row down and two columns to the right, $(4q-4l-3+1, 4l+2) = (4q-4(l+1)+2, 4(l+1)-2)$.

This paragraph will first explain the intuition regarding the rest of the proof before the formal proof (again, while referencing Example 1 and its figures). Looking at Figure 2, at the diagonal marked β_0, there is a dot for each clause, spaced evenly apart. Then, a sequence of words will be read that are each either *bbbabaaa* or *bbbbaaaa* between consecutive diagonal lines marked as β_k, for some k. At first, if *bbbabaaa* is read, then the first "clause" can pass the horizontal line marked "prune x_1", as it does in the diagram, and end with a single point four rows down and four columns to the right on the next diagonal. If *bbbbaaaa* is read, then the "clause" gets cut off instead (as the second clause does in the figure when reaching the "prune x_1" line). Then, as some word of $(bbbabaaa + bbbbaaaa)^q$ is read, each clause is being "cut off", one at a time if and only if the literal x_1^- is in the clause (by having the word *bbbbaaaa*). Any clause not at the "prune" line (either before or after the line) leads to an identical point on the next diagonal four rows down and four columns to the right when reading either *bbbabaaa* or *bbbbaaaa*. Thus, it is only the prune line that affects whether the clause continues, and each clause reaches the "'prune x_1" line consecutively as each $f(c_i, x_1, d(x_1))$ is read, for each $i, 1 \leq i \leq q$. Moreover, since M consecutively reads an entire word, either $g(x_1, +) = f(c_1, x_1, +) \cdots f(c_q, x_1, +)$ or $g(x_1, -) = f(c_1, x_1, -) \cdots f(c_q, x_1, -)$, this enforces that x_1 is set to true or false identically for each clause. This process then continues for each variable from $x_2, \ldots, x_p$ using the consecutive prune lines. Should a "clause" continue past all prune lines, that means that all three literals in the clause were the opposite sign as the function d applied to those variables, implying that it corresponds to a non-satisfiable truth assignment. Therefore, if every word in $L(M)$ has at least one path continue past every prune line, then no possible truth assignment is satisfying. Conversely, if a clause does get cut off, that means that one of the variables in the clause has the same value as d. Therefore, if every clause has some variable set as in d (F is satisfiable), then every clause gets cut off by some prune line and $w_d \notin u \sqcup\!\sqcup v$. This is the case in Example 1 and Figure 2.

Formally, it can be shown that (proof omitted due to space constraints)

$$h(a(aabb)^{q-1}aaa \cdot y(d)) = \{(4q-4l+2+4pq, 4l-2+4pq) \mid 1 \leq l \leq q, d(c_l) = -\}.$$

This means that after reading $y(d)$, if all of the clauses are satisfied by d, then $h(a(aabb)^{q-1}aaa \cdot y(d)) = \emptyset$. Therefore, if we add any suffix to the end of $a(aabb)^{q-1}aaa \cdot y(d)$, it cannot be in $u \sqcup\!\sqcup v$. Hence, $a(aabb)^{q-1}aaa \cdot y(d) \cdot bbbb(aabb)^{q-1} \notin u \sqcup\!\sqcup v$.

Conversely, if after reading $y(d)$, at least one of the clauses is not satisfied by d. Then, $h(a(aabb)^{q-1}aaa \cdot y(d)) \neq \emptyset$, and there must exist some $l, 1 \leq l \leq q$ such that $d(c_l) = -$ and $(4q-4l+2+4pq, 4l-2+4pq) \in h(a(aabb)^{q-1}aaa \cdot y(d))$. Notice that $|u| = |v| = 4q + 4pq$, and so for each l between 1 and q, there are

$4l-2$ letters left from u and $2+4(q-l)$ letters left from v. Then what remains of u is $bb(aabb)^{l-1}$ and what remains of v is $bb(aabb)^{q-l}$. But every point in this set can reach point $(|u|,|v|)$ on input $bbbb(aabb)^{q-1}$ since $(l-1)+(q-l)=q-1$.

