Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 341 (2005) 263–275
Theoretical Computer Science
www.elsevier.com/locate/tcs

Synchronized shuffles

Maurice H. ter Beek[a,*], Carlos Martín-Vide[b], Victor Mitrana[b,c]

[a]*Istituto di Scienza e Tecnologie dell'Informazione, CNR, Via G. Moruzzi 1, 56124 Pisa, Italy*
[b]*Research Group in Mathematical Linguistics, Rovira i Virgili University, Pça. Imperial Tarraco 1, 43005 Tarragona, Spain*
[c]*Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei 14, 70109 Bucharest, Romania*

Received 26 April 2004; received in revised form 18 April 2005; accepted 20 April 2005

Communicated by T. Yokomori

Abstract

We extend the basic shuffle on words and languages, a well-known operation in theoretical computer science, by introducing three synchronized shuffles. These synchronized shuffles have some relevance to molecular biology since they may be viewed as the formal representations of various forms of gene linkage during genome shuffling. More precisely, each synchronized shuffle preserves the genetic backbone of the organisms, as well as the linked genes, by requiring the synchronization of some predefined genes while all other genes are arbitrarily shuffled. As for their mathematical properties, we prove that in a trio the closure under shuffle is equivalent to the closure under any of the synchronized shuffles studied here. Finally, based on this result, we present an algorithm for deciding whether a given regular language is synchronized shuffle closed.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Shuffles; Synchronized shuffles; Gene linkage; Genome shuffling

1. Introduction

Genetics is the study of the function and behaviour of genes, biochemical instructions included in the *deoxyribonucleic acid* (DNA) found inside the cells of all organisms. Genes

* Corresponding author. Tel.: +39 050 3153471; fax: +39 050 3152810.
E-mail addresses: maurice.terbeek@isti.cnr.it (M.H. ter Beek), carlos.martin@urv.net (C. Martín-Vide), vmi@urv.net (V. Mitrana).

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.04.007

can be thought of as words over the alphabet of nucleic acids adenine, cytosine, thymine, and guanine. Genes are arranged into chromosomes, one or more of which form a genome. A genome can thus be seen as a language over the same alphabet. Increasingly, genetics focuses on engineering techniques to manipulate genes and even complete genomes.

Bacteria, viruses, and other microbes evolve much faster than most mammals do, in fact giving rise to a new generation as often as once every few minutes. In addition, they mix and match their genetic information on a regular basis. In the microbial world, such gene mutation and recombination regularly gives rise to microbes with new options. Confronted with the genetic flexibility in shuffling the genes in a gene pool, some believe that this reality of nature equates mega-evolution, even though the process never results in an entirely new kind of life. Others, like us, claim that this is impossible. A deck of 52 cards can be repeatedly shuffled and dealt to produce a wide variety of hands, but the result never adds new cards (information) to the deck. Likewise, no new information is added to the gene pool by shuffling the genes or by gene mutations. While mutations may damage or destroy one or more genes in the gene pool, such modifications can never launch one kind of life form into another. Whatever gene mutations or shuffles, a bacterium still descends from a bacterium, remains a bacterium, and will propagate only other bacteria—even in millions of years. In other words, whatever the gene mutation or gene shuffle, the output still preserves a “backbone” of the original organism.

A possible explanation is the phenomenon of gene linkage, the hereditary association of genes located on the same chromosome. Genes being close to each other on a chromosome tend to remain together over time (where one goes they all go). When the genetic code is shuffled in each generation, genes being close to each other thus tend to remain close to each other in future offspring—a major gene shuffle involving these genes thus being a rare event. The same phenomenon occurs when shuffling a deck of cards: two cards next to each other have a high probability of remaining next to each other for many more shufflings. Gene linkage thus explains why multiple diseases are sometimes inherited together: if two defective genes, coding two diseases, are right next to each other on a chromosome, they tend to be inherited as a single unit.

Recently, genetic engineers have developed techniques of gene shuffling and genome shuffling as in vitro processes that mimic the natural evolutionary processes of gene mutation and recombination on an accelerated scale. This process can be thought of as repeatedly shuffling and splitting two decks of cards based on tests for desired characteristics. In [24], it was shown how a combination of gene shuffling and recombination can be used to direct the evolution of the bacterium *Streptomyces fradiae*, which is widely used to produce antibiotics. In [20], on the other hand, the manipulation of the plasmids k1 and k2 (particular types of genomes of circular shape) from the dairy yeast *Kluyveromyces lactis* by means of gene shuffling was exploited to investigate gene function.

Along the same lines we mention a computational framework for modelling the logical behaviour of a class of gene networks, mainly based on language shuffle under certain types of constraints, which is discussed in [7]. The individual genes of the networks are modelled by finite automata. The interaction mechanism among the genes is regulated by some constraints, called positive and negative controls. By computing the shuffle of languages defined by genes under these constraints one obtains the complete set of pathways in the given gene network.

In this article we extend the basic shuffle operation on words and languages, well known in theoretical computer science, by introducing three synchronized shuffle operations. These synchronized shuffles have some relevance to molecular biology since they may be viewed as the formal representations of various forms of gene linkage during genome shuffling. More precisely, each synchronized shuffle preserves the genetic backbone of the organisms, as well as the linked genes, by requiring the synchronization of some predefined genes while all the other genes are arbitrarily shuffled. In theoretical computer science these synchronized shuffles come into play, e.g., when one tries to prove that an automata-based specification model satisfies compositionality, in the sense that the specification (behaviour) of a composite system (automaton) can be obtained from those of its constituents.

This article is organized as follows. We begin by formally defining the synchronized shuffle operations, after which we compare them to similar operations from the literature. Subsequently, we show that in a trio the closure under shuffle is equivalent to the closure under any of these synchronized shuffles. Based on this result, we then give an algorithm for deciding whether a given regular language is synchronized shuffle closed. Finally, we briefly come back to the relevance of synchronized shuffles for proving the compositionality of automata-based models.

2. Preliminaries

We assume familiarity with some basic notions from algebra and formal language theory. For any unexplained notion, we refer the reader to [19,23].

