10-5-2012

Algorithmic Decomposition of Shuffle on Words

Franziska Biegler
Department of Software Engineering and Theoretical Computer Science, Berlin Institute of Technology, Germany

Mark Daley
Departments of Computer Science and Biology, University of Western Ontario, London, ON N6A 5B7, Canada

Ian McQuillan
Department of Computer Science, University of Saskatchewan, Saskatoon, SK S7N 5A9, Canada

Theoretical Computer Science 454 (2012) 38–50

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

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

Algorithmic decomposition of shuffle on words

Franziska Biegler [a], Mark Daley [b,*], Ian McQuillan [c]

[a] Department of Software Engineering and Theoretical Computer Science, Berlin Institute of Technology, Germany
[b] Departments of Computer Science and Biology, University of Western Ontario, London, ON N6A 5B7, Canada
[c] Department of Computer Science, University of Saskatchewan, Saskatoon, SK S7N 5A9, Canada

ARTICLE INFO

Keywords:
Shuffle decomposition
Finite languages
Finite automata
Words

ABSTRACT

We investigate shuffle-decomposability into two words. We give an algorithm which takes as input a DFA M (under certain conditions) and determines the unique candidate decomposition into words u and v such that $L(M) = u \sqcup\!\sqcup v$ if M is shuffle decomposable, in time $\mathcal{O}(|u| + |v|)$. Even though this algorithm does not determine whether or not the DFA is shuffle decomposable, the sublinear time complexity of only determining the two words under the assumption of decomposability is surprising given the complexity of shuffle, and demonstrates an interesting property of the operation.

We also show that for given words u and v and a DFA M we can determine whether $u \sqcup\!\sqcup v \subseteq L(M)$ in polynomial time.

1. Introduction

The shuffle decomposition problem has been a long standing open problem in formal language theory. The problem is defined as follows: Given a regular language L, do there exist two non-trivial (neither is the singleton language consisting of the empty word) languages L_1 and L_2, such that $L = L_1 \sqcup\!\sqcup L_2$, where $\sqcup\!\sqcup$ denotes the shuffle operation on languages? Although the decidability status of this problem remains open, it is decidable if the given language is a commutative regular language or a locally testable language [1], or if there is a decomposition into a regular language and certain classes of finite languages [2].

Lately, attention has been paid to gaining a deeper understanding of a special case: the shuffle of individual words. In [3], it was proven that shuffle decomposition on words is unique as long as there are two letters used within the words. In the case of binary shuffle (which we will restrict ourselves to in this paper), this can be formally stated as $u \sqcup\!\sqcup v = u' \sqcup\!\sqcup v'$ implies $\{u, v\} = \{u', v'\}$ (where there are two combined letters used). The proof in [3] also gives rise to an algorithm that can be used to compute a candidate shuffle root for a given set of words, but the complexity of this algorithm is not further discussed. In [4], the result from [3] was extended to show that if two words u and v both contain at least two letters, then the shuffle decomposition is the unique decomposition over arbitrary sets and not just words. That is, $u \sqcup\!\sqcup v = L_1 \sqcup\!\sqcup L_2$ implies $\{\{u\}, \{v\}\} = \{L_1, L_2\}$. Lastly, the authors have recently shown [5] that the size of minimal deterministic finite shuffle automata for two words can grow exponentially, $\Omega(\sqrt[8]{2}^n) \approx \Omega(1.09^n)$, as a function of words of length n. This extended the result in [6] that the size of the shuffle automaton for two regular languages accepted by m- and n-state automata is 2^{mn} in the worst case (using infinite languages).

In Section 3 of this paper, we present an efficient algorithm which takes as input a trim, acyclic non-unary DFA. If the accepted language satisfies the assumption to be shuffle decomposable into two words, then we are able to determine the two words u and v in time $\mathcal{O}(|u| + |v|)$. This efficiency is perhaps surprising since shuffle decomposition seems to be an

* Corresponding author. Tel.: +1 519 661 3566; fax: +1 519 661 3515.
E-mail addresses: franziska.biegler@tu-berlin.de (F. Biegler), daley@csd.uwo.ca (M. Daley), mcquillan@cs.usask.ca (I. McQuillan).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.04.001

extremely difficult problem and given that there can be an exponential number of states in the automaton relative to the lengths. Thus, only a small number of states of the DFA need to be examined to determine u and v. We present an example of an application of this algorithm on a specific automaton in Section 4. Our algorithm only computes a candidate solution or rejects a subset of automata that do not accept the shuffle of two words for obvious reasons. Thus, the complexity of deciding whether the language accepted by a given DFA is shuffle decomposable into two words is still open, however our algorithm is a key step towards a solution. It also demonstrates an interesting property, as the two strings are encoded in a relatively small part of a shuffle decomposable DFA. In Section 5, we furthermore show that it is decidable in polynomial time, for an acyclic DFA M and two words u, v, whether $u \sqcup\!\sqcup v \subseteq L(M)$.

2. Definitions

We require some mathematical preliminaries and custom definitions to discuss the results of this paper. Let $\mathbb{N}$ be the set of positive integers. An alphabet Σ is a finite, non-empty set of symbols. 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^+.

Let Σ be an alphabet and let $u, v \in \Sigma^*$. If $u = a_1^{\alpha_1} a_2^{\alpha_2} \cdots a_n^{\alpha_n}$ where we have $a_1, \ldots, a_n \in \Sigma$, $\alpha_1, \ldots, \alpha_n \in \mathbb{N}$ and $a_i \neq a_{i+1}$, for $1 \leq i < n$, then the *skeleton* of u is defined as $\chi(u) = a_1 a_2 \cdots a_n$. The different occurrences of the same letter a in the skeleton of u are called the *a-sections* of u. Let $u, v \in \Sigma^*$. The *shuffle* of u and v is defined as $u \sqcup\!\sqcup v = \{u_1 v_1 \cdots u_n v_n \mid u = u_1 \cdots u_n, v = v_1 \cdots v_n, u_i \in \Sigma^*, v_i \in \Sigma^*, 1 \leq i \leq n\}$. We say u is a *prefix* of v, written $u \leq_{\mathrm{p}} v$, if $v = ux$, for some $x \in \Sigma^*$. Also, $(u)^{-1}v = w$ if $v = uw$ and is undefined otherwise.

A deterministic finite automaton (DFA) $M = (Q, \Sigma, q_0, F, \delta)$ is *accessible* if, for each state $q \in Q$, there exists a word $u \in \Sigma^*$ such that $\delta(q_0, u) = q$. M is *co-accessible* if, for each state $q \in Q$, there exists a word $u \in \Sigma^*$ such that $\delta(q, u) \in F$. M is *trim* if it is both accessible and co-accessible. Each DFA can be converted into an equivalent trim DFA. For each $q \in Q$, we define $L_M(q) = L(M')$, where $M' = (Q, \Sigma, q_0, \{q\}, \delta)$. We say M is *acyclic* if $q \notin \delta(q, w)$ for every $q \in Q$ and $w \in \Sigma^+$. The *depth* of an acyclic DFA is the longest path from a start state to a final state. We say that a language L is *shuffle decomposable* if $L = u \sqcup\!\sqcup v$ for some $u, v \in \Sigma^+$, and we say that an automaton M is shuffle decomposable if $L(M)$ is shuffle decomposable. See [7] for details on finite automata.

3. Algorithms for shuffle decomposition

The purpose of this section is to provide an algorithm which takes as input a trim, acyclic non-unary DFA M and determines strings u, v such that M being shuffle decomposable implies $L(M) = u \sqcup\!\sqcup v$ is the unique decomposition. For some automata, the algorithm can determine that M is not shuffle decomposable. However, for some other automata, the algorithm will still output two strings even though M is not decomposable. We know from [3] that there exists at most one solution over words when at least two letters are used in the words (this is not true when both words are over the same one letter alphabet). This problem is obviously solvable by an exponential algorithm. However, we provide an algorithm which operates in time $\mathcal{O}(|u| + |v|)$. This is usually less than the number of states in the automaton. It could also be classified as $\mathcal{O}(|Q|)$, where Q is the state set of the input DFA.

We are only concerned with non-unary DFAs, because decomposition into unary DFAs is not unique, and also being decomposable implies $L(M) = \{a^n\}$, for some n, which is testable in linear time, $\mathcal{O}(n)$.

Some notation for this section is now defined. Let $M = (Q, \Sigma, q_0, F, \delta)$ be a DFA, $q \in Q$ and $a \in \Sigma$. We define $\delta(q) = \{a \in \Sigma \mid \delta(q, a) \neq \emptyset\}$,

$$\delta^{max}(q, \lambda) = 0,$$
$$\delta^{max}(q, a) = n, \text{ where } \delta(q, a^n) \neq \emptyset, \delta(q, a^{n+1}) = \emptyset,$$
$$\delta^{min}(q, a) = n, \text{ where } n \text{ is minimal such that either}$$
$$\delta(q, a^n b) \neq \emptyset \text{ for some } b \neq a, \text{ or } \delta(q, a^n) \in F.$$

For the rest of this section, let $M = (Q, \Sigma, q_0, F, \delta)$ be a trim acyclic, non-unary DFA. Since M is not unary, if $L(M)$ is shuffle decomposable, then there exists A unique u, v such that $L(M) = u \sqcup\!\sqcup v$ (from [3]). Then, for the rest of this section, we assume that M is shuffle decomposable, and we will effectively determine u and v. Also, if $L(M)$ is shuffle decomposable then some word, must contain at least two letters since $a^n \in u \sqcup\!\sqcup v$ implies $L(M)$ only contains one letter, and thus M must be unary since it is trim. Hence, every word in $L(M)$ contains at least two letters.

We need another definition which will be used in the proof of Lemma 3. For $w \in \Sigma^*$ and $q \in Q$, we let

$$w^{(q)} = \begin{cases} w & \text{if } w = \lambda \text{ or } w = w'a, a \in \Sigma, a \notin \delta(q), \\ w' & \text{where } w = w'a^i, i > 0 \text{ maximal, if } \delta(q, a^i) \text{ defined,} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Essentially $w^{(q)}$ allows to remove the last section of a's from w depending on the current state. This is used in order to determine which of the next letters to be read in M comes from u and which comes from v.

