Synchronized Shuffles

Maurice H. ter Beek[1], Carlos Martín-Vide[2], and Victor Mitrana[2,3]

[1] Istituto di Scienza e Tecnologie dell'Informazione
Consiglio Nazionale delle Ricerche,
Area della Ricerca di Pisa, Via G. Moruzzi 1, 56124 Pisa, Italy
maurice.terbeek@isti.cnr.it

[2] Research Group in Mathematical Linguistics
Rovira i Virgili University
Pça. Imperial Tarraco 1, 43005 Tarragona, Spain
{cmv,vmi}@correu.urv.es

[3] Faculty of Mathematics and Computer Science
University of Bucharest
Str. Academiei 14, 70109 Bucharest, Romania

Abstract. We introduce and investigate three synchronized shuffle operations on words and languages. We show how such operations come into play when proving compositionality of automata-based specification models, by demonstrating their usage in the context of team automata. As far as the mathematical properties of these operations are concerned, we prove that in a trio the closure under shuffle is equivalent to the closure under any of the synchronized shuffles studied in this paper. Finally, based on this result, we present an algorithm for deciding whether or not a given regular language is synchronized shuffle closed.

1 Introduction

We introduce and investigate three synchronized shuffle operations on words and languages. Before doing so, we elaborate on the usefulness of such operations in a setting which is relevant to practice, viz. when proving compositionality of automata-based specification models like team automata.

Component-based system design is a complex task that benefits from a stepwise development. This means that an abstract high-level specification of a design is decomposed into a more concrete low-level specification by step-by-step refinement, at each step replacing components of the current specification by more detailed ones. To guarantee correct decompositions it is of the utmost importance that the chosen specification model is compositional. This means that a specification of a composite system can be obtained from specifications of its components [15]. In case of automata-based specification models, compositionality thus requires that the relevant behaviour of a composite automaton can be obtained from the behaviour of its constituting automata.

Most automata-based specification models guarantee compositionality by choosing a single and very strict method of composing automata, in effect resulting in composite automata that are uniquely defined by their constituents. The

choice prevalent in the literature is to allow the execution of an action in a composite automaton if and only if all of its constituting automata that share this action simultaneously execute it. In [4] this type of synchronization of shared actions is coined *maximal-action-indispensable* (*maximal-ai* for short). Examples of automata-based specification models with composition based on *maximal-ai* synchronizations include I/O automata [19], I/O systems [15], timed cooperating automata [17], and reactive transition systems [8].

Team automata were introduced for the specification of groupware systems and their interconnections, but they were shown to provide a flexible framework for modelling collaboration between components of reactive systems in general [11,3,4,2]. A team automaton is composed of component automata, which are automata with a partition of their sets of actions into input, output, and internal actions. Input actions are not under the automaton's control, but instead are triggered by output actions of the environment consisting of other components. Output and internal actions are under its control, but only output actions are observable by the environment. Input and output actions together constitute the external actions and form the interface between the automaton and its environment, whereas internal actions are not available for interactions.

The crux of composing a team automaton is to define the way in which its constituting component automata interact through synchronizations of shared actions. Whereas all of the aforementioned specification models compose the originally independent automata in a unique way, there is no such thing as the unique team automaton. Rather, a whole range of team automata, distinguishable only by their synchronizations, can be composed over a set of component automata. A team automaton is determined on the basis of its components by choosing synchronizations reflecting the specific protocol of collaboration to be modelled. This freedom offers the flexibility to distinguish even the smallest nuances in the meaning of a design and thus sets this approach apart from most other automata-based specification models from the literature.

In [4] a variety of fixed strategies for composing team automata, all leading to uniquely defined team automata, was introduced. These strategies are based on the basic types of synchronization *maximal-ai*, *maximal-free*, *maximal-state-indispensable*, and on the more complex types of synchronization *maximal-peer-to-peer* and *maximal-master-slave* involving the role of actions. In [5], consequently, the conditions under which these strategies lead to team automata satisfying compositionality were investigated. In particular, the relation between the behaviour of team automata defined according to the *maximal-ai* and *maximal-free* strategies and the behaviour of their constituting components is studied in detail. This required establishing which combinations of words—if any—from the behaviour of the components can be combined—and in particular how—such that a word from the behaviour of the particular team automaton composed over those components results. To this aim, a synchronized shuffle (S-shuffle for short) and two special cases of this S-shuffle, viz. the *relaxed synchronized shuffle* (rS-shuffle for short) and the *fully synchronized shuffle* (fS-shuffle for short), were introduced.