We have the following conventions. Set inclusion is denoted by $\subseteq$. The set difference of sets V and W is denoted by $V \setminus W$. The powerset of a set V is denoted by $\mathcal{P}(V)$ and the empty set is denoted by $\varnothing$. For convenience we sometimes denote the set $\{1, 2, \ldots, n\}$ by $[n]$. Then $[0] = \varnothing$. The empty word is denoted by λ. For a word $v \in V^*$, we use $|v|$ to denote its length. Thus $|\lambda| = 0$. The alphabet of v is denoted by $\alpha(v)$ and consists of all symbols that actually occur in v. Thus $\alpha(\lambda) = \varnothing$.

Let Σ and Γ be two alphabets. A morphism $h : \Sigma^* \longrightarrow \Gamma^*$ such that $|h(a)| \leqslant 1$ for any $a \in \Sigma$ is called a *weak literal* morphism. The function pres_Γ, also called the *projection* on Γ, is a weak literal morphism from Σ^* into Γ^* defined by $\text{pres}_\Gamma(a) = a$ if $a \in \Gamma$ and $\text{pres}_\Gamma(a) = \lambda$ otherwise. In other words, pres_Γ preserves the symbols from Γ and erases all other symbols. By convention, $\text{pres}_\varnothing(v) = \lambda$, for any word v.

A *finite transducer* is a sixtuple $M = (Q, V, U, f, q_0, F)$, where Q is the set of states, V is the input alphabet, U is the output alphabet, f is the transition-and-output mapping from $Q \times (V \cup \{\lambda\})$ to finite subsets of $Q \times U^*$, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. If f is a function from $Q \times V$ to finite subsets of $Q \times U^*$, i.e. M reads exactly one symbol at each transition, then A is said to be a *sequential transducer*. In the literature, these devices are also called *generalized sequential machines* [9]. The transition function may be extended in a natural way to $Q \times V^*$. Each finite transducer M as above defines a *finite transduction*

$$T_M(v) = \{\, u \in U^* \mid (q, u) \in f(q_0, v),\ q \in F \,\},$$

which can be extended to a language $L \subseteq V^*$ by $T_M(L) = \bigcup_{v \in L} T_M(v)$.

3. Shuffles and synchronized shuffles

A shuffle of two words is an arbitrary interleaving of subwords of these words such that it contains all symbols of both words, like shuffling two decks of cards. This is a well-known language-theoretic operation with a long history in theoretical computer science, in particular within formal languages. In the literature shuffling may also be called interleaving, weaving, or merging, and—given words u and v—denoted by $u \odot v$, $u \,||\, v$, $u \sqcup\!\sqcup v$, $u \,\square\, v$, $u \otimes v$, $u \,|||\, v$, or $u \diamond v$ [6,10–12,17,19,21]. The underlying idea also appears in other forms throughout computer science, within concurrency theory, e.g., as semantics of parallel operators modelling communication between processes [1,5,18].

3.1. Shuffle

Formally, the *shuffle* of words $u, v \in \Sigma^*$ is a set of words denoted by $u \sqcup\!\sqcup v$ and defined recursively as

$$x \sqcup\!\sqcup \lambda = \lambda \sqcup\!\sqcup x = \{x\}, \quad x \in \Sigma^*,$$

and

$$ax \sqcup\!\sqcup by = a(x \sqcup\!\sqcup by) \cup b(ax \sqcup\!\sqcup y), \quad a, b \in \Sigma, \ \ x, y \in \Sigma^*.$$

The shuffle of two languages $L_1, L_2 \subseteq \Sigma^*$ is denoted by $L_1 \sqcup\!\sqcup L_2$ and is defined as a language consisting of all words that are a shuffle of a word from L_1 and a word from L_2. Thus

$$L_1 \sqcup\!\sqcup L_2 = \{\, w \in u \sqcup\!\sqcup v \mid u \in L_1,\ v \in L_2 \,\} = \bigcup_{u \in L_1,\, v \in L_2} u \sqcup\!\sqcup v.$$

Note that the shuffle of two words never equals the empty set, i.e. given $u, v \in \Sigma^*$, then $u \sqcup\!\sqcup v \neq \emptyset$. Furthermore, given a word $w \in u \sqcup\!\sqcup v$, it is clear that $\alpha(w) = \alpha(u) \cup \alpha(v)$ and that $|w| = |u| + |v|$. Finally, it is plain that the shuffle operation is both commutative and associative, i.e. $u \sqcup\!\sqcup v = v \sqcup\!\sqcup u$ and $(u \sqcup\!\sqcup v) \sqcup\!\sqcup w = u \sqcup\!\sqcup (v \sqcup\!\sqcup w)$, for all words $u, v, w \in \Sigma^*$.

We continue our exposition on shuffles by introducing three more intriguing types, built on top of the basic shuffle. Rather than freely interleaving the letters of the words being shuffled, part of these letters are now synchronized. These synchronized letters, while occurring in each of the words being shuffled, thus occur only once in the resulting words. Furthermore, the sequence of synchronized letters forms a "backbone" of the resulting words, which means that there is an order that the letters being synchronized must adhere to.

As was the case for shuffling, the idea underlying these synchronized shuffles is not new, but it appears in other forms throughout computer science. The idea seems to stem from the concurrent composition $P \oplus Q$ of synchronizing processes P and Q as defined in [13]. Within formal languages, a slightly adapted version of the idea was introduced in [8] as the 'produit de mixage' $u \sqcap\!\sqcap v$ of words u and v. This operation was renamed synchronized shuffle in [15] and in [2,4] it was generalized to the synchronized shuffle (S-shuffle for short) $u \,\|^{\Gamma}\, v$ on an arbitrary alphabet Γ (of letters subject to synchronization) of (possibly infinite) words u and v. In [2,4] two special cases of the S-shuffle were also defined, viz.—given a word u over Σ_1 and a word v over Σ_2—the relaxed synchronized shuffle (rS-shuffle for short) $u\,{}_{\Sigma_1}\underline{\|}^{\Gamma}{}_{\Sigma_2}\,v = u\,\|^{\Gamma\cap\Sigma_1\cap\Sigma_2}\,v$ on an arbitrary alphabet Γ of u and v, and the full synchronized shuffle (fS-shuffle for short) $u\,{}_{\Sigma_1}\underline{\|}{}_{\Sigma_2}\,v = u\,\|^{\Sigma_1\cap\Sigma_2}\,v$ of u and v, both with respect to Σ_1 and Σ_2. Note that the S-shuffle on an arbitrary alphabet $\Gamma \subseteq (\Sigma_1 \cup \Sigma_2)$ of $u \in \Sigma_1^*$ and $v \in \Sigma_2^*$ is a slight generalization of both the concurrent composition as defined in [13], which requires $\Sigma_1 = \Sigma_2$, and the synchronized shuffle operation as defined in [8,15], which requires $\Gamma = \alpha(u) \cap \alpha(v)$. These operations can thus be distinguished by the alphabet on which they require words to synchronize. Within concurrency theory, finally, two more slightly adapted versions of the idea were introduced in [22] as the weave $u\,\underline{w}\,v = u\,{}_{\Sigma_1}\underline{\|}{}_{\Sigma_2}\,v$ of words u and v, and in [18] as the alphabetized parallel composition $P\,{}_X\underline{\|}{}_Y\,Q$ of processes P and Q—given alphabets X and Y.