We say that $x_1 \cdots x_k, x_i \in \Sigma$ has a unique (u, v)-decomposition if $L(M) = u \sqcup v$ implies that there exists unique $\bar{u} \leq_p u$, $\bar{v} \leq_p v$ such that $x_1 \cdots x_k \in \bar{u} \sqcup \bar{v}$. In such a case, we say that $(\bar{u}, \bar{v})$ is the unique (u, v)-decomposition of $x_1 \cdots x_k$. We say that it is the unique decomposition if (u, v) is obvious from the context.

Lemma 1. *If $L(M) = u \sqcup v$, $L(M)$ has at least two letters, $x_1 \cdots x_k, x_i \in \Sigma$ has a unique (u, v)-decomposition and $\delta(q_0, x_1 \cdots x_k) = q$, then $|\delta(q)| \leq 2$.*

Proof. Let $(\bar{u}, \bar{v})$ be the unique (u, v)-decomposition. Assume that there exists $c, d, e \in \Sigma$, pairwise distinct, such that $\delta(q, c)$, $\delta(q, d)$ and $\delta(q, e)$ are all defined. But since $(\bar{u}, \bar{v})$ is unique, each of c, d, e must appear at position $|\bar{u}| + 1$ of u or $|\bar{v}| + 1$ of v, a contradiction. □

The entire algorithm consists of four functions together with a main procedure. We first give two simple standalone functions before describing the full algorithm. The first function is called **maxShuf**. Essentially, it creates a word in $z \in s \sqcup t$ for $s, t \in \Sigma^*$ by looping through each section in the skeleton of s, and if the letter in that section is the same as the current letter in t, then it merges both sections.

maxShuf(s, t)
1 let $s = a_1^{\alpha_1} \cdots a_n^{\alpha_n}$; $t = b_1^{\beta_1} \cdots b_m^{\beta_m}$;
2 $z := \lambda$; $k := 1$; $l := 1$;
3 while ($k \leq n$ or $l \leq m$)
4 if ($k \leq n, l \leq m$ and $a_k = b_l$)
5 $z = z \cdot a_k^{\alpha_k} b_l^{\beta_l}$; $k := k + 1$; $l := l + 1$;
6 else if ($k \leq n$) $z = z \cdot a_k^{\alpha_k}$; $k := k + 1$;
7 else $z := z \cdot b_l^{\beta_l}$; $l := l + 1$;
8 return z;

The following small example should help to illustrate **maxShuf**, and it also shows that **maxShuf** is not commutative.

Example 2. Let $s = a^4b^3a^2c^2$ and $t = b^4ac^5$. Then **maxShuf**$(s, t) = a^4b^7a^3c^7$ and **maxShuf**$(t, s) = b^4a^5c^5b^3a^2c^2$.

The second function, **findUnique**(q), for some state q, is used only in situations where M being shuffle decomposable implies that there is only one sequence of states from q to a final state (we will show that this holds when we use the function), and it returns the unique string z such that $\delta(q, z) \in F$. If there is no unique sequence of states from q to a final state, then M cannot be shuffle decomposable and **findUnique** prints "Not decomposable" and halts the algorithm.

findUnique(q)
1 let $q' := q$; $z := \lambda$;
2 while ($q' \notin F$)
3 if ($|\delta(q')| \geq 2$ or $|\delta(q')| = 0$)
4 print ("Not decomposable");
5 else let $a \in \delta(q')$; $z := z \cdot a$; $q' := \delta(q', a)$;
6 return z;

We are now ready to describe the full algorithm, consisting of a main procedure and two additional functions. Throughout the algorithm, we use variables $\bar{u}$, $\bar{v}$, w, q, f and g. The variables $\bar{u}$, $\bar{v}$ represent prefixes of u and v respectively that we have determined so far. The variable w represents a prefix of some word in $u \sqcup v$ such that $w \in \bar{u} \sqcup \bar{v}$. The state q will always be $\delta(q_0, w)$ as we will always concatenate letters to w simultaneously as we change state q by using transitions via the same sequence. The variables f and g are in $\Sigma \cup \{\lambda\}$. At most one of these at any given time will be in Σ and the other will be λ. Throughout the proof, if we informally say some string y must be read from u (or v), this means that $(\bar{u}y, \bar{v})$ is the unique decomposition of wy.

For the proof of correctness, we continue to assume that $L(M) = u \sqcup v$ (if this is false, then either it will output "Not decomposable", or two words u and v, but $L(M)$ is not shuffle decomposable). At the start of the procedure, and each time we evaluate the loop condition of the procedure (line 5 below), we verify that the following three conditions are true, which we call the *inductive conditions*, and which we refer to throughout the proof of correctness:

1. $f \in \Sigma$ (and $g = \lambda$) implies that $(\bar{u}f, \bar{v})$ is the unique (u, v)-decomposition of wf and $(\bar{u}^{(q)}, \bar{v})$ is the unique (u, v)-decomposition of $w^{(q)}$ and $|\delta(q)| = 2$,
2. $g \in \Sigma$ (and $f = \lambda$) implies that $(\bar{u}, \bar{v}g)$ is the unique (u, v)-decomposition of wg and $(\bar{u}, \bar{v}^{(q)})$ is the unique (u, v)-decomposition of $w^{(q)}$ and $|\delta(q)| = 2$,
3. $f = g = \lambda$ implies that $(\bar{u}, \bar{v})$ is the unique (u, v)-decomposition of w and either $q \in F$ or $q = q_0$.

Then, each time through the loop as we concatenate letters to $\bar{u}$, $\bar{v}$ and w, either $(\bar{u}, \bar{v})$, $(\bar{u}f, \bar{v})$ or $(\bar{u}, \bar{v}g)$ is a unique decomposition. We use the variable x in the procedure to group together $f, g, q, w, \bar{u}, \bar{v}$, which is passed and returned from the functions and as we update any of the variables discussed in the previous paragraph, it is assumed to change x also.

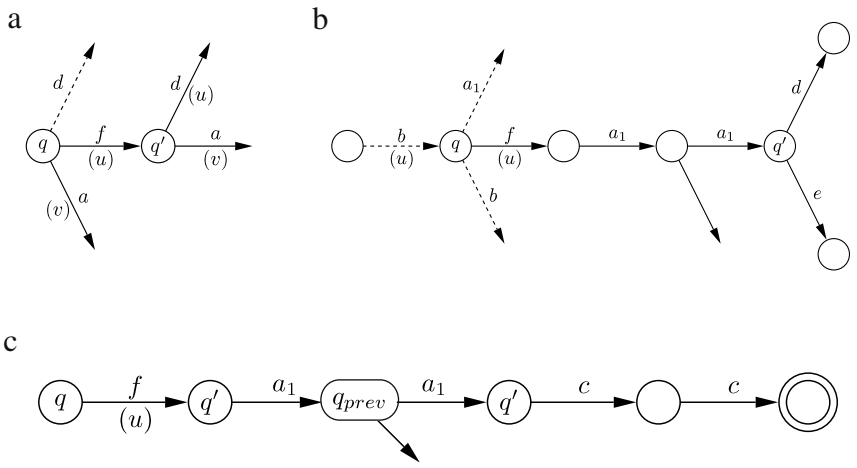

Fig. 1. Pictured above are schematic drawings for (a) Case 1, (b) Case 2 and (c) Case 3 of **firstDiff**. Having u or v in parentheses indicates that that letter must come from u or v. The symbols used in the diagram are described in **firstDiff** and in Lemma 3. Also, a dashed line represents an optional transition.

We will now describe the main algorithm, which uses two additional functions to be described below:

Procedure.

1 $x := (f, g, q, w, \bar{u}, \bar{v})$ where
2 $f := \lambda;\ g := \lambda;\ q := q_0;\ w := \lambda;\ \bar{u} := \lambda;\ \bar{v} := \lambda;$
3 if $(|\delta(q)| = 1)$ $x :=$ **firstDiff**(x);
4 else set f to be one of the letters in $\delta(q)$;
5 while ($|\delta(q)| = 2$ and either $f \neq \lambda$ or $g \neq \lambda$)
6 $x :=$ **firstDiff**(x);
7 if $(|\delta(q)| = 0)$ then print ("Candidate Solution: $\{\bar{u}, \bar{v}\}$");
8 else print ("Not a shuffle automaton"); □

As indicated above, at the beginning of the procedure, and each time we evaluate the loop condition (line 5), at most one of $f \neq \lambda$ or $g \neq \lambda$. Moreover, $f = g = \lambda$ will occur if and only if either $\bar{u} = \bar{v} = w = \lambda$ (at the beginning of the procedure) or $q \in F$ (at the end of the procedure). Also, we already know $q \in F$ if and only if $|\delta(q)| = 0$ since M is co-accessible and all words are of the same length in $u \sqcup\!\sqcup v$, in shuffle decomposable DFAs. Otherwise, exactly one of $f \in \Sigma$ or $g \in \Sigma$. We will show that the inductive conditions are always true by induction.

For the base case, we see that at the beginning of the procedure, $w = \lambda, f = \lambda$ and $g = \lambda$ and indeed, $(\bar{u}, \bar{v}) = (\lambda, \lambda)$ is the unique decomposition of $w = \lambda$ here. Thus assume by way of induction, that after the kth visit to either the start of the procedure or the loop condition on line 5, the inductive conditions are true.

This procedure makes use of the function **firstDiff** which we will describe below. At the start of the main procedure above, if $|\delta(q)| = 1$, then we use **firstDiff**. If this is not the case, then $|\delta(q)| = 2$ and, furthermore, inside the loop of the main procedure, either $f \neq \lambda$ or $g \neq \lambda$.