In [5] it is shown that some of the aforementioned fixed strategies lead to team automata satisfying compositionality. For one, the behaviour of a team automaton composed according to the *maximal-ai* strategy equals the fS-shuffle of the behaviour of its constituting component automata. Furthermore, the behaviour of a team automaton composed according to a mixture of the *maximal-free* and *maximal-ai* strategies—under some conditions—equals the rS-shuffle of the behaviour of its constituting components. Hence, for a few specific types of team automata, compositionality is proved through the use of synchronized shuffle operations. In [2], finally, it is shown that corresponding results hold when also the infinitary behaviour of team automata is taken into account.

In order to identify more types of team automata satisfying compositionality than those investigated in [5], it is necessary to establish the precise conditions under which the behaviour of team automata defined according to other fixed strategies can be obtained from the behaviour of their constituting component automata. This calls for more types of synchronized shuffles than those introduced in [5]. This is the main goal of this paper. To this aim, we introduce and investigate three synchronized shuffle operations: the *strongly synchronized shuffle* (SS-shuffle for short), the *weakly synchronized shuffle* (WS-shuffle for short), and the *arbitrarily synchronized shuffle* (AS-shuffle for short). While there exist conditions under which the SS-shuffle degenerates to the rS-shuffle or the fS-shuffle (and thus to the S-shuffle), both the WS-shuffle and the AS-shuffle are—to the best of our knowledge—new types of synchronized shuffles.

This paper is organized as follows. First we formally define the three aforementioned synchronized shuffle operations and compare them to other synchronized shuffle operations from the literature. Consequently, we show that in a trio the closure under shuffle is equivalent to the closure under any of these synchronized shuffle operations. Finally, based on this result, we give an algorithm for deciding whether or not a given regular language is synchronized shuffle closed.

2 Preliminaries

We assume familiarity with some basic notions from algebra and formal language theory. For any unexplained notion, we refer the reader to [26, 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 $\alpha(\lambda) = \varnothing$.

The function pres_Γ, also called the projection on Γ, is a morphism from Σ^* into Γ^* defined by $\mathrm{pres}_\Gamma(a) = a$ if $a \in \Gamma$ and $\mathrm{pres}_\Gamma(a) = \lambda$ otherwise. In other words, pres_Γ preserves the symbols from Γ and erases all other symbols. By convention, $\mathrm{pres}_\varnothing(v) = \lambda$, 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* [10]. The transition function may be extended in a natural way to $Q \times V^*$. Each finite transducer M as above defines a *finite transduction*

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

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 the original words such that it contains all symbols of both words, like the shuffling of two decks of cards. This is a well-known language-theoretic operation with a long history in theoretical computer science, in particular within formal language theory. In the literature shuffling is sometimes called interleaving, weaving, or merging, and—given two words u and v—it may be denoted by $u \odot v$, $u \,||\, v$, $u \sqcup\!\sqcup v$, $u \square v$, $u \otimes v$, $u \,|||\, v$, or $u \diamond v$ [12, 24, 21, 14, 13, 7, 23]. The idea underlying shuffling also appears in numerous other disguises throughout the computer science literature. Within concurrency theory, e.g., as a semantics of parallel operators modelling communication between processes [1, 22, 6].

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

$$\begin{aligned} x \sqcup\!\sqcup \lambda = \lambda \sqcup\!\sqcup x = \{x\},\ x \in \Sigma^*, \text{ and} \\ ax \sqcup\!\sqcup by = a(x \sqcup\!\sqcup by) \cup b(ax \sqcup\!\sqcup y),\ a, b \in \Sigma,\ x, y \in \Sigma^*. \end{aligned}$$

The shuffle of two languages $L_1, L_2 \subseteq \Sigma^*$ is denoted by $L_1 \sqcup\!\sqcup L_2$ and is defined as the 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 $u \sqcup\!\sqcup v \neq \varnothing$. 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 now continue our exposition on shuffle operations by introducing some more intriguing types of shuffles, built on top of the basic shuffle operation defined so far. We generalize the basic shuffle operation recalled above by defining three synchronized shuffle operations. 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 shuffle operations is not new, but it appears in numerous disguises throughout the computer science literature. The idea seems to stem from the concurrent composition $P \oplus Q$ of synchronizing processes P and Q as defined in [16]. Within formal language theory, a slightly adapted version of the idea was introduced in [9] as the 'produit de mixage' $u \sqcap\!\sqcap v$ of words u and v. This operation was renamed synchronized shuffle in [18] and in [5] 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 [5] also two special cases of the S-shuffle were 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 fully 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. 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 [25] as the weave $u \,\underline{w}\, v = u \,{}_{\Sigma_1}\underline{\|}_{\Sigma_2}\, v$ of two words u and v, and in [22] as the alphabetized parallel composition $P \,{}_{X}\underline{\|}_{Y}\, Q$ of processes P and Q—given alphabets X and Y.