Also, the new synchronized shuffles that we define next can be distinguished from each other by the manner in which they require the words to synchronize. As we proceed, we will briefly compare our variants to those above, but for a more complete comparison—including (fair) synchronized shuffles on infinite words and infinitary languages—we refer the reader to [2].

3.2. *Strongly synchronized shuffle*

Given words $u, v \in \Sigma^*$ and a subset Γ of Σ, the strongly synchronized shuffle of u and v on Γ requires u and v to synchronize on all occurrences of the letters from Γ, while all other letters are shuffled. This means that $\mathrm{pres}_\Gamma(u) = \mathrm{pres}_\Gamma(v)$ must hold.

Formally, the *strongly synchronized shuffle* (*SS-shuffle* for short) of words $u, v \in \Sigma^*$ on $\Gamma \subseteq \Sigma$ is denoted by $u \sqcup\!\sqcup_{\Gamma}^{S} v$ and is defined as

$$\begin{aligned} u \sqcup\!\sqcup_{\Gamma}^{S} v = \{\, & (u_1 \sqcup\!\sqcup v_1)x_1(u_2 \sqcup\!\sqcup v_2)x_2 \cdots x_{n-1}(u_n \sqcup\!\sqcup v_n) \mid n \geqslant 1, \\ & u_i, v_i \in (\Sigma \setminus \Gamma)^*,\ i \in [n],\ x_1, x_2, \ldots, x_{n-1} \in \Gamma, \\ & u_1x_1u_2x_2 \cdots x_{n-1}u_n = u,\ v_1x_1v_2x_2 \cdots x_{n-1}v_n = v \,\}. \end{aligned}$$

Note that $u \sqcup\!\sqcup_{\Gamma}^{S} v = \varnothing$ as soon as $\mathrm{pres}_\Gamma(u) \neq \mathrm{pres}_\Gamma(v)$. Let $w \in u \sqcup\!\sqcup_{\Gamma}^{S} v$. Then $\mathrm{pres}_\Gamma(w)$ is called the *backbone* of w. Clearly, $\mathrm{pres}_\Gamma(w) = \mathrm{pres}_\Gamma(u) = \mathrm{pres}_\Gamma(v)$, for all $w \in u \sqcup\!\sqcup_{\Gamma}^{S} v$, i.e. all words in $u \sqcup\!\sqcup_{\Gamma}^{S} v$ have the same backbone.

3.3. *Weakly synchronized shuffle*

Given words $u, v \in \Sigma^*$ and a subset Γ of Σ, the weakly synchronized shuffle of u and v on Γ requires u and v to synchronize on some occurrences of the letters from Γ, while all other occurrences together with the letters not appearing in Γ are shuffled, provided that each pair of subwords that is to be shuffled cannot synchronize on any occurrence of the letters from Γ. Now $\mathrm{pres}_\Gamma(u) = \mathrm{pres}_\Gamma(v)$ does not necessarily hold anymore.

Formally, the *weakly synchronized shuffle* (*WS-shuffle* for short) of words $u, v \in \Sigma^*$ on $\Gamma \subseteq \Sigma$ is denoted by $u \sqcup\!\sqcup_{\Gamma}^{W} v$ and is defined as

$$\begin{aligned} u \sqcup\!\sqcup_{\Gamma}^{W} v = \{\, & (u_1 \sqcup\!\sqcup v_1)x_1(u_2 \sqcup\!\sqcup v_2)x_2 \cdots x_{n-1}(u_n \sqcup\!\sqcup v_n) \mid n \geqslant 1, \\ & u_i, v_i \in \Sigma^*,\ \alpha(u_i) \cap \alpha(v_i) \cap \Gamma = \varnothing, \end{aligned}$$

$$i \in [n],\ x_1, x_2, \ldots, x_{n-1} \in \Gamma,$$
$$u_1x_1u_2x_2 \cdots x_{n-1}u_n = u,\ v_1x_1v_2x_2 \cdots x_{n-1}v_n = v\,\}.$$

Note that $\text{pres}_\Gamma(u) = \text{pres}_\Gamma(v)$ does not imply that $u \sqcup\!\sqcup\,^W_\Gamma\, v = u \sqcup\!\sqcup\,^S_\Gamma\, v$, but $u \sqcup\!\sqcup\,^S_\Gamma\, v \subseteq u \sqcup\!\sqcup\,^W_\Gamma\, v$ always holds. Let $w \in u \sqcup\!\sqcup\,^W_\Gamma\, v$ be such that $w = w_1x_1w_2x_2 \cdots x_{n-1}w_n$, $w_i \in u_i \sqcup\!\sqcup\, v_i$, $u_i, v_i \in \Sigma^*$, $\alpha(u_i) \cap \alpha(v_i) \cap \Gamma = \emptyset$, $i \in [n]$, $x_1, x_2, \ldots, x_{n-1} \in \Gamma$, $u_1x_1u_2x_2 \cdots x_{n-1}u_n = u$, and $v_1x_1v_2x_2 \cdots x_{n-1}v_n = v$. Then $x_1x_2 \cdots x_{n-1}$ is called the backbone of w w.r.t. this decomposition. Contrary to the SS-shuffle, different words in $u \sqcup\!\sqcup\,^W_\Gamma\, v$ may thus have different backbones. Also note that the weakly synchronized shuffle of two words $u, v \in \Sigma^*$ on $\Gamma \subseteq \Sigma$ is always a nonempty set.