We continue by describing the **firstDiff** function, which takes in x as parameter, with a proof below (Lemma 3) that if the inductive conditions hold upon its commencement, then it also holds upon its termination. Although lengthy, it is essentially divided into four major cases marked within. We will intuitively describe and prove correctness of the **firstDiff** function in Lemma 3 and we will prove correctness of the first three cases with separate claims within the proof of Lemma 3. In line 8, it calls on the final function **maxEqual** and we also have a claim proving its functionality. In addition, Case 4 and **maxEqual** are used in the example of Section 4 and schematic drawings are provided in Fig. 1 for Cases 1, 2, 3.

firstDiff$(x = (f, g, q, w, \bar{u}, \bar{v}))$

1 if $(g = \lambda)$ // either $f \in \Sigma$ and $|\delta(q)| = 2$ or $f = \lambda$ and $|\delta(q)| = 1$,
2 $\gamma := \delta^{max}(q, f);\ q_{prev} := \delta(q, f^{\gamma-1});\ q' := \delta(q, f^{\gamma});\ s = \lambda;\ t = \lambda;\ z = \lambda;$

3 if $(|\delta(q')| = 2)$ //**Case 1**
4 if $(\delta(q') \setminus \delta(q) = \emptyset)$ let $d \in \delta(q')$ where d is not the last letter of $\bar{u}$;
5 else let $d \in \delta(q') \setminus \delta(q)$;
6 $\bar{u} := \bar{u}f^{\gamma};\ w := wf^{\gamma};\ q := \delta(q, f^{\gamma});$ return $(d, \lambda, q, w, \bar{u}, \bar{v})$;

```
7   else if |δ(q')| = 1
8     (s, t, z, q_prev, q', c, ϵ, β) := maxEqual(q_prev, q');
9     if (|δ(q')| = 2)                    //Case 2
10      let {d, e} := δ(q');
11      if δ(q, maxShuf(se, f^γ t)) is defined
12        ū := ūf^γ s;  v̄ := v̄t;  w := wf^γ z;  q := δ(q, f^γ z);
13        return (d, λ, q, w, ū, v̄);
14      else ū := ūf^γ t;  v̄ := v̄s;  w := wf^γ z;  q := δ(q, f^γ z);
15        return (λ, d, q, w, ū, v̄);
16    else if (ϵ = β)                     //Case 3
17      α := δ^max(q_prev, c);
18      if (α = β) s := s · findUnique(q');  z := z · findUnique(q');
19      else s := sc^max{α,β−α};  t := tc^min{α,β−α};  z := zc^β;
20      if (δ(q, maxShuf(s, f^γ t)) is defined
21        ū := ūf^γ t;  v̄ := v̄s;  w := wf^γ z;  q := δ(q, f^γ z);
22        return (λ, λ, q, w, ū, v̄);
23      else ū := ūf^γ s;  v̄ := v̄t;  w := wf^γ z;  q := δ(q, f^γ z);
24        return (λ, λ, q, w, ū, v̄);
25    else if (ϵ ≠ β)                     //Case 4
26      if (2ϵ > β)
27        q' := δ(q', c^β);  r := findUnique(q');
28        if (δ(q, maxShuf(sc^ϵ, f^γ t)) is defined
29          ū := ūf^γ sc^{β−ϵ};  v̄ := v̄tc^ϵ r;
30        else ū := ūf^γ c^ϵ r;  v̄ := v̄tsc^{β−ϵ};
31        w := wf^γ zc^β r;  q := δ(q, f^γ zc^β r);  return (λ, λ, q, w, ū, v̄);
32      else let d ∈ δ(δ(q', c^ϵ)) \ {c}
33        if (δ(q, maxShuf(sc^{ϵ+1}, f^γ t)) is defined
34          ū := ūf^γ sc^ϵ;  v̄ := v̄t;  w := wf^γ zc^ϵ;  q := δ(q, f^γ zc^ϵ);
35          return (d, λ, q, w, ū, v̄);
36        else ū := ūf^γ t;  v̄ := v̄sc^ϵ;  w := wf^γ zc^ϵ;  q := δ(q, f^γ zc^ϵ);
37          return (λ, d, q, w, ū, v̄);
38    else (|δ(q')| ∉ {1, 2}) print("Not decomposable"); //end if line 3
39  else if (f = λ) (similar to the case where g = λ with f, g and ū, v̄ switched)
40  else print("Not decomposable");   □
```

We will briefly describe **firstDiff** intuitively, with further additional details in Lemma 3. Each time **firstDiff** executes, it concatenates some letters to at least one of $\bar{u}$ or $\bar{v}$ until u and v have been completely determined. We use the variable q' as a temporary variable which holds a state reachable from q. We will only consider the case where $g = \lambda$, as the case where $g \neq \lambda$ is symmetric. In line 2, the maximal number of f's from state q is read to obtain state q'. Then, if there are two transitions which can be followed, case 1 applies. Otherwise there must only be one transition and either case 2, 3 or 4 will apply. Each of these cases reads additional letters from state q' such that when we concatenate them with w, we get a unique (u, v)-decomposition, which is necessary to prove the inductive definition on return to the main procedure. Other temporary variables will be explained in the proof.

We now give the last function, **maxEqual**, which is called from line 8 of **firstDiff**. It determines the longest sequence $s = t = a_1^{\alpha_1} a_2^{\alpha_2} \cdots a_k^{\alpha_k}$ such that both u and v continue with $s = t$. This is done by repetitively checking, for the current state q' if the maximal number of a_i's is twice as much as the minimal number of a_i's that can be read starting from q'. Intuitively, the purpose of this function is that when we read both s and t, which are the same, we will know that $(\bar{u}f^\gamma s, \bar{v}t)$ is a unique decomposition. But in order to determine whether the letters that follow those are read from either u or read from v, we need to read an alternate path of characters inside either case 2, 3 or 4 of **firstDiff** before we can conclude which must be read from u and which from v. Indeed, we need to use state q (the original state at the beginning of **firstDiff** before reading s and t to determine whether the additional letters are read from u or v. A formal proof of the behavior of **maxEqual** as well as an explanation of additional temporary variables are found inside the proof of Lemma 3. Also, **maxEqual** is used multiple times in the example of Section 4.

```
maxEqual(q_prev, q')
1  s := λ;  t := λ;
2  while (|δ(q')| = 1)
```

3 $c := \delta(q'); \beta := \delta^{max}(q', c); \epsilon := \delta^{min}(q', c);$
4 if $(2 \cdot \epsilon = \beta)$
5 $s := sc^{\epsilon}$; $t := tc^{\epsilon}$; $z := zc^{\beta}$;
6 $q_{prev} := \delta(q', c^{\beta-1})$; $q' := \delta(q', c^{\beta})$;
7 else return $(s, t, z, q_{prev}, q', c, \epsilon, \beta)$;
8 return $(s, t, z, q_{prev}, q', c, \epsilon, \beta)$; □

The following lemma shows that whenever the inductive condition holds on some x when **firstDiff** starts, then the inductive condition also holds on it when returned by **firstDiff**. Furthermore, at least some letters are read from M on every execution of **firstDiff** ensuring termination of the main procedure. From Lemma 3, it then follows that the entire procedure works correctly whenever M is shuffle decomposable into two words. The proof of Lemma 3 is divided into five claims which correspond to the different cases of the algorithm and the **maxEqual** function.

Lemma 3. *If $x = (f, g, q, w, \bar{u}, \bar{v})$, the inductive conditions hold on x and exactly one of $(f \in \Sigma, g = \lambda)$, or $(g \in \Sigma, f = \lambda)$, or $(f = g = \bar{u} = \bar{v} = w = \lambda, q = q_0)$ is true before executing* **firstDiff**(x)*, which returns $x_1 = (f_1, g_1, q_1, w_1, \bar{u}_1, \bar{v}_1)$ then the inductive conditions hold on x_1, $|w_1| > |w|$, and exactly one of $(f_1 \in \Sigma, g_1 = \lambda)$, or $(f_1 = \lambda, g_1 \in \Sigma)$, or $(f_1 = g_1 = \lambda, q_1 \in F)$ is true.*

Proof. We will only consider the case where $g = \lambda$ since the case where $f = \lambda$ and $g \in \Sigma$ (line 39) is symmetric by switching all f's for g's and all $\bar{u}$'s for $\bar{v}$'s and vice versa. Thus, at the beginning of **firstDiff**, $(\bar{u}f, \bar{v})$ is the unique decomposition of wf.

If $f = \lambda$, then on line 2, $\gamma = 0$ (as $\delta^{max}(q, \lambda) = 0$ always), and $q_{prev} = q' = q = q_0$, $|\delta(q)| = 1$ since the main procedure only calls **firstDiff** with $f = g = \lambda$ when this is true, and hence $|\delta(q')| = 1$.

If instead $f \in \Sigma$, then $(\bar{u}^{(q)}, \bar{v})$ is the unique decomposition of $w^{(q)}$ by the assumption, $|\delta(q)| = 2$ and $\delta^{max}(q, f) = \gamma > 0$. Then if we read f^{γ} from state q, it must be read from u because $(\bar{u}f, \bar{v})$ is the unique decomposition of wf and thus the next letter of v cannot be f. Thus wf^{γ} is uniquely decomposable into $(\bar{u}f^{\gamma}, \bar{v})$. Then, we read f^{γ} to obtain state q' on line 2.

We show correctness of Case 1:

Claim 1. If line 3 is true (Case 1), then $f \in \Sigma$ and either there exist $d \in \delta(q') \setminus \delta(q)$, or $\delta(q') \subseteq \delta(q)$ and there exist $d \in \delta(q')$, with d not the last letter of $\bar{u}$. In either case, $(\bar{u}f^{\gamma}d, \bar{v})$ is the unique decomposition of $wf^{\gamma}d$, $(\bar{u}f^{\gamma\ (q')}, \bar{v}) = (\bar{u}f^{\gamma}, \bar{v})$ is the unique decomposition of wf^{γ} and $|\delta(q')| = 2$.