Hence, there exists a word in L that is not in $u \sqcup\!\sqcup v$ if and only F is satisfiable. □

Example 1. Consider the following instance of 3SAT with clauses $c_1 = (x_1^+ \vee x_2^+ \vee x_3^+), c_2 = (x_1^- \vee x_2^+ \vee x_3^-), c_3 = (x_1^+ \vee x_2^- \vee x_3^-)$. From this, we can construct each $g(x_j, \alpha)$ as follows:

$$g(x_1,+) = \overbrace{bbbbaaaa}^{f(c_1,x_1,+)}\overbrace{bbbabaaa}^{f(c_2,x_1,+)}\overbrace{bbbbaaaa}^{f(c_3,x_1,+)}$$
$$g(x_1,-) = \overbrace{bbbabaaa}^{f(c_1,x_1,-)}\overbrace{bbbbaaaa}^{f(c_2,x_1,-)}\overbrace{bbbabaaa}^{f(c_3,x_1,-)}$$
$$g(x_2,+) = \overbrace{bbbbaaaa}^{f(c_1,x_2,+)}\overbrace{bbbbaaaa}^{f(c_2,x_2,+)}\overbrace{bbbabaaa}^{f(c_3,x_2,+)}$$
$$g(x_2,-) = \overbrace{bbbabaaa}^{f(c_1,x_2,-)}\overbrace{bbbabaaa}^{f(c_2,x_2,-)}\overbrace{bbbbaaaa}^{f(c_3,x_2,-)}$$
$$g(x_3,+) = \overbrace{bbbbaaaa}^{f(c_1,x_3,+)}\overbrace{bbbabaaa}^{f(c_2,x_3,+)}\overbrace{bbbabaaa}^{f(c_3,x_3,+)}$$
$$g(x_3,-) = \overbrace{bbbabaaa}^{f(c_1,x_3,-)}\overbrace{bbbbaaaa}^{f(c_2,x_3,-)}\overbrace{bbbbaaaa}^{f(c_3,x_3,-)}$$

From this, we can construct the set Y. In Figure 1, we draw the automaton accepting the set Y (and therefore, $a(aabb)^{q-1}aaa$ must be prepended and $bbbb(aabb)^{q-1}$ must be appended to transform it into a DFA accepting L).

This instance has a solution $d : x_1 \to -, x_2 \to -, x_3 \to +$. Then consider the word $w_d = a(aabb)^2aaa \cdot y(d) \cdot bbbb(aabb)^2$, where $y(d)$ is equal to

$$\underbrace{\overbrace{bbbabaaa}^{f(c_1,x_1,-)}\overbrace{bbbbaaaa}^{f(c_2,x_1,-)}\overbrace{bbbabaaa}^{f(c_3,x_1,-)}}_{g(x_1,-)}\underbrace{\overbrace{bbbabaaa}^{f(c_1,x_2,-)}\overbrace{bbbabaaa}^{f(c_2,x_2,-)}\overbrace{bbbbaaaa}^{f(c_3,x_2,-)}}_{g(x_2,-)}\underbrace{\overbrace{bbbbaaaa}^{f(c_1,x_3,+)}\overbrace{bbbabaaa}^{f(c_2,x_3,+)}\overbrace{bbbabaaa}^{f(c_3,x_3,+)}}_{g(x_3,+)}.$$

This word is in $L(M)$ as seen in Figure 1 by taking the lower path, then the next lower path, then the upper path. But this word is not in $u \sqcup\!\sqcup v$ as demonstrated in Figure 2. The dots are placed at indices (i,j) where each prefix of w_d is in $u[i] \sqcup\!\sqcup v[j]$. The additional annotation in the diagram is referred to in the proof of Theorem 2.