3.4. Arbitrarily synchronized shuffle

Given words $u, v \in \Sigma^*$ and a subset Γ of Σ, the arbitrarily synchronized shuffle of u and v on Γ requires u and v to synchronize on some occurrences of the letters from Γ or none, while all other occurrences together with the letters not appearing in Γ are shuffled.

Formally, the *arbitrarily synchronized shuffle* (*AS-shuffle* for short) of words $u, v \in \Sigma^*$ on $\Gamma \subseteq \Sigma$ is denoted by $u \sqcup\!\sqcup\,^A_\Gamma\, v$ and is defined as

$$\begin{aligned} u \sqcup\!\sqcup\,^A_\Gamma\, v = \{\, & (u_1 \sqcup\!\sqcup\, v_1)x_1(u_2 \sqcup\!\sqcup\, v_2)x_2 \cdots x_{n-1}(u_n \sqcup\!\sqcup\, v_n) \mid n \geqslant 1, \\ & u_i, v_i \in \Sigma^*,\ i \in [n],\ x_1, x_2, \ldots, x_{n-1} \in \Gamma, \\ & u_1x_1u_2x_2 \cdots x_{n-1}u_n = u,\ v_1x_1v_2x_2 \cdots x_{n-1}v_n = v\,\}. \end{aligned}$$

Note that $\text{pres}_\Gamma(u) = \text{pres}_\Gamma(v)$ does not imply $u \sqcup\!\sqcup\,^A_\Gamma\, v = u \sqcup\!\sqcup\,^S_\Gamma\, v$, but $u \sqcup\!\sqcup\,^S_\Gamma\, v \subseteq u \sqcup\!\sqcup\,^A_\Gamma\, v$ and $u \sqcup\!\sqcup\,^W_\Gamma\, v \subseteq u \sqcup\!\sqcup\,^A_\Gamma\, v$ always hold. As for WS-shuffles, we define a decomposition of $w \in u \sqcup\!\sqcup\,^A_\Gamma\, v$ and the backbone of w w.r.t. this decomposition. Again, different words in $u \sqcup\!\sqcup\,^A_\Gamma\, v$ may have different backbones.

Example 1. Let $\Sigma = \{a, b\}$, let $\Gamma = \{a\}$, and let $u = v = aba$. Then $u \sqcup\!\sqcup\,^S_\Gamma\, v = \{abba\}$, $u \sqcup\!\sqcup\,^W_\Gamma\, v = (u \sqcup\!\sqcup\,^S_\Gamma\, v) \cup \{ababa\}$, and $u \sqcup\!\sqcup\,^A_\Gamma\, v = (u \sqcup\!\sqcup\,^W_\Gamma\, v) \cup \{abbaa, aabba, aababa, aabbaa, abaaba, ababaa\}$.

As this example shows, the three synchronized shuffle operations applied to words u and v may yield languages of increasing size, even if $\text{pres}_\Gamma(u) = \text{pres}_\Gamma(v)$.

We now look at the relation between the three synchronized shuffles and the basic shuffle. To begin with, each synchronized shuffle is indeed a generalization of the shuffle: $u \sqcup\!\sqcup\,^X_\emptyset\, v = u \sqcup\!\sqcup\, v$, for all $u, v \in \Sigma^*$, and $X \in \{S, W, A\}$.

Next, let $u, v \in \Sigma^*$ and let $\Gamma \subseteq \Sigma$ be such that $\alpha(u) \cap \alpha(v) \subseteq \Gamma$. Then

$$u \sqcup\!\sqcup\,^S_\Gamma\, v = (u \sqcup\!\sqcup\, \text{pres}_{\Sigma\setminus\Gamma}(v)) \cap (\text{pres}_{\Sigma\setminus\Gamma}(u) \sqcup\!\sqcup\, v).$$

The condition $\alpha(u) \cap \alpha(v) \subseteq \Gamma$ is necessary. This can be concluded from Example 1, where we see that $(u \sqcup\!\sqcup\, \text{pres}_{\Sigma\setminus\Gamma}(v)) \cap (\text{pres}_{\Sigma\setminus\Gamma}(u) \sqcup\!\sqcup\, v) = (aba \sqcup\!\sqcup\, b) \cap (b \sqcup\!\sqcup\, aba) = \{abab, abba, baba\} \neq \{abba\} = u \sqcup\!\sqcup\,^S_\Gamma\, v$.

Based on the relation between SS-shuffles and shuffles, we obtain an alternative definition of the SS-shuffle operation in terms of morphisms. Let $u, v \in \Sigma^*$ and let $\Gamma \subseteq \Sigma$ be such that $\alpha(u) \cap \alpha(v) \subseteq \Gamma$. Then

$$u \mathbin{\text{Ш}}{}^{S}_{\Gamma}\, v = \{\, w \in \Sigma^* \mid \mathrm{pres}_{\Sigma}(w) = u,\ \mathrm{pres}_{\Sigma}(w) = v \,\}.$$

Note that condition $\alpha(u) \cap \alpha(v) \subseteq \Gamma$ is satisfied whenever $\Gamma = \Sigma$.

It is worth mentioning that the SS-shuffle on Γ of u and v equals the rS-shuffle (S-shuffle) on Γ of u and v as defined in [2,4]. Moreover, if $\Gamma = \Sigma$, then the SS-shuffle on Γ of $u, v \in \Sigma^*$ equals the fS-shuffle (S-shuffle) on Γ of u and v as defined in [2,4] as well as the weave of u and v as defined in [22]. To the best of our knowledge, however, the WS-shuffle and the AS-shuffle are new types of synchronized shuffles.

Recall that the result of the shuffle of an arbitrary word and the empty word consists of the arbitrary word only, i.e. the shuffle operation has unit element λ. Due to the requirement of a matching backbone, we immediately conclude that this in general does not hold when the SS-shuffle rather than the shuffle is considered. In fact, for an alphabet Γ and a word u, we note that $u \mathbin{\text{Ш}}{}^{S}_{\Gamma}\, \lambda = \lambda \mathbin{\text{Ш}}{}^{S}_{\Gamma}\, u$ equals $\{u\}$ if and only if $\Gamma \cap \alpha(u) = \emptyset$ or $\Gamma = \emptyset$. On the other hand, $u \mathbin{\text{Ш}}{}^{X}_{\Gamma}\, \lambda = \lambda \mathbin{\text{Ш}}{}^{X}_{\Gamma}\, u = \{u\}$, for any word u and $X \in \{W, A\}$.