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

Strongly Synchronized Shuffle Given two words $u \in \Sigma_1^*$ and $v \in \Sigma_2^*$ and a subset Γ of $\Sigma_1 \cap \Sigma_2$, 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 their 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 $u \in \Sigma_1^*$ and $v \in \Sigma_2^*$ on $\Gamma \subseteq \Sigma_1 \cap \Sigma_2$ is denoted by $u \sqcup\!\sqcup^S_\Gamma v$ and is defined as

$$
\begin{aligned}
u \sqcup\!\sqcup^S_\Gamma v = \{\, & (u_1 \sqcup\!\sqcup v_1)x_1(u_2 \sqcup\!\sqcup v_2)x_2 \cdots x_{n-1}(u_n \sqcup\!\sqcup v_n) \mid n \geq 1, \\
& u_i \in (\Sigma_1 \setminus \Gamma)^*,\ v_i \in (\Sigma_2 \setminus \Gamma)^*,\ i \in [n],\ x_1, x_2, \ldots, x_{n-1} \in \Gamma, \\
& u_1 x_1 u_2 x_2 \cdots x_{n-1} u_n = u,\ v_1 x_1 v_2 x_2 \cdots x_{n-1} v_n = v \,\}.
\end{aligned}
$$

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

Weakly Synchronized Shuffle Given two words $u \in \Sigma_1^*$ and $v \in \Sigma_2^*$ and a subset Γ of $\Sigma_1 \cap \Sigma_2$, 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 the 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 $u \in \Sigma_1^*$ and $v \in \Sigma_2^*$ on $\Gamma \subseteq \Sigma_1 \cap \Sigma_2$ is denoted by $u \sqcup\!\sqcup \,{}^{W}_{\Gamma}\, v$ and is defined as

$$\begin{aligned} u \sqcup\!\sqcup \,{}^{W}_{\Gamma}\, v = \{\, &(u_1 \sqcup\!\sqcup v_1)x_1(u_2 \sqcup\!\sqcup v_2)x_2 \cdots x_{n-1}(u_n \sqcup\!\sqcup v_n) \mid n \geq 1, \\ &u_i \in \Sigma_1^*,\ v_i \in \Sigma_2^*,\ \alpha(u_i) \cap \alpha(v_i) \cap \Gamma = \varnothing,\ i \in [n],\ x_1, x_2, \ldots, x_{n-1} \in \Gamma, \\ &u_1 x_1 u_2 x_2 \cdots x_{n-1} u_n = u,\ v_1 x_1 v_2 x_2 \cdots x_{n-1} v_n = v \,\}. \end{aligned}$$

Note that $\mathrm{pres}_\Gamma(u) \neq \mathrm{pres}_\Gamma(v)$ 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 $w = w_1 x_1 w_2 x_2 \cdots x_{n-1} w_n$, $w_i \in u_i \sqcup\!\sqcup v_i$, $u_i \in \Sigma_1^*$, $v_i \in \Sigma_2^*$, $\alpha(u_i) \cap \alpha(v_i) \cap \Gamma = \varnothing$, $i \in [n]$, $x_1, x_2, \ldots, x_{n-1} \in \Gamma$, $u_1 x_1 u_2 x_2 \cdots x_{n-1} u_n = u$, and $v_1 x_1 v_2 x_2 \cdots x_{n-1} v_n = v$. Then $x_1 x_2 \ldots x_{n-1}$ is called the backbone of w w.r.t. the above decomposition. Note that, contrary to the SS-shuffle, different words in $u \sqcup\!\sqcup \,{}^{W}_{\Gamma}\, v$ may have different backbones.

Arbitrarily Synchronized Shuffle Given two words $u \in \Sigma_1^*$ and $v \in \Sigma_2^*$ and a subset Γ of $\Sigma_1 \cap \Sigma_2$, the arbitrarily synchronized shuffle of u and v on Γ requires u and v to synchronize on some occurrences of the letters from Γ, while all the other occurrences together with the letters not appearing in Γ are shuffled.