Proof of Claim. Assume $|\delta(q')| = 2$ (line 3). If $f = \lambda$ then $|\delta(q')| = 1$ as discussed above, and hence necessarily $f \in \Sigma$. Then there must be some letter $a \in \delta(q')$ with $a \neq f$ where a is also in $\delta(q)$ (this is the next letter of v as f is next in u after $\bar{u}$), and $f \notin \delta(q')$ as we have read the maximal number (line 2). Since $|\delta(q')| = 2$, there is some $d \in \delta(q')$, $d \neq a$, $d \neq f$ (the next letter of u after $\bar{u}f^{\gamma}$). Suppose $d \in \delta(q)$. Then $|\delta(q)| \geq 3$ and $\delta(q') \subseteq \delta(q)$ as $d, a, f \in \delta(q)$. But, $(\bar{u}^{(q)}, \bar{v})$ is the unique decomposition of $w^{(q)}$. If $\bar{u}^{(q)} = \bar{u}$, then either $\bar{u} = \lambda$ or the last letter of $\bar{u}$, b say, is not in $\delta(q)$. Then $(\bar{u}^{(q)}, \bar{v}) = (\bar{u}, \bar{v})$ is the unique decomposition of w, but $|\delta(q)| \geq 3$ contradicts Lemma 1. Otherwise, $\bar{u}^{(q)} \neq \bar{u}$, and $b^l = (\bar{u}^{(q)})^{-1}\bar{u}$, $l > 0$ and $\delta(q, b^l)$ is defined. Also, $b^l = (w^{(q)})^{-1}w$ as $wf \in \bar{u}f \sqcup\!\sqcup \bar{v}$ and $w^{(q)} \in \bar{u}^{(q)} \sqcup\!\sqcup \bar{v}$. If $a \neq b$, then b^l must be read from u from state $\delta(q_0, w^{(q)})$, as $(\bar{u}^{(q)}, \bar{v})$ is a unique decomposition, and because b is the next letter of u and a is the next letter of v, and thus $\delta(q) = \{a, f\}$, a contradiction. Hence $a = b$, and indeed, here d is not the last letter of $\bar{u}$ since $d \neq a = b$ and is the only member in $\delta(q')$ that is not, since $\delta(q') = \{a, d\}$. This proves the first statement of the claim. Also, $b^l f^{\gamma}$ must be read from u since $(\bar{u}f^{\gamma}, \bar{v})$ is the unique decomposition of wf^{γ}, and $d \in \delta(q')$, $d \neq a$ must be read from u. In this case, the claim follows. Suppose instead that $d \notin \delta(q)$, and then $d \in \delta(q') \setminus \delta(q)$ must be read from u. Thus, in either case, $(\bar{u}f^{\gamma}d, \bar{v})$ is the unique decomposition of $wf^{\gamma}d$ and $(\bar{u}f^{\gamma\ (q')}, \bar{v}) = (\bar{u}f^{\gamma}, \bar{v})$ is the unique decomposition of wf^{γ}. □

Thus, at the end of line 6 when the function returns, the inductive conditions are satisfied, and Case 1 is correct.

Otherwise, assume $|\delta(q')| = 1$ (lines 7–37). Starting at q', there is a function call of **maxEqual** at line 8 which will determine $s, t, z \in \Sigma^*$ and for which we then show that z is uniquely $((\bar{u}f^{\gamma})^{-1}u, (\bar{v})^{-1}v)$-decomposable into s and t.

More precisely, let $a_1^{\alpha_1} \cdots a_n^{\alpha_n} = (\bar{u}f^{\gamma})^{-1}u$, and $b_1^{\beta_1} \cdots b_m^{\beta_m} = (\bar{v})^{-1}v$ (this is what remains of u and v), where $a_1, \ldots, a_n, b_1, \ldots, b_m \in \Sigma$, $a_i \neq a_{i+1}$, $b_j \neq b_{j+1}$, for $1 \leq i < n$, $1 \leq j < m$ and $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m \in \mathbb{N}$. The **maxEqual** function will find the largest $i < \max\{m, n\}$ such that $a_1^{\alpha_1} \cdots a_i^{\alpha_i} = s = b_1^{\beta_1} \cdots b_i^{\beta_i} = t$, as proven next.

Claim 2. Let i is the largest integer less than $\max\{n, m\}$ with $a_1^{\alpha_1} \cdots a_i^{\alpha_i} = b_1^{\beta_1} \cdots b_i^{\beta_i}$. Then lines 4-6 of **maxEqual** are executed exactly i times, upon termination $s = t = a_1^{\alpha_1} \cdots a_i^{\alpha_i} = b_1^{\beta_1} \cdots b_i^{\beta_i}$ and after executing the jth iteration, with $1 \leq j \leq i$,

$$q' = \delta(q, f^{\gamma}a_1^{2\alpha_1} \cdots a_j^{2\alpha_j}), q_{prev} = \delta(q, f^{\gamma}a_1^{2\alpha_1} \cdots a_j^{2\alpha_j - 1}).$$

Also, (s, t) is the unique $((\bar{u}f^{\gamma})^{-1}u, (\bar{v})^{-1}v)$-decomposition of $z = a_1^{2\alpha_1} \cdots a_i^{2\alpha_i}$, and $(\bar{u}f^{\gamma}a_1^{\alpha_1} \cdots a_i^{\alpha_i}, \bar{v}a_1^{\alpha_1} \cdots a_i^{\alpha_i})$ is the unique (u, v)-decomposition of

$$wf^{\gamma}a_1^{2\alpha_1} \cdots a_i^{2\alpha_i}.$$

Proof of Claim. If $i = 0$, then this implies that $2 \cdot \epsilon \neq \beta$ in line 4, which implies that $\alpha_1 \neq \beta_1$ because $(\bar{u}f^\gamma, \bar{v})$ is the unique decomposition of wf^γ and so one of u or v does not have $\beta/2$ of the next letter, and the claim holds.

Assume $i > 0$ and by induction, for $1 \leq j \leq i$, after reading $a_1^{2\alpha_1} \cdots a_{j-1}^{2\alpha_{j-1}}$, it is uniquely decomposable into $(a_1^{\alpha_1} \cdots a_{j-1}^{\alpha_{j-1}}, a_1^{\alpha_1} \cdots a_{j-1}^{\alpha_{j-1}})$. Then $\delta^{max}(q', a_j) = \alpha_j + \beta_j$ (number of a_j's we can read from both words), and also $\delta^{min}(q', a_j) = \min\{\alpha_j, \beta_j\}$ (the least amount of a_j's before we can read something else). Then $|\delta(q')| = 1$ and the condition of the while loop on line 2 of **maxEqual** is true, $2 \cdot \delta^{min}(q', a_j) = \delta^{max}(q', a_j)$, since $\alpha_j = \beta_j$. Therefore, by induction, i is maximal less than $\max\{m, n\}$ such that $a_1^{2\alpha_1} \cdots a_i^{2\alpha_i}$ is uniquely decomposable into $(a_1^{\alpha_1} \cdots a_i^{\alpha_i}, a_1^{\alpha_1} \cdots a_i^{\alpha_i})$.

After this has been determined, then we have $q' = \delta(q, f^\gamma a_1^{2\alpha_1} \cdots a_i^{2\alpha_i})$, and if $i > 0$, then $q_{prev} = \delta(q, f^\gamma a_1^{2\alpha_1} \cdots a_i^{2\alpha_i - 1})$. Also, we have determined that $(\bar{u}f^\gamma)^{-1}u$ and $(\bar{v})^{-1}v$ both start with $s = t = a_1^{\alpha_1} \cdots a_i^{\alpha_i} = b_1^{\beta_1} \cdots b_i^{\beta_i}$ which together forms the unique $((\bar{u}f^\gamma)^{-1}u, (\bar{v})^{-1}v)$-decomposition of $z = a_1^{2\alpha_1} \cdots a_i^{2\alpha_i}$. Thus, we know that $(\bar{u}f^\gamma s, \bar{v}t)$ is the unique (u, v)-decomposition of $wf^\gamma z$ and $s = t$. □

Thus, after executing **maxEqual**, which returns $(s, t, z, q_{prev}, q', c, \epsilon, \beta)$, we know $(\bar{u}f^\gamma s, \bar{v}t)$ is the unique decomposition of $wf^\gamma z$. If at this point $|\delta(q')| \geq 2$, then it must be equal to two since it is a unique decomposition by Lemma 1. Then, $\delta(q') = \{e, d\}$ and we pick $d \in \delta(q')$ and we know that either $(\bar{u}f^\gamma t, \bar{v}sd)$ is the unique decomposition of $wf^\gamma zd$, or $(\bar{u}f^\gamma sd, \bar{v}t)$ is the unique decomposition of $wf^\gamma zd$. The following claim shows that we can determine which is true using a test starting at state q (and not q', as we need to test an alternate path starting at q to determine the correct result).

Claim 3. Assume line 9 of **firstDiff** is true (Case 2). Then $i \geq 1$. Also, if $\delta(q, \textbf{maxShuf}(se, f^\gamma t))$ is defined (line 11) then $(\bar{u}f^\gamma sd, \bar{v}t)$ is the unique decomposition of $wf^\gamma zd$. Otherwise, if $\delta(q, \textbf{maxShuf}(se, f^\gamma t))$ is not defined then $(\bar{u}f^\gamma t, \bar{v}sd)$ is the unique decomposition of $wf^\gamma zd$.

Proof of Claim. In this case we know $i \geq 1$ because $|\delta(q')| = 2$, and there was only one transition before executing **maxEqual**. To check which is true, we calculate if $\delta(q, \textbf{maxShuf}(se, f^\gamma t))$ is defined. Indeed, $se = a_1^{\alpha_1} \cdots a_i^{\alpha_i} e$, $f^\gamma t = f^\gamma a_1^{\alpha_1} \cdots a_i^{\alpha_i}, f \neq a_1$, and we know $(\bar{u}^{(q)}, \bar{v})$ is the unique decomposition of $w^{(q)}$ from the inductive conditions. Let b^l be such that $b \in \Sigma, l \geq 0$ is maximal, $\bar{u}$ ends with b^l with $l = 0$ if $\bar{u} = \lambda$.

We know $\textbf{maxShuf}(se, f^\gamma t) = \textbf{maxShuf}(a_1^{\alpha_1} \cdots a_i^{\alpha_i} e, f^\gamma a_1^{\alpha_1} \cdots a_i^{\alpha_i})$, and this has $y = a_1^{\alpha_1} a_2^{\alpha_2 + \theta_2} \cdots a_i^{\alpha_i + \theta_i} e^{\theta_{i+1}}$ as a prefix, where $\theta_{i+1} > 0$ is maximal, $\theta_j \geq 0$ for $2 \leq j \leq i + 1$ (the **maxShuf** function always takes all copies of each letter from the first parameter with all of the next letter from the second parameter if the letters are the same).