We now discuss the associativity of the three synchronized shuffles that we introduced. We start by extending the operations defined above to languages as follows. Let $X \in \{S, W, A\}$. The XS-shuffle on $\Gamma \subseteq \Sigma$ of languages $L_1, L_2 \subseteq \Sigma^*$ is denoted by $L_1 \mathbin{\text{Ш}}{}^{X}_{\Gamma}\, L_2$ and is defined as the language consisting of all words that are an XS-shuffle on Γ of a word from L_1 and a word from L_2. Hence

$$L_1 \mathbin{\text{Ш}}{}^{X}_{\Gamma}\, L_2 = \bigcup_{u \in L_1,\, v \in L_2} u \mathbin{\text{Ш}}{}^{X}_{\Gamma}\, v.$$

Proposition 2. *Let Σ be an alphabet and let $\Gamma \subseteq \Sigma$ have at least two letters.*
1. *The SS-shuffle and the AS-shuffle are commutative and associative.*
2. *The WS-shuffle is commutative, but not associative.*

Proof. 1. The commutativity of all three shuffles follows easily from the definitions and the commutativity of the basic shuffle. As the backbone of every word in the SS-shuffle on a given alphabet of two words is the same word, inherited from the two words, and the basic shuffle is associative, it follows that the SS-shuffle is associative.

Let $x, y, z \in \Sigma^*$ and let $\Gamma \subseteq \Sigma$. Since $\mathbin{\text{Ш}}{}^{A}_{\Gamma}$ is commutative, the inclusion $((x \mathbin{\text{Ш}}{}^{A}_{\Gamma}\, y) \mathbin{\text{Ш}}{}^{A}_{\Gamma}\, z) \subseteq (x \mathbin{\text{Ш}}{}^{A}_{\Gamma}\, (y \mathbin{\text{Ш}}{}^{A}_{\Gamma}\, z))$ is sufficient for proving the associativity of the AS-shuffle. We associate with any word $w \in ((x \mathbin{\text{Ш}}{}^{A}_{\Gamma}\, y) \mathbin{\text{Ш}}{}^{A}_{\Gamma}\, z)$ the word

$$w' = \langle a_1, 1_1, 2_1, 3_1\rangle \langle a_2, 1_2, 2_2, 3_2\rangle \cdots \langle a_n, 1_n, 2_n, 3_n\rangle$$

with $i_j \in \{0, 1\}$, $i \in [3]$, and $j \in [n]$, such that the following conditions are satisfied:
(i) $w = a_1 a_2 \cdots a_n$ and
(ii) $x = h_1(w'),\ y = h_2(w')$, and $z = h_3(w')$, where the morphisms h_i, with $i \in [3]$, are defined by

$$h_i(\langle a_j, 1_j, 2_j, 3_j\rangle) = \begin{cases} a_j & \text{if } i_j = 1, \\ \lambda & \text{otherwise.} \end{cases}$$

Let us assume $w \in u \sqcup\!\sqcup_{\Gamma}^{A} z$, for some $u \in x \sqcup\!\sqcup_{\Gamma}^{A} y$. Informally, the occurrence of $\langle a_j, 1, 0, 0\rangle$, $\langle a_j, 0, 1, 0\rangle$, or $\langle a_j, 0, 0, 1\rangle$ on the jth position of w' means that the occurrence of a_j on the jth position of w comes directly from an occurrence of a_j in x, y, or z, respectively. The occurrence of $\langle a_j, 1, 1, 0\rangle$ on the jth position of w' means that the occurrence of a_j on the jth position of w comes from the synchronization of x and y on an occurrence of a_j in both words, while u and z do not synchronize on this occurrence of a_j. The occurrence of $\langle a_j, 1, 0, 1\rangle$ on the jth position of w' means that the occurrence of a_j on the jth position of w comes from the synchronization of u and z on an occurrence of a_j in both words, but this occurrence of a_j in u comes directly from x. An analogous meaning is associated with each occurrence of a letter $\langle a_j, 0, 1, 1\rangle$. Finally, the occurrence of $\langle a_j, 1, 1, 1\rangle$ on the jth position of w' means that the occurrence of a_j on the jth position of w comes from the synchronization of x and y on an occurrence of a_j in both words, while u and z do further synchronize on this occurrence of a_j.

More formally, the word w' can be constructed in two phases as follows:

(A) We associate with $u \in (x_1 \sqcup\!\sqcup y_1)v_1(x_2 \sqcup\!\sqcup y_2)v_2 \cdots v_{p-1}(x_p \sqcup\!\sqcup y_p)$, with $p \geqslant 1$, $v_1, v_2, \ldots, v_{p-1} \in \Gamma$, $x_1v_1x_2v_2 \cdots v_{p-1}x_p = x$, and $y_1v_1y_2v_2 \cdots v_{p-1}y_p = y$, the word u' constructed from u as follows:

(i) each letter a of all x_i, with $i \in [p]$, is replaced by $\langle a, 1, 0, 0\rangle$,
(ii) each letter a of all y_i, with $i \in [p]$, is replaced by $\langle a, 0, 1, 0\rangle$, and
(iii) each letter v_i, with $i \in [p]$, is replaced by $\langle v_i, 1, 1, 0\rangle$.

(B) We associate with $w \in (u_1 \sqcup\!\sqcup z_1)t_1(u_2 \sqcup\!\sqcup z_2)t_2 \cdots t_{m-1}(u_m \sqcup\!\sqcup z_m)$, with $m \geqslant 1$, $t_1, t_2, \ldots, t_{m-1} \in \Gamma$, $u_1t_1u_2t_2 \cdots t_{n-1}u_n = u$, and $z_1t_1z_2t_2 \cdots t_{n-1}z_n = z$, the word w' constructed from w as follows:

(i) each letter a of all z_i, with $i \in [m]$, is replaced by $\langle a, 0, 0, 1\rangle$,
(ii) each occurrence of any letter a of all u_i, with $i \in [m]$, is replaced by the letter of u' associated with this occurrence, and
(iii) each letter t_i, with $i \in [m]$, is replaced by
$\langle t_i, 1, 0, 1\rangle$ if the letter of u' associated with this occurrence of t_i in u is $\langle t_i, 1, 0, 0\rangle$,
$\langle t_i, 0, 1, 1\rangle$ if the letter of u' associated with this occurrence of t_i in u is $\langle t_i, 0, 1, 0\rangle$, and
$\langle t_i, 1, 1, 1\rangle$ if the letter of u' associated with this occurrence of t_i in u is $\langle t_i, 1, 1, 0\rangle$.