Formally, the *arbitrarily synchronized shuffle* (*AS-shuffle* for short) of words $u \in \Sigma_1^*$ and $v \in \Sigma_2^*$ on $\Gamma \subseteq \Sigma_1 \cap \Sigma_2$ is denoted by $u \sqcup\!\sqcup \,{}^{A}_{\Gamma}\, v$ and is defined as

$$\begin{aligned} u \sqcup\!\sqcup \,{}^{A}_{\Gamma}\, v = \{\, &(u_1 \sqcup\!\sqcup v_1)x_1(u_2 \sqcup\!\sqcup v_2)x_2 \cdots x_{n-1}(u_n \sqcup\!\sqcup v_n) \mid n \geq 1, \\ &u_i \in \Sigma_1^*,\ v_i \in \Sigma_2^*,\ i \in [n],\ x_1, x_2, \ldots, x_{n-1} \in \Gamma, \\ &u_1 x_1 u_2 x_2 \cdots x_{n-1} u_n = u,\ v_1 x_1 v_2 x_2 \cdots x_{n-1} v_n = v \,\}. \end{aligned}$$

Note that $\mathrm{pres}_\Gamma(u) \neq \mathrm{pres}_\Gamma(v)$ does not imply that $u \sqcup\!\sqcup \,{}^{A}_{\Gamma}\, v = u \sqcup\!\sqcup \,{}^{S}_{\Gamma}\, v$, but $u \sqcup\!\sqcup \,{}^{S}_{\Gamma}\, v \subseteq u \sqcup\!\sqcup \,{}^{A}_{\Gamma}\, v$ always holds. Also $u \sqcup\!\sqcup \,{}^{W}_{\Gamma}\, v \subseteq u \sqcup\!\sqcup \,{}^{A}_{\Gamma}\, v$ holds. As before, let $w \in u \sqcup\!\sqcup \,{}^{A}_{\Gamma}\, v$ be such that $w = w_1 x_1 w_2 x_2 \cdots x_{n-1} w_n$, $w_i \in u_i \sqcup\!\sqcup v_i$, $u_i \in \Sigma_1^*$, $v_i \in \Sigma_2^*$, $\alpha(u_i) \cap \alpha(v_i) \cap \Gamma = \varnothing$, $i \in [n]$, $x_1, x_2, \ldots, x_{n-1} \in \Gamma$, $u_1 x_1 u_2 x_2 \cdots x_{n-1} u_n = u$, and $v_1 x_1 v_2 x_2 \cdots x_{n-1} v_n = v$. Then $x_1 x_2 \ldots x_{n-1}$ is called the backbone of w w.r.t. the above decomposition. Again, different words in $u \sqcup\!\sqcup \,{}^{A}_{\Gamma}\, v$ may have different backbones.

We now take a closer look at the three synchronized shuffle operations introduced above, in particular regarding their relation to the basic shuffle operation. First we note that each synchronized shuffle operation is indeed a generalization of the shuffle operation: $u \sqcup\!\sqcup \overset{X}{\varnothing} v = u \sqcup\!\sqcup v$, for all $u \in \Sigma_1^*$, $v \in \Sigma_2^*$, and $X \in \{S, W, A\}$.

Next we let $u \in \Sigma_1^*$, $v \in \Sigma_2^*$, and $\Gamma \subseteq \Sigma_1 \cap \Sigma_2$ be such that $(\alpha(u) \setminus \Gamma) \cap (\alpha(v) \setminus \Gamma) = \varnothing$. Then

$$u \sqcup\!\sqcup \overset{S}{\Gamma} v = (u \sqcup\!\sqcup \mathrm{pres}_{\Sigma_2 \setminus \Gamma}(v)) \cap (\mathrm{pres}_{\Sigma_1 \setminus \Gamma}(u) \sqcup\!\sqcup v).$$

The condition $(\alpha(u) \setminus \Gamma) \cap (\alpha(v) \setminus \Gamma) = \varnothing$ is necessary. This follows from the following example, where we show that given words $u \in \Sigma_1^*$ and $v \in \Sigma_2^*$ and an alphabet $\Gamma \subseteq \Sigma_1 \cap \Sigma_2$, in general $u \sqcup\!\sqcup \overset{S}{\Gamma} v$ does not equal $(u \sqcup\!\sqcup \mathrm{pres}_{\Sigma_2 \setminus \Gamma}(v)) \cap (\mathrm{pres}_{\Sigma_1 \setminus \Gamma}(u) \sqcup\!\sqcup v)$. We note, however, that $u \sqcup\!\sqcup \overset{S}{\Gamma} v \subseteq (u \sqcup\!\sqcup \mathrm{pres}_{\Sigma_2 \setminus \Gamma}(v)) \cap (\mathrm{pres}_{\Sigma_1 \setminus \Gamma}(u) \sqcup\!\sqcup v)$ always holds.

Example 1 *Let $\Sigma = \{a, b, c\}$, let $\Gamma = \{c\}$, let $u = abc$, and let $v = bca$. Then $(u \sqcup\!\sqcup pres_{\Sigma_2 \setminus \Gamma}(v)) \cap (pres_{\Sigma_1 \setminus \Gamma}(u) \sqcup\!\sqcup v) = (abc \sqcup