Assume either $l = 0$ or $l > 0$ and $b \notin \delta(q)$. We also will assume that $\delta(q, \textbf{maxShuf}(se, f^\gamma t))$ is defined. Then $(\bar{u}^{(q)}, \bar{v}) = (\bar{u}, \bar{v})$ by the assumption, which is a unique decomposition by the inductive conditions. But when reading y from state q, $a_1^{\alpha_1} \cdots a_i^{\alpha_i} e$ must be read from v, as $f \neq a_1$, $(\bar{v})^{-1}v$ starts with $a_1^{\alpha_1} \cdots a_i^{\alpha_i}$, **maxShuf** gives the maximum number of letters in each section, and θ_{i+1} is maximal. This implies $(\bar{u}f^\gamma sd, \bar{v}t)$ is the unique decomposition of $wf^\gamma zd$ as d must be read from u. If instead $\delta(q, \textbf{maxShuf}(se, f^\gamma t))$ is not defined, then the remaining part of v starts with $a_1^{\alpha_1} \cdots a_i^{\alpha_i} d$, and $(\bar{u}f^\gamma t, \bar{v}sd)$ is the unique decomposition of $wf^\gamma zd$.

Assume $b \in \delta(q)$ and $l > 0$. But $(\bar{u}^{(q)})^{-1}u$ must start with $b^l f$ since $\bar{u}^{(q)}$ is defined by the inductive conditions, and $(\bar{v})^{-1}v$ must start with b since $b \in \delta(q)$ and $(\bar{u}f, \bar{v})$ is the unique decomposition of wf. Hence, $a_1 = b$. Furthermore, if $\delta(q, \textbf{maxShuf}(se, f^\gamma t))$ is defined, then when reading y, $a_1^{\alpha_1} \cdots a_i^{\alpha_i} e$ must come from v as $a_1 = b \neq f$, $(\bar{v})^{-1}v$ starts with $a_1^{\alpha_1} \cdots a_i^{\alpha_i}$, **maxShuf** gives the maximum number of letters in each section, and $(\bar{u}^{(q)}, \bar{v})$ is the unique decomposition of $w^{(q)}$. This implies $(\bar{u}f^\gamma sd, \bar{v}t)$ is the unique decomposition of $wf^\gamma zd$ if $\delta(q, \textbf{maxShuf}(se, f^\gamma t))$ is defined. If instead it is not defined then $(\bar{u}f^\gamma t, \bar{v}sd)$ is the unique decomposition. □

In either case, after concatenating $f^\gamma s$ (where $s = t$) to $\bar{u}$ and t to $\bar{v}$, then $(\bar{u}^{(q)}, \bar{v}) = (\bar{u}, \bar{v}) = (\bar{u}, \bar{v}^{(q)})$ which is the unique decomposition of $w = w^{(q)}$ since $i \geq 1$ and $a_i \notin \delta(q)$. The previous claim also intuitively illustrates the purpose of the **maxEqual** function. When in state q' and there are transitions on d and e, it is necessary to examine state q before reading s and t in order to determine whether d is read from u or v.

If $|\delta(q')| = 1$ after executing **maxEqual** in line 8 of **firstDiff**, then in the last iteration of the while loop of **maxEqual**, the condition of the while loop in **maxEqual** will be true, and it will calculate c, the unique next letter, β, the maximal number of c's which can be read, and ϵ, which is the minimal number before reading another letter or reaching a final state. But the if statement will be false, as it is the last iteration, and c, β and ϵ will be returned. In this case, either $\epsilon = \beta$, and lines 16–24 (Case 3) will be executed, or $\epsilon \neq \beta$ and lines 25–37 (Case 4) will be executed. Notice that it is impossible for $|\delta(q')| = 0$ when evaluating the condition of the while loop in **maxEqual** since otherwise the second last time the condition was evaluated, $2\epsilon = \beta$, meaning we have ϵ c's to read from both u and v before reading the end of the word, but then $\epsilon = \beta$, a contradiction. Thus, after executing line 8 of **firstDiff**, either Case 2, 3 or 4 occurs.

Assume $\epsilon = \delta^{max}(q', c) = \beta$ (Case 3, line 16). Notice in this case that $q' \neq q_0$ as M is acyclic and otherwise $f = \bar{u} = \bar{v} = \lambda$ and then one of u or v would be empty, or both would be unary over the same letter as $\epsilon = \beta$, a contradiction. Then either $f \neq \lambda$ or $i \geq 1, f = \lambda$ and $q_{prev} = \delta(q_0, a_1^{2\alpha_1} \cdots a_i^{2\alpha_i - 1})$.

We will split the correctness of this case across two claims, depending on whether line 18 holds or not.

Claim 4. Assume line 18 is true. If $\delta(q, \textbf{maxShuf}(s \cdot \textbf{findUnique}(q'), f^\gamma t))$ is defined (line 20), then $(\bar{u}f^\gamma t, \bar{v}s \cdot \textbf{findUnique}(q'))$ is the unique decomposition of $wf^\gamma z \cdot \textbf{findUnique}(q')$, otherwise $(\bar{u}f^\gamma s \cdot \textbf{findUnique}(q'), \bar{v}t)$ is the unique decomposition. In either case, $\delta(q, f^\gamma z \cdot \textbf{findUnique}(q')) \in F$.

Proof of Claim. Assume first that $f \neq \lambda$ and $i \geq 1$. As we can read as many c's from q_{prev} as we were able to read from q' (line 18 is true), then all of the c's that come from both u and v come from only one word (as we have not read all a_i's from both words). Then from state q', one word has already been read entirely since $|\delta(q')| = 1$ and we can determine the other word with **findUnique**(q'), thus giving the unique $((\bar{u}f^\gamma)^{-1}u, (\bar{v})^{-1}v)$-decomposition of z. Then, either $(\bar{u}f^\gamma t, \bar{v}s \cdot$ **findUnique**$(q'))$ or $(\bar{u}f^\gamma s \cdot$ **findUnique**$(q'), \bar{v}t)$ is the unique decomposition of $wf^\gamma z \cdot$ **findUnique**(q').

To check which is true, we calculate **maxShuf**$(s \cdot$ **findUnique**$(q'), f^\gamma t) =$

$$\textbf{maxShuf}(a_1^{\alpha_1} \cdots a_i^{\alpha_i} \cdot \textbf{findUnique}(q'), f^\gamma a_1^{\alpha_1} \cdots a_i^{\alpha_i}),$$

which has $y = a_1^{\alpha_1} a_2^{\alpha_2+\theta_2} \cdots a_i^{\alpha_i+\theta_i} c^{\theta_{i+1}}$ as prefix, for some θ_j, $2 \leq j \leq i+1$, $\theta_{i+1} > 1$ maximal, $c \neq a_i$, as $f \neq a_1$, and we test if $\delta(q, y)$ is defined. As with the case above, if it is, then $a_1^{\alpha_1} \cdots a_i^{\alpha_i} c$ must be a prefix of what remains of v, and thus $(\bar{u}f^\gamma t, \bar{v}s \cdot$ **findUnique**$(q'))$ is the unique decomposition of $wf^\gamma z \cdot$ **findUnique**(q') and $\delta(q, f^\gamma z \cdot$ **findUnique**$(q'))$ takes the automaton to a final state. Otherwise, $(\bar{u}f^\gamma s \cdot$ **findUnique**$(q'), \bar{v}t)$ is the unique decomposition and we are done determining both words. The cases where $f \neq \lambda$, $i = 0$ and $f = \lambda$, $i \geq 1$ are similar. □

Claim 5. Assume that line 18 is false. If the following transition of M is defined, $\delta(q,$ **maxShuf**$(sc^{\max\{\alpha,\beta-\alpha\}}, f^\gamma t c^{\min\{\alpha,\beta-\alpha\}}))$ (line 20), then the pair $(\bar{u}f^\gamma t c^{\min\{\alpha,\beta-\alpha\}}, \bar{v}sc^{\max\{\alpha,\beta-\alpha\}})$ is the unique decomposition of $wf^\gamma z c^\beta$. Otherwise, $(\bar{u}f^\gamma s c^{\max\{\alpha,\beta-\alpha\}}, \bar{v}tc^{\min\{\alpha,\beta-\alpha\}})$ is the unique decomposition. In either case, $\delta(q, f^\gamma z c^\beta) \in F$.

Proof of Claim. Assume $f \neq \lambda$ and $i \geq 1$. If we cannot read as many c's from q_{prev} as we have read from q' (line 18 is false), then u ends with $f^\gamma a_1^{\alpha_1} \cdots a_i^{\alpha_i}$ followed by some positive number of c's, and v ends with $a_1^{\alpha_1} \cdots a_i^{\alpha_i}$ followed by some positive number of c's (if some word had other letters after, then line 16 would not have been true and if one word has zero c's, then $\alpha = \beta$). Then $\alpha = \delta^{max}(q_{prev}, c)$ gives the number of c's in one of the words (as we have not read all a_i's from both words), and $\beta - \alpha$ gives the other. Then either $(\bar{u}f^\gamma sc^{\max\{\alpha,\beta-\alpha\}}, \bar{v}tc^{\min\{\alpha,\beta-\alpha\}})$ or $(\bar{u}f^\gamma tc^{\min\{\alpha,\beta-\alpha\}}, \bar{v}sc^{\max\{\alpha,\beta-\alpha\}})$ gives the unique decomposition. Then we calculate

$$\textbf{maxShuf}(sc^{\max\{\alpha,\beta-\alpha\}}, f^\gamma tc^{\min\{\alpha,\beta-\alpha\}}) = \textbf{maxShuf}(a_1^{\alpha_1} \cdots a_i^{\alpha_i} c^{\max\{\alpha,\beta-\alpha\}}, f^\gamma a_1^{\alpha_1} \cdots a_i^{\alpha_i} c^{\min\{\alpha,\beta-\alpha\}}),$$

which starts with $y = a_1^{\alpha_1} a_2^{\alpha_2+\theta_2} \cdots a_i^{\alpha_i+\theta_i} c^{\theta_{i+1}}$, where $\theta_j \geq 0$, $2 \leq j \leq i$, θ_{i+1} is maximal and $\theta_{i+1} \geq \max\{\alpha, \beta - \alpha\}$. Similar to the third claim above, if $\delta(q, y)$ is defined, then $a_1^{\alpha_1} \cdots a_i^{\alpha_i} c^{\max\{\alpha,\beta-\alpha\}}$ must be read from v, which implies that $(\bar{u}f^\gamma tc^{\min\{\alpha,\beta-\alpha\}}, \bar{v}sc^{\max\{\alpha,\beta-\alpha\}})$ is the unique decomposition of $wf^\gamma zc^\beta$, otherwise $(\bar{u}f^\gamma sc^{\max\{\alpha,\beta-\alpha\}}, \bar{v}tc^{\min\{\alpha,\beta-\alpha\}})$ is the unique decomposition.