From these explanations we infer that $h_{(1,2)}(w') \in x \sqcup\!\sqcup_{\Gamma}^{A} y$, $h_{(1,3)}(w') \in x \sqcup\!\sqcup_{\Gamma}^{A} z$, and $h_{(2,3)}(w') \in y \sqcup\!\sqcup_{\Gamma}^{A} z$, where the morphisms $h_{(i,j)}$, with $i, j \in [3]$ and $i \neq j$, are defined by

$$h_{(i,j)}(\langle a_k, 1_k, 2_k, 3_k\rangle) = \begin{cases} a_k & \text{if } i_k \vee j_k = 1, \\ \lambda & \text{otherwise.} \end{cases}$$

Consequently, $w \in (x \sqcup\!\sqcup_{\Gamma}^{A} (y \sqcup\!\sqcup_{\Gamma}^{A} z))$.

2. The WS-shuffle is not associative. Let $\Sigma = \Gamma = \{a, b\}$ and consider the three words $x = ab$, $y = ba$, and $z = ba$. One can easily verify by direct calculus that $baba \in ((x \sqcup\!\sqcup_{\Gamma}^{W} y) \sqcup\!\sqcup_{\Gamma}^{W} z) \setminus (x \sqcup\!\sqcup_{\Gamma}^{W} (y \sqcup\!\sqcup_{\Gamma}^{W} z))$. $\square$

4. Synchronized shuffles on languages

In this section we present a few properties of some families of languages with respect to the (synchronized) shuffle operations defined above.

We write $⧢^X_\Gamma(L)$ instead of $L ⧢^X_\Gamma L$. Let $L \subseteq \Sigma^*$ be a language and let $\Gamma \subseteq \Sigma$. Then we say that L is *XS-shuffle closed* w.r.t. Γ if $⧢^X_\Gamma(L) \subseteq L$. We simply say that L is *XS-shuffle closed* if it is XS-shuffle closed w.r.t. $\alpha(L)$. Note that any language L such that $\alpha(x) = \alpha(L)$, for all $x \in L$, is SS-shuffle closed; more precisely, $⧢^S_{\alpha(L)}(L) = L$.

All three synchronized shuffles are distributive over the union. Hence

Proposition 3. 1. *For any $\Gamma \subseteq \Sigma$, $(\mathcal{P}(\Sigma^*), \cup, ⧢^S_\Gamma, \emptyset)$ is a commutative hemiring* (*i.e. a semiring without a unit element*).

2. *For any $\Gamma \subseteq \Sigma$, $(\mathcal{P}(\Sigma^*), \cup, ⧢^A_\Gamma, \emptyset, \lambda)$ is a commutative semiring.*

Let $X \in \{S, W, A\}$. A family of languages $\mathcal{F}$ is said to be *fully closed* under XS-shuffle if for any two languages $L_1, L_2 \in \mathcal{F}$ and any $\Gamma \subseteq \alpha(L_1) \cap \alpha(L_2)$, $L_1 ⧢^X_\Gamma L_2 \in \mathcal{F}$. Moreover, $\mathcal{F}$ is said to be *closed* under XS-shuffle if for any two languages $L_1, L_2 \in \mathcal{F}$, $L_1 ⧢^X_{\alpha(L_1)\cap\alpha(L_2)} L_2 \in \mathcal{F}$.

A family of languages which is closed under (non-erasing) morphisms, inverse morphisms, and intersection with regular languages is called a *full trio* (*trio*). By the aforementioned considerations, any family that is fully closed under XS-shuffle, with $X \in \{S, W, A\}$, is closed under shuffle as well. By the next three propositions, we show that in a trio the closure under any of the shuffles defined in this article implies the closure under all the others.

Proposition 4. *Every trio is closed under shuffle if and only if it is fully closed under SS-shuffle if and only if it is closed under SS-shuffle.*

Proof. We first prove the full closure of the trio $\mathcal{F}$ under SS-shuffle, provided that $\mathcal{F}$ is closed under shuffle. Let $L_1, L_2 \subseteq \Sigma^*$ be two languages in $\mathcal{F}$ and let $\Gamma \subseteq \Sigma$. We define the new alphabet $\widehat{\Sigma} = \{\widehat{a} \mid a \in \Sigma\}$ and consider the morphism $h : \Sigma^* \longrightarrow \widehat{\Sigma}^*,\ h(a) = \widehat{a},\ a \in \Sigma$. Let M be a sequential transducer which defines a finite transduction from $(\Sigma \cup \widehat{\Sigma})^*$ onto Σ^* and works as follows:

1. M verifies the following two conditions to be satisfied:
   (i) each occurrence of a letter a from Γ in the input word w is immediately followed by $\widehat{a}$, and
   (ii) each occurrence of a letter $\widehat{a}$, with a from Γ, in the input word w is immediately preceded by a.
2. When reading any subword $a\widehat{a}$, with $a \in \Gamma$, M outputs a.
3. When reading any b or $\widehat{b}$, with $b \notin \Gamma$, M outputs b.

By these explanations,

$$L_1 ⧢^S_\Gamma L_2 = T_M(L_1 ⧢ h(L_2)).$$

Since any trio is closed under transducer mappings, it follows that $\mathcal{F}$ is fully closed under SS-shuffle.

Clearly, any family fully closed under SS-shuffle is closed under SS-shuffle. It remains to prove the closure of $\mathcal{F}$ under shuffle, provided that $\mathcal{F}$ is closed under SS-shuffle. To this end, for two languages $L_1, L_2 \subseteq \Sigma^*$ in $\mathcal{F}$ we define the same alphabet $\widehat{\Sigma}$ and morphism h as above. Furthermore, we consider the morphism $f : (\Sigma \cup \widehat{\Sigma})^* \longrightarrow \Sigma^*,\ f(a) = a,\ f(\widehat{a}) = a$, for all $a \in \Sigma$. It is plain that $\Gamma = \alpha(L_1) \cap \alpha(h(L_2)) = \emptyset$. Therefore

$$L_1 \sqcup\!\sqcup\, L_2 = f(L_1 \sqcup\!\sqcup_{\Gamma}^{S} h(L_2)),$$

which concludes the proof. □

It is worth mentioning that the above result holds for families of languages having only two out of the three closure properties of a trio, namely they are closed under morphisms and inverse morphisms.