If instead we tried the assignment $d' : x_1 \to +, x_2 \to -, x_3 \to +$ (which is not a satisfying truth assignment), then first "clause 1" gets cut off by the x_1 prune line since $x_1^+ \in c_1$, then "clause 2" can go through the line since $x_1^+ \notin c_2$, then "clause 3" gets cut off since $x_1^+ \in c_3$. Then "clause 1" is already cut off and doesn't reach the x_2 prune line, then "clause 2" can continue as $x_2^- \notin c_2$.

Then when "clause 2" reaches the x_3 prune line, it can continue since $x_3^+ \notin c_2$. So, at least one clause passes all prune lines and reading the remaining portion of w_d gives a word in $u \sqcup\!\sqcup v$, and thus d' is not satisfiable. But if there is at least one word in L that is not in $u \sqcup\!\sqcup v$, then this corresponds to a satisfying truth assignment. Hence, at least one word in L is not in $u \sqcup\!\sqcup v$ if and only if there is some satisfying truth assignment.

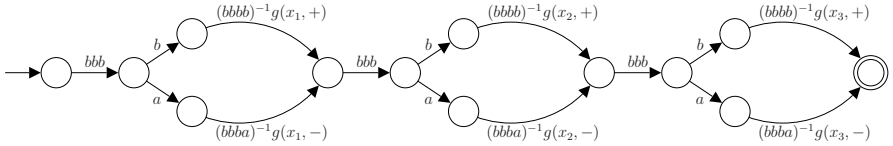

Fig. 1. The DFA accepting Y obtained from the instance of 3SAT from Example 1. We use a word on a transition as a compressed notation to represent a sequence of non-branching transitions.

As mentioned earlier, because testing $L(M) \not\subseteq u \sqcup\!\sqcup v$ is an NP-Complete problem, this implies that testing whether $L(M) \subseteq u \sqcup\!\sqcup v$ is a coNP-Complete problem.

Corollary 1. *Let M be a DFA and let $u, v \in \Sigma^+$, where Σ has at least two letters. The problem of determining whether $L(M) \subseteq u \sqcup\!\sqcup v$ is coNP-Complete.*

Despite the now known complexity of both deciding whether $L(M) \subseteq u \sqcup\!\sqcup v$ and $u \sqcup\!\sqcup v \subseteq L(M)$, the exact complexity of deciding whether or not $L(M) = u \sqcup\!\sqcup v$ is not immediate. In the proof of Theorem 2, had we started with u, v, M under the assumption that $u \sqcup\!\sqcup v \subseteq L(M)$, and shown coNP-Completeness as to whether $L(M) \subseteq u \sqcup\!\sqcup v$, then that would imply that testing whether $L(M) = u \sqcup\!\sqcup v$ would also be coNP-Complete. However, in the proof of Theorem 2, there are many words in $u \sqcup\!\sqcup v$ that are not in $L(M)$ and it is not clear how one could alter M to solve the problem while still creating it in polynomial time. Hence, the problem of calculating the exact complexity of testing whether $L(M) = u \sqcup\!\sqcup v$ remains an open problem.

References

1. Berstel, J., Boasson, L.: Shuffle factorization is unique. Theoretical Computer Science 273, 47–67 (2002)
2. Biegler, F.: Decomposition and Descriptional Complexity of Shuffle on Words and Finite Languages. Ph.D. thesis, University of Western Ontario, London, Canada (2009)
3. Biegler, F., Daley, M., Holzer, M., McQuillan, I.: On the uniqueness of shuffle on words and finite languages. Theoretical Computer Science 410, 3711–3724 (2009)