If instead $f \neq \lambda$, $i = 0$ or $f = \lambda$, $i \geq 1$, then the cases are similar. □

Hence, at the end of case 3, we have finished determining both u and v, and indeed the inductive conditions hold.

Lastly, assume $\epsilon = \delta^{min}(q', c) \neq \beta$ (Case 4). Then there must exist $d \neq c$ that can be read from state $\delta(q', c^\epsilon)$ since $\epsilon \neq \beta$ and because all words in $u \sqcup\!\sqcup v$ are of the same length.

Assume first $2\epsilon > \beta$. Then after reading $a_1^{2\alpha_1} \cdots a_i^{2\alpha_i}$, we know that one word ends with $a_1^{\alpha_1} \cdots a_i^{\alpha_i} c^{\beta-\epsilon}$ as we cannot read any letter before reading ϵ c's, and $\beta - \epsilon < \epsilon$ (if there are letters after this segment, then ϵ would not be minimal as $\beta - \epsilon$ is smaller). Then, we read c^β from the state q' and determine the remaining portion of the other word with $r =$ **findUnique**(q').

Then

$$y = \textbf{maxShuf}(sc^\epsilon, f^\gamma t) = \textbf{maxShuf}(a_1^{\alpha_1} \cdots a_i^{\alpha_i} c^\epsilon, f^\gamma a_1^{\alpha_1} \cdots a_i^{\alpha_i})$$

which has as prefix $a_1^{\alpha_1} a_2^{\alpha_2+\theta_2} \cdots a_i^{\alpha_i+\theta_i} c^{\theta_{i+1}}$, where $\theta_j \geq 0$, $2 \leq j \leq i$, $\theta_{i+1} \geq \epsilon$ is maximal. Then, as with case 2 above, if $\delta(q, y)$ is defined, the remaining portion of v must start with tc^ϵ. Thus, $(\bar{u}f^\gamma sc^{\beta-\epsilon}, \bar{v}tc^\epsilon r)$ is the unique decomposition of $wf^\gamma zc^\beta r$. The case is similar if $\delta(q, y)$ is not defined.

If instead $2\epsilon \leq \beta$ (which must not be equal otherwise i would not have been maximal), then consider

$$y = \textbf{maxShuf}(sc^{\epsilon+1}, f^\gamma t) = \textbf{maxShuf}(a_1^{\alpha_1} \cdots a_i^{\alpha_i} c^{\epsilon+1}, f^\gamma a_1^{\alpha_1} \cdots a_i^{\alpha_i})$$

which has as prefix $a_1^{\alpha_1} a_2^{\alpha_2+\theta_2} \cdots a_i^{\alpha_i+\theta_i} c^{\theta_{i+1}}$, where $\theta_j \geq 0$, $2 \leq j \leq i$, $\theta_{i+1} > \epsilon$ is maximal. Then, as with case 2 above, if $\delta(q, y)$ is defined, the remaining portion of v must start with $sc^{\epsilon+1}$. Thus, $(\bar{u}f^\gamma sc^\epsilon d, \bar{v}t)$ is the unique decomposition of $wf^\gamma zc^\epsilon d$. Furthermore, after setting q, $\bar{u}$, $\bar{v}$, w on line 34, then $(\bar{u}^{(q)}, \bar{v})$ has $\bar{u} = \bar{u}^{(q)}c^\epsilon$ and $w = w^{(q)}c^\epsilon$ and since $2\epsilon < \beta$, and thus $\delta(q, c^\epsilon)$ is defined (from the definition of $\bar{u}^{(q)}$), and this is the unique decomposition of $w^{(q)}$. The case is similar if $\delta(q, y)$ is not defined.

This concludes Lemma 3. □

Hence, by induction, each time we iterate the loop of the main procedure and also at the beginning of the main procedure, the inductive conditions hold, and at the end of the algorithm $(\bar{u}, \bar{v})$, is the unique decomposition of $w \in u \sqcup\!\sqcup v$.

The time complexity is proportional to the depth of the automaton as, every time we read some letters, we concatenate them to either $\bar{u}$ or $\bar{v}$, with the exception of Case 4, where we test if $\epsilon \neq \beta$, but then sometimes concatenate only ϵ c's to either $\bar{u}$ or $\bar{v}$. But there, we only need to determine if $2\epsilon < \beta$ without exactly determining β.

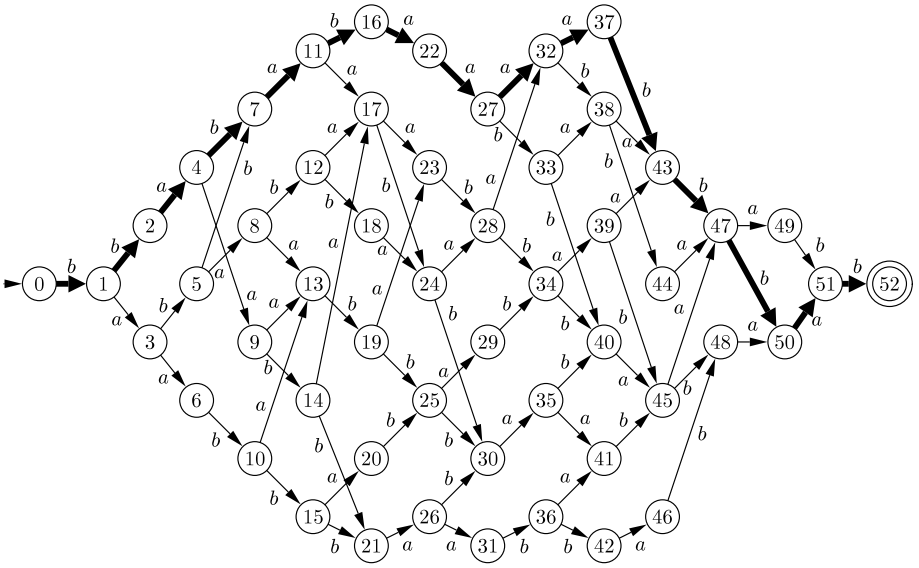

Fig. 2. The trim DFA M, where the word read in $u \sqcup\!\sqcup\ v$ is shown along the path marked in bold.

Theorem 4. *Let M be an acyclic, trim, non-unary DFA over Σ. Then we can determine words $u, v \in \Sigma^+$ such that, $L(M)$ has a shuffle decomposition implies that $L(M) = u \sqcup\!\sqcup\ v$ is the unique decomposition. This can be calculated in time $\mathcal{O}(|u| + |v|)$.*

We note again that this theorem only provides the correct u, v under the assumption that M is decomposable. If M is not decomposable, then depending on the structure of M, the algorithm will either output that there is no decomposition, or output two "incorrect" words (since M does not have a decomposition).

This immediately implies that it can also run in time $\mathcal{O}(M)$, which is always at least $|u| + |v|$ in size. This is indeed an interesting property of shuffle automata, as the words being determined are encoded in a relatively small subset of the automata.

4. An example to illustrate Section 3

We now show how our algorithm works on the DFA M in Fig. 2. As explained previously, x implicitly defines $f, g, q, w, \bar{u}$ and $\bar{v}$ and vice versa.

We start out the Procedure by setting $x = (\lambda, \lambda, q_0, \lambda, \lambda, \lambda)$. As $|\delta(q_0)| = 1$, we call **firstDiff**(x) on line 3.

Inside **firstDiff**$(\lambda, \lambda, q_0, \lambda, \lambda, \lambda)$, we have $g = \lambda$ and set $\gamma := \delta^{max}(q_0, \lambda) = 0, s := \lambda, t := \lambda, z := \lambda, q_{prev} := q_0$ and $q' := \delta(q_0, \lambda) = q_0$. Then, as $|\delta(q_0)| = 1$, we call $(s, t, q_{prev}, q', c, \epsilon, \beta) :=$ **maxEqual**(q_0, q_0) on line 8.

Inside **maxEqual**(q_0, q_0) we enter the while loop on line 2 and set $c := b, \beta := 2$ and $\epsilon = 1$. Then on line 4, $2 \cdot \epsilon = \beta$ holds and we set $s := b, t := b, z := b^2, q_{prev} = q_1$ and $q' = q_2$. Then $|\delta(q_2)| = 1$ and we re-iterate the while loop on line 2 by setting $c := a, \beta := 3$ and $\epsilon = 1$. Then $2 \cdot \epsilon \neq \beta$ and we return $(b, b, b^2, q_1, q_2, a, 1, 3)$.

Back in **firstDiff** we have neither $|\delta(q_2)| = 2$ nor $\epsilon = \beta$ and, hence, **Case 4** applies on line 25. As $2\epsilon < \beta$, then **maxShuf**$(sc^{\epsilon+1}, f^{\gamma}t) =$ **maxShuf**$(baa, b) = b^2a^2$ and $\delta(q_0, b^2a^2) = q_9$ is defined. Hence we set $\bar{u} := ba, \bar{v} := b, w := b^2a$ and $q := q_4$. Then, we return $x := (b, \lambda, q_4, b^2a, ba, b)$.