Proposition 5. *Every trio is closed under shuffle if and only if it is fully closed under WS-shuffle if and only if it is closed under WS-shuffle.*

Proof. We first prove the full closure of the trio $\mathcal{F}$ under WS-shuffle, provided that $\mathcal{F}$ is closed under shuffle. Let $L_1, L_2 \subseteq \Sigma^*$ be two languages in $\mathcal{F}$ and let $\Gamma \subseteq \Sigma$. We define the new alphabets $\widehat{\Sigma} = \{\widehat{a} \mid a \in \Sigma\}$ and $\overline{\Sigma} = \{\overline{a} \mid a \in \Sigma\}$. Let the finite substitution $s : \Sigma^* \longrightarrow \mathcal{P}((\widehat{\Sigma} \cup \overline{\Sigma})^*)$ be defined by $s(a) = \{\overline{a}\}$, for any $a \in \Sigma \setminus \Gamma$, and $s(a) = \{\overline{a}, \widehat{a}\}$, for any $a \in \Gamma$. Now we construct a sequential transducer M which reads words from $(\Sigma \cup \widehat{\Sigma} \cup \overline{\Sigma})^*$ and outputs words in Σ^*. It works iteratively in two phases as follows:

1. M scans the input word until either an occurrence of $\widehat{a}$, for some $a \in \Gamma$, is encountered or the input word is completely read. Along this computation, when reading a letter b from Σ and $\overline{c}$ from $\overline{\Sigma}$, M writes b and c, respectively. However, if the currently scanned subword of the input word contains a pair of letters $(c, \overline{c})$, for some $c \in \Gamma$, then M enters a designated error state and blocks the computation. This checking process can be done by storing the letters from $\Gamma \cup \overline{\Gamma}$ read so far in the current state.
2. When $\widehat{a}$ is reached, the second phase starts. M enters a new state and, without writing anything, verifies whether the next input symbol is exactly a. If this is not the case, then M enters the error state and blocks the computation. Otherwise, M enters the initial state and writes one a only. Now the first phase is resumed for the rest of the input word.
3. All states are final, except the error state and the states of the second phase.

From these explanations,

$$L_1 \sqcup\!\sqcup_{\Gamma}^{W} L_2 = T_M(L_1 \sqcup\!\sqcup\, s(L_2)).$$

Since any trio is closed under finite substitutions and transducer mappings, it follows that $\mathcal{F}$ is closed under WS-shuffle. The final part of the previous proof works well for proving the closure of $\mathcal{F}$ under shuffle, provided that $\mathcal{F}$ is closed under WS-shuffle. □

Obviously, a similar construction as above holds for the AS-shuffle. Hence

Proposition 6. *Every trio is closed under shuffle if and only if it is fully closed under AS-shuffle if and only if it is closed under AS-shuffle.*

Based on the above results we can now present a procedure for deciding whether a regular language is XS-shuffle closed w.r.t. any given alphabet of letters subject to synchronization.

Proposition 7. *For any regular language $L \subseteq \Sigma^*$, any $\Gamma \subseteq \Sigma$, and any $X \in \{S, W, A\}$, one can algorithmically decide whether L is XS-shuffle closed w.r.t. Γ.*

Proof. Since the family of regular languages is a trio closed under shuffle, $⧢^X_\Gamma(L)$ is still regular for any regular language $L \subseteq \Sigma^*$, any $\Gamma \subseteq \Sigma$, and any $X \in \{S, W, A\}$. Moreover, given a finite automaton which accepts L one can effectively construct a finite automaton which recognizes $⧢^X_\Gamma(L)$. On the other hand, the inclusion problem is decidable for regular languages; therefore, one can algorithmically decide whether $⧢^X_\Gamma(L) \subseteq L$. □

Proposition 8. *Let $\mathcal{F}$ be an arbitrary family of languages.*
1. *If $\mathcal{F}$ is (fully) closed under SS-shuffle, then $\mathcal{F}$ is closed under intersection.*
2. *If $\mathcal{F}$ is closed under morphisms, inverse morphisms, and (fully) closed under XS-shuffle, with $X \in \{W, A\}$, then $\mathcal{F}$ is closed under intersection.*

Proof. 1. Let $L_1, L_2 \subseteq \Sigma^*$ be two languages in $\mathcal{F}$. Clearly,

$$L_1 \cap L_2 = L_1 ⧢^S_{\alpha(L_1)\cap\alpha(L_2)} L_2.$$

2. As $\mathcal{F}$ is (fully) closed under XS-shuffle, for any $X \in \{W, A\}$, it is closed under shuffle, a property which together with the closure under morphisms and inverse morphisms implies the closure of $\mathcal{F}$ under intersection. □

We now present a characterization of the family of nonempty finite languages with a single binary generator involving the synchronized shuffles.

Proposition 9. *The family of nonempty finite languages is the smallest family containing the language $\{ab\}$, closed under union, closed under weak literal morphisms, and (fully) closed under any synchronized shuffle.*

Proof. Clearly the smallest family containing the language $\{ab\}$, closed under union, closed under weak literal morphisms, and (fully) closed under any synchronized shuffle, denoted by $\mathcal{F}$, contains finite languages only.

Conversely, the languages $\{\lambda\}$ and $\{a\}$, for any letter a, belong to $\mathcal{F}$ due to its closure under weak literal morphisms. The closure properties of $\mathcal{F}$ imply that it suffices to prove that the singleton language consisting of an arbitrary word $w = a_1a_2\cdots a_n$, with $a_i \neq a_j$ and $1 \leqslant i \neq j \leqslant n$, lies in $\mathcal{F}$. We prove this by induction on n. Obviously, the assertion holds for $n \in \{0, 1, 2\}$. For any $n \geqslant 3$ and $X \in \{S, W, A\}$, we have $\{w\} = \{a_1a_2\cdots a_{n-1}\} ⧢^X_{\{a_{n-1}\}} \{a_{n-1}a_n\}$. □

5. Conclusion

In this article we have studied synchronized shuffles on words and languages. In the introduction we have already hinted at their relevance to molecular biology by viewing them as the formal representations of varieties of gene linkage during genome shuffling. This is a topic worth further investigation. Here we elaborate on their usefulness in other areas. To begin with, synchronized shuffles are useful when proving compositionality of automata-based specification models such as team automata.