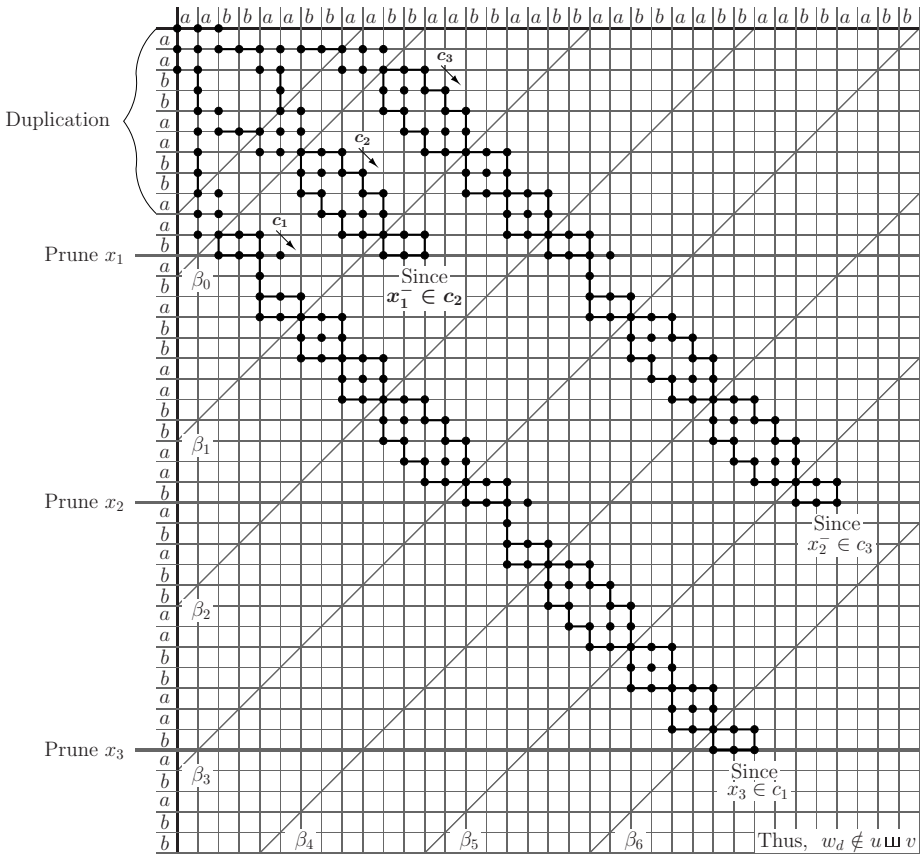

Fig. 2. The word u is labelling the vertical axis and v is labelling the horizontal axis. Considering the word w_d from Example 1, a dot is placed at (i, j) if w_d has a prefix in $u[i]$ shuffled with $v[j]$. Only a portion of the diagram is shown, and u and v continue along the axes, although there are no dots in the rest of the diagram. The lines connecting the dots demonstrates the change of states by reading individual characters.

4. Biegler, F., Daley, M., McQuillan, I.: On the shuffle automaton size for words. Journal of Automata, Languages and Combinatorics 15, 53–70 (2010)
5. Biegler, F., Daley, M., McQuillan, I.: Algorithmic decomposition of shuffle on words. Theoretical Computer Science 454, 38–50 (2012)
6. Câmpeanu, C., Salomaa, K., Vágvölgyi, S.: Shuffle quotient and decompositions. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) Proceedings DLT 5. pp. 186–196. No. 2295 in LNCS, Springer, Wien, Austria (Jul 2001)
7. Ginsburg, S., Spanier, E.: Mappings of languages by two-tape devices. Journal of the ACM 12(3), 423–434 (1965)
8. Holub, J., Melichar, B.: Implementation of nondeterministic finite automata for approximate pattern matching. In: Champarnaud, J.M., Maurel, D., Ziadi, D. (eds.) WIA '98, Lecture Notes in Computer Science, vol. 1660, pp. 92–99. Springer-Verlag (1999)
9. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, MA (1979)
10. Jędrzejowicz, J., Szepietowski, A.: Shuffle languages are in **P**. Theoretical Computer Science 250, 31–53 (2001)
11. Papadimitriou, C.M.: Computational complexity. Addison-Wesley, Reading, Massachusetts (1994)
12. Warmuth, M., Haussler, D.: On the complexity of iterated shuffle. Journal of Computer and System Sciences 28(3), 345–358 (1984)
13. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 1, pp. 41–110. Springer, Berlin Heidelberg (1997)