In the Procedure, we have $|\delta(q_4)| = 2$ and $f \neq \lambda$ and, therefore the condition of the while loop on line 5 still holds and we call **firstDiff**(x) again.

Inside **firstDiff**$(b, \lambda, q_4, b^2a, ba, b)$, we have $g = \lambda$ and set $\gamma := \delta^{max}(q_4, b) = 1, s := \lambda, t := \lambda, z := \lambda, q_{prev} := q_4$ and $q' := \delta(q_4, b) = q_7$.

Then, as $|\delta(q_0)| = 1$, we call $(s, t, q_{prev}, q', c, \epsilon, \beta) :=$ **maxEqual**(q_4, q_7) on line 8. Inside **maxEqual**(q_4, q_7) we set $s := \lambda, t := \lambda$ and enter the while loop on line 2 and set $c := a, \beta := 3$ and $\epsilon = 1$. Then $2 \cdot \epsilon \neq \beta$ and **maxEqual** returns $(\lambda, \lambda, q_4, q_7, a, 3, 1)$.

Back in **firstDiff** we have $|\delta(q_7)| \neq 2$ and $\epsilon = 1 \neq 3 = \beta$ and, hence, we go to **Case 4**, where $2\epsilon < \beta$ and **maxShuf**$(sc^{\epsilon+1}, f^{\gamma}t) =$ **maxShuf**$(aa, b) = a^2b$ and $\delta(q_4, a^2b) = q_{19}$ is defined. Hence we set $\bar{u} := ba \cdot ba, \bar{v} := b \cdot \lambda$, $w := b^2a \cdot ba$ and $q := q_{11}$. Then, we return $x := (b, \lambda, q_{11}, b^2aba, baba, b)$.

In the Procedure, we have $|\delta(q_{11})| = 2$ and $f \neq \lambda$ and we enter the while loop in the Procedure on line 5 and call **firstDiff**(x) again.

In **firstDiff**$(b, \lambda, q_{11}, b^2aba, baba, b)$, then $g = \lambda$ and set $\gamma := \delta^{max}(q_{11}, b) = 1, s := \lambda, t := \lambda, z := \lambda, q_{prev} := q_{11}$ and $q' := \delta(q_{11}, b) = q_{16}$. Then, as $|\delta(q_{16})| = 1$, we call $(s, t, q_{prev}, q', c, \epsilon, \beta) :=$ **maxEqual**(q_{11}, q_{16}) on line 8. Inside **maxEqual**(q_{11}, q_{16}) we set $s := \lambda, t := \lambda$ and enter the while loop on line 2 and set $c := a, \beta := 4$ and $\epsilon = 2$. Then on

line 4, $2 \cdot \epsilon = \beta$ holds and we set $s := a^2$, $t := a^2$, $z := a^4$, $q_{prev} = q_{32}$ and $q' = q_{37}$. Then $|\delta(q_{37})| = 1$ and we re-iterate the while loop on line 2 by setting $c := b$, $\beta := 3$ and $\epsilon := 2$. Then $2 \cdot \epsilon \neq \beta$ and **maxEqual** returns $(a^2, a^2, a^4, q_{32}, q_{37}, b, 2, 3)$.

Back in **firstDiff**, we have $|\delta(q_{37})| \neq 2$, $\epsilon \neq \beta$ and we go to **Case 4**. But $2\epsilon > \beta$ and so $q' := \delta(q_{37}, b^3) = q_{50}$ and $r := ab$. Then

$$\textbf{maxShuf}(sc^{\epsilon}, f^{\gamma}t) = \textbf{maxShuf}(a^2b^2, ba^2) = a^2b^3a^2$$

and $\delta(q_{11}, a^2b^3a^2)$ is defined. Hence, in line 29 we set $\bar{u} := baba \cdot ba^2b$, $\bar{v} := b \cdot a^2b^2ab$, $w := b^2aba \cdot ba^4b^3ab$ and $q := q_{52}$, which is a final state, and we return $x := (\lambda, \lambda, q_{52}, b^2ababa^4b^3ab, bababa^2b, ba^2b^2ab)$.

In the Procedure, we now have $|\delta(q_{52})| = 0$ and therefore we output the candidate solution

$$\{u, v\} = \{bababa^2b, ba^2b^2ab\}.$$

It is not difficult to see that, in fact, $L(M) = bababa^2b \sqcup\!\sqcup ba^2b^2ab$.

5. Efficiently deciding shuffle inclusion

We present an algorithm which decides, for a given DFA $M = (Q, \Sigma, \delta, q_0, F)$ and two words $u, v \in \Sigma^+$ whether $u \sqcup\!\sqcup v \subseteq L(M)$. We then proceed to show the algorithm's correctness and establish its complexity, which is quite low. For the opposite problem (deciding whether $L(M) \subseteq u \sqcup\!\sqcup v$), it is easy to see that deciding whether $L(M) \subseteq u \sqcup\!\sqcup v$ is in $co\mathcal{NP}$. Indeed, $L(M) \not\subseteq u \sqcup\!\sqcup v$ is in $\mathcal{NP}$, as we can guess a word in $L(M)$ and verify that it is not in $u \sqcup\!\sqcup v$ in polynomial time. The exact complexity however is unknown.

We first define a naive shuffle NFA, as introduced and discussed in [5]. Essentially, it is the "natural" NFA produced by using a state associated with every position pair inside both u and v.

Definition 5. Let Σ be a finite alphabet and let $u = u_1 \cdots u_m$, $v = v_1 \cdots v_n \in \Sigma^+$, where $u_i, v_j \in \Sigma$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$. We say M is the naive shuffle NFA for u and v (or the naive NFA for $u \sqcup\!\sqcup v$) if and only if $M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{0, \ldots, m\} \times \{0, \ldots, n\}$, $q_0 = (m, n)$, $F = \{(0, 0)\}$ and δ is defined as follows:

- for $1 \leq k \leq m$, $0 \leq l \leq n$, we have $(k-1, l) \in \delta((k, l), u_{(m-k+1)})$; and
- for $0 \leq k \leq m$, $1 \leq l \leq n$, we have $(k, l-1) \in \delta((k, l), v_{(n-l+1)})$.

For all i and j with $1 \leq i \leq m$ and $1 \leq j \leq n$ we denote by $\overline{u}_i$ and $\overline{v}_j$ the suffixes of length i and j of the words u and v, respectively. We furthermore define

$$L_M(i, j) = \overline{u}_i \sqcup\!\sqcup \overline{v}_j,$$

which is accepted by the automaton M', where $M' = (Q, \Sigma, \delta, (i, j), F)$. The subscript M is omitted when there is no possibility for confusion.

We now need to define, how to compute the "next" set of states for a given subset of states of a naive NFA and a letter. Let $u, v \in \Sigma^+$, and let $M' = (Q', \Sigma', \delta', q_0', F')$ be the naive shuffle NFA of u and v, then we define $\text{next}_{(u,v)} : 2^{Q'} \times \Sigma \to 2^{Q'}$ by

$$\text{next}_{(u,v)}(X, a) = \bigcup_{(i,j) \in X} \{(i', j') \mid (i', j') \in \delta'((i, j), a)\}.$$

In the following, we assume that $M = (Q, \Sigma, \delta, q_0, F)$ is a deterministic finite automaton. The algorithm below calculates a label associated with each state of M. For each prefix w of length i of any word in $u \sqcup\!\sqcup v$, the algorithm assigns all the states of M' that are in $\delta'(q_0', w)$ is placed in the label associated with the unique state $\delta(q_0, w) \in Q$. Then, we can examine the labels to decide whether or not $u \sqcup\!\sqcup v \subseteq L(M)$.

Subset(M, u, v)
1 $\text{label}_0(q_0) = \{(|u|, |v|)\}$;
2 for($i = 0$ to $|u| + |v| - 1$)
3 foreach state $q \in Q$ with $\text{label}_i(q) \neq \emptyset$
4 foreach $a \in \Sigma$
5 $x := \text{next}(\text{label}_i(q), a)$;
6 if ($x \neq \emptyset$)
7 if ($\delta(q, a)$ is undefined)
8 return false;
9 else
10 $\tilde{q} := \delta(q, a)$;
11 $\text{label}_{i+1}(\tilde{q}) := \text{label}_{i+1}(\tilde{q}) \cup x$;
12 foreach ($q \in Q$ with $\text{label}_{|u|+|v|}(q) \neq \emptyset$)
13 if ($q \notin F$)
14 return false;
15 return true;

We now show that the algorithm **Subset**(M, u, v) does in fact decide whether $u \sqcup v \subseteq L(M)$ in low polynomial time with respect to the input parameters.

Theorem 6. *Let $M = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton and let u, v be words over some alphabet Σ, such that $|v| \leq |u|$. Then the algorithm* **Subset**(M, u, v) *returns "true" if and only if $u \sqcup v \subseteq L(M)$ in time $\mathcal{O}((|u| + |v|) \cdot |Q| \cdot |\Sigma| \cdot |v|)$.*

Proof. The proof in broken into several claims to increase readability.

Claim 6. **Subset**(M, u, v) returns "true" if $u \sqcup v \subseteq L(M)$.

Proof of Claim. Let

$$u = u_1 \cdots u_m \text{ and } v = v_1 \cdots v_n,$$

where $u_1, \ldots, u_m, v_1, \ldots, v_n \in \Sigma$. We show by induction that after iteration i of the for-loop on line 2, for each word $w \leq_p \tilde{w} \in u \sqcup v$, $|w| = i + 1$, we have

$$\{(m - j, n - l) \mid w \in u_1 \cdots u_j \sqcup v_1 \cdots v_l\} \subseteq \text{label}_{i+1}(\delta(q_0, w)).$$

Let b and c be the first letters of u and v, respectively (where, naturally, $b = c$ is possible), which implies that b and c are the only prefixes of $u \sqcup v$ of length 1.