Team automata form a flexible framework for modelling collaboration between system components [2,3,14]. The crux of composing a team automaton is to define the way in which its constituting automata communicate through synchronizations of shared actions. A number of fixed strategies for composing team automata were defined in [3]. Consequently, the conditions under which these strategies lead to team automata satisfying compositionality were studied in [4]. To this end, a synchronized shuffle and two special cases, viz. the relaxed synchronized and the full synchronized shuffle, were defined. It was shown that some fixed strategies lead to team automata satisfying compositionality, in the sense that the behaviour of a certain team automaton equals a certain synchronized shuffle of the behaviour of its constituting automata. For some types of team automata, compositionality was thus proved by using synchronized shuffles.

In order to identify more types of team automata satisfying compositionality, it is necessary to establish precise conditions under which the behaviour of team automata defined according to other fixed strategies can be obtained from that of their constituting automata. This calls for more types of synchronized shuffles than those introduced in [4]. It remains to be investigated whether the synchronized shuffles introduced in this article can be used to prove specific team automata to be compositional. This undoubtedly requires these synchronized shuffles to satisfy some of the fundamental mathematical properties studied in this article.

Finally, it also remains to be investigated whether the synchronized shuffles introduced in this article can be used to model some aspects of parallel compositions of concurrent processes that contain re-entrant routines, such as those that are part of the kernel of operating systems like UNIX and LINUX. An algebraic approach to modelling such processes was initiated in [16] and continued in [1].

Acknowledgements

The first author would like to thank Jetty Kleijn for many stimulating and in-depth discussions on synchronized shuffles. The authors thank the anonymous referees for their useful suggestions that have improved the article.

References

[1] J.C.M. Baeten, W.P. Weijland, Process Algebra, Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, Cambridge, 1990.

[2] M.H. ter Beek, Team Automata—A Formal Approach to the Modeling of Collaboration Between System Components, Ph.D. Thesis, Leiden Institute of Advanced Computer Science, Leiden University, 2003.

[3] M.H. ter Beek, C.A. Ellis, J. Kleijn, G. Rozenberg, Synchronizations in team automata for groupware systems, Comput. Supported Cooperative Work—J. Collaborative Comput. 12 (1) (2003) 21–69.

[4] M.H. ter Beek, J. Kleijn, Team automata satisfying compositionality, in: K. Araki, S. Gnesi, D. Mandrioli (Eds.), Proc. of FME 2003: Formal Methods—the 12th Internat. Symp. on Formal Methods Europe, Pisa, Italy, Lecture Notes in Computer Science, Vol. 2805, Springer, Berlin, 2003, pp. 381–400.

[5] J.A. Bergstra, A. Ponse, S.A. Smolka (Eds.), Handbook of Process Algebra, Elsevier Science Publishers, Amsterdam, 2001.

[6] S.L. Bloom, Z. Ésik, Free shuffle algebras in language varieties, Theoret. Comput. Sci. 163 (1996) 55–98.

[7] P.C.Y. Chen, A computational model of a class of gene networks with positive and negative controls, BioSystems 73 (2004) 13–24.
[8] R. De Simone, Langages infinitaires et produit de mixage, Theoret. Comput. Sci. 31 (1984) 83–100.
[9] S. Eilenberg, Automata, Languages, and Machines, Vol. A. Academic Press, New York, 1974.
[10] S. Ginsburg, E.H. Spanier, Mappings of languages by two-tape devices, J. ACM 12 (3) (1965) 423–434.
[11] J.L. Gischer, Shuffle languages, Petri nets, and context sensitive grammars, Comm. ACM 24 (1981) 597–605.
[12] M. Jantzen, The power of synchronizing operations on strings, Theoret. Comput. Sci. 14 (1981) 127–154.
[13] T. Kimura, An algebraic system for process structuring and interprocess communication, in: Proc. of the 8th ACM SIGACT Symp. on Theory of Computing, Hershey, Pennsylvania, ACM Press, New York, 1976, pp. 92–100.
[14] J. Kleijn, Team automata for CSCW—a survey, in: H. Ehrig, W. Reisig, G. Rozenberg, H. Weber (Eds.), Petri Net Technology for Communication-Based Systems—Advances in Petri Nets, Lecture Notes in Computer Science, Vol. 2472, Springer, Berlin, 2003, pp. 295–320.
[15] M. Latteux, Y. Roos, Synchronized shuffle and regular languages, in: J. Karhumäki, H.A. Maurer, Gh. Păun, G. Rozenberg (Eds.), Jewels are Forever—Contributions on Theoretical Computer Science in Honor of Arto Salomaa, Springer, Berlin, 1999, pp. 35–44.
[16] R. Milner, A Calculus of Communicating Systems, Lecture Notes in Computer Science, Vol. 92, Springer, Berlin, 1980.
[17] D. Park, On the semantics of fair parallelism, in: D. Bjørner (Ed.), Proc. of the Copenhagen Winter School on Abstract Software Specifications, Lecture Notes in Computer Science, Vol. 86, Springer, Berlin, 1979, pp. 504–526.
[18] A.W. Roscoe, The Theory and Practice of Concurrency, Prentice-Hall International Series in Computer Science, London, 1997.
[19] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Springer, Berlin, 1997.
[20] R. Schaffrath, K. Sasnauskas, P.A. Meacock, Use of gene shuffles to study the cytoplasmic transcription system operating on Kluyveromyces lactis linear DNA plasmids, Enzyme Microbial Technol. 26 (9–10) (2000) 664–670.
[21] A.C. Shaw, Software descriptions with flow expressions, IEEE Trans. Software Engng. SE-4 (3) (1978) 242 –254.
[22] J. van de Snepscheut, Trace Theory and VLSI Design, Lecture Notes in Computer Science, Vol. 200, Springer, Berlin, 1985.
[23] W. Wechler, Universal Algebra for Computer Scientists, EATCS Monographs on Theoretical Computer Science, Vol. 25, Springer, Berlin, 1992.
[24] Y.-X. Zhang, K. Perry, V.A. Vinci, K. Powell, W.P.C. Stemmer, S.B. del Cardayré, Genome shuffling leads to rapid phenotypic improvement in bacteria, Nature 415 (2002) 644–646.