We first show that after iteration 0, we have $\{(m - 1, n)\} \subseteq \text{label}_1(\delta(q_0, b))$ and $\{(m, n - 1)\} \subseteq \text{label}_1(\delta(q_0, c))$. Lines 5 through 11 with $i := 0$, $q := q_0$ and $a := b$ assign

$$x := \text{next}(\text{label}_0(q_0), b) = \text{next}(\{(m, n)\}, b) \supseteq \{(m - 1, n)\}.$$

As $u \sqcup v \subseteq L(M)$, $\delta(q_0, b)$ is defined and assign $\tilde{q} = \delta(q_0, b)$ and thus, at the end of iteration 0, we have

$$\{(m - 1, n)\} \subseteq \text{label}_1(\tilde{q}) = \text{label}_1(\delta(q_0, b)).$$

Similarly we can show that after iteration 0, we have

$$\{(m, n - 1)\} \subseteq \text{label}_1(\delta(q_0, c)).$$

Now assume that after iteration $k - 1$, where $0 < k \leq |u| + |v|$, for each word $w' \leq_p \tilde{w} \in u \sqcup v$, $|w'| = k$, we have $\{(m - j, n - l) \mid w' \in u_1 \cdots u_j \sqcup v_1 \cdots v_l\} \subseteq \text{label}_k(\delta(q_0, w'))$.

Let $i = k$ and $w \leq_p \tilde{w} \in u \sqcup v$ with $|w| = k + 1$. Then $w = w'b$ for some word w' with $|w'| = k$ and some letter $b \in \Sigma$, and we know by our assumption that after iteration $k - 1$ we have $\{(m - j, n - l) \mid w' \in u_1 \cdots u_j \sqcup v_1 \cdots v_l\} \subseteq \text{label}_k(\delta(q_0, w'))$. Then, when executing lines 5 through 11 with $i := k$, $q := q' = \delta(q_0, w')$ and $a := b$, we get $x := \text{next}(\text{label}_k(q'), b) \supseteq \text{next}(\{(m - j, n - l) \mid w' \in u_1 \cdots u_j \sqcup v_1 \cdots v_l\}, b)$ by the inductive hypothesis, which is equal to $\{(m - j, n - l) \mid w'b \in u_1 \cdots u_j \sqcup v_1 \cdots v_l\}$. As $u \sqcup v \subseteq L(M)$, we have $\delta(q', b) \neq \emptyset$, which implies that

$$\begin{aligned}\text{label}_{k+1}(\delta(q', b)) &\supseteq \text{next}(\text{label}_k(q'), b)\\ &\supseteq \{(m - j, n - l) \mid w'b \in u_1 \cdots u_j \sqcup v_1 \cdots v_l\}.\end{aligned}$$

Hence, we know by induction that after iteration $|u| + |v| - 1$ of the for-loop on line 2, for each word $w \in u \sqcup v$, we have $\{(0, 0)\} \subseteq \text{label}_{|u|+|v|}(\delta(q_0, w))$. But then, as $u \sqcup v \subseteq L(M)$, we must have $\delta(q_0, w) \in F$ for all $w \in u \sqcup v$ and the algorithm returns "true" on line 15. □

Claim 7. **Subset**(M, u, v) returns "false" if $u \sqcup v \not\subseteq L(M)$.

Proof of Claim. Let $L' = u \sqcup v \setminus L(M)$. As $u \sqcup v \not\subseteq L(M)$, it follows that L' is non-empty. For each $w \in L'$, we define

$$\max(w) = \max\{|w'| \mid w' \leq_p w, \delta(q_0, w') \neq \emptyset\}, \text{ and}$$
$$k = \min_{w \in L'}\{\max(w)\}.$$

Assume first that $k < |u| + |v|$. Then there exists a word $w \in L'$, with $w = w'bw''$, where $w', w'' \in \Sigma^*$, $|w'| = k$ and $b \in \Sigma$, such that $\delta(q_0, w')$ is defined but $\delta(q_0, w'b)$ is undefined. We know from the proof of the previous claim, and by the minimality of w', that after iteration $k - 1$ of the for-loop on line 2, we have

$$\{(m - j, n - l) \mid w' \in u_1 \cdots u_j \sqcup v_1 \cdots v_l\} \subseteq \text{label}_k(\delta(q_0, w')).$$

Then, during iteration k, when executing lines 5 through 11 with $q := \delta(q_0, w')$ and $a := b$, we obtain $x \neq \emptyset$, as $w' \leq_p \tilde{w} \in u \sqcup v$, but $\delta(\delta(q_0, w'), b)$ is undefined, and, hence, the algorithm returns "false".

Now assume that $k = |u| + |v|$. Then, by the proof of the previous claim, after iteration $|u| + |v| - 1$ of the for-loop on line 2, for each word $w \in u \sqcup v$, we have

$$\{(0, 0)\} \subseteq \text{label}_{|u|+|v|}(\delta(q_0, w)).$$

Let $w' \in L'$, $|w'| = |u| + |v|$. Then $\text{label}_{|u|+|v|}(\delta(q_0, w'))$ is defined, but, as $w' \notin L(M)$, $\delta(q_0, w') \notin F$ and the algorithm returns "false". □

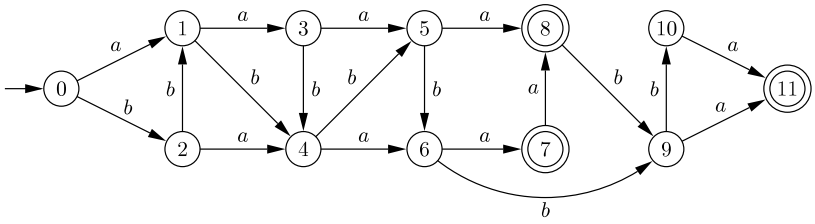

Fig. 3. Pictured above is a DFA M, such that $bba \sqcup\!\sqcup aba \subseteq L(M)$.

Table 1
The entries contain $\text{label}_i(q)$, for all $q \in Q_M$ and $0 \leq i \leq |u| + |v|$.

	0	1	2	3	4	5	6
q_0	{(3, 3)}						
q_1		{(3, 2)}	{(1, 3)}				
q_2		{(2, 3)}					
q_3				{(0, 3), (1, 2)}			
q_4			{(2, 2), (3, 1)}		{(1, 1)}		
q_5				{(1, 2), (2, 1)}	{(0, 2)}		
q_6				{(3, 0)}	{(1, 1)}	{(1, 0), (0, 1)}	
q_7						{(1, 0), (0, 1)}	{(0, 0)}
q_8					{(2, 0), (0, 2)}		{(0, 0)}
q_9					{(2, 0)}	{(1, 0), (0, 1)}	
q_{10}						{(1, 0)}	
q_{11}							{(0, 0)}

Claim 8. **Subset**(M, u, v) operates in time $\mathcal{O}((|u| + |v|) \cdot |Q| \cdot |\Sigma| \cdot |v|)$.

Proof of Claim. The three loops beginning on lines 2, 3 and 4, operate in time $\mathcal{O}(|u|+|v|)$, $\mathcal{O}(|Q|)$ and $\mathcal{O}(|\Sigma|)$, respectively. Inside the foreach-loop on line 4 every step takes constant time except for lines 5 and 11, which takes time $\mathcal{O}(|v|)$, as there can be at most $|v|$ elements in $\text{label}_i(q)$ for any $0 \leq i \leq |u| + |v|$ and $q \in Q$. Thus the nested loops starting on line 2 operate in time $\mathcal{O}((|u| + |v|) \cdot |M| \cdot |\Sigma| \cdot |v|)$, which is the time complexity of the entire algorithm, as the remaining foreach-loop on line 12 operates in time $\mathcal{O}(|M|)$. □

This proves the theorem. □

5.1. Example

We now apply the algorithm to the automaton M pictured in Fig. 3 and $u = bba$ and $v = aba$.

When executing **Subset**(M, u, v), The entries of Table 1 represent which sets are assigned to which states and labeling functions at the end of the algorithm. As all the states $q \in Q$ which have $\text{label}_6(q) \neq \emptyset$ are final states, the algorithm returns "true". It is easy to see that $u \sqcup\!\sqcup v \subsetneq L(M)$, as, for example, $abaa \in L(M)$.

6. Conclusions and discussion

We provide an algorithm which takes as input a trim, acyclic, non-unary DFA, and either indicates that the automaton does not accept the shuffle of two words, or determines two words such that if the DFA is decomposable, then the two words is the unique solution. We cannot tell if the DFA is itself decomposable. However, if the algorithm outputs two words u and v, then we can attempt to test equivalence of $u \sqcup\!\sqcup v$ with M. We can then confirm that $u \sqcup\!\sqcup v \subseteq L(M)$ in polynomial time. However, it is unknown if we can additionally test whether $L(M) \subseteq u \sqcup\!\sqcup v$ in polynomial time.

Other open questions involving shuffle on words involve improving the lower and upper bounds on the size of the minimal DFAs accepting the shuffle of two words. Although an exponential upper and lower bound is known, the bound is not tight.

Acknowledgments

Research supported in part by grants from the Natural Sciences and Engineering Research Council of Canada.

References

[1] C. Câmpeanu, K. Salomaa, S. Vágvölgyi, Shuffle decompositions of regular languages, International Journal of Foundations of Computer Science 13 (2002) 799–816.
[2] M. Ito, Shuffle decomposition of regular languages, Journal of Universal Computer Science 8 (2) (2002) 257–259.

[3] J. Berstel, L. Boasson, Shuffle factorization is unique, Theoretical Computer Science 273 (2002) 47–67.
[4] F. Biegler, M. Daley, M. Holzer, I. McQuillan, On the uniqueness of shuffle on words and finite languages, Theoretical Computer Science 410 (2009) 3711–3724.
[5] F. Biegler, M. Daley, I. McQuillan, On the shuffle automaton size for words, in: Proceedings of the workshop on Descriptional Complexity of Formal Systems, DCFS 2009, 2009, pp. 115–126.
[6] C. Câmpeanu, K. Salomaa, S. Yu, Tight lower bound for the state complexity of shuffle of regular languages, Journal of Automata, Languages and Combinatorics 7 (2002) 303–310.
[7] S. Yu, Regular languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Vol. 1, Springer, Berlin Heidelberg, 1997, pp. 41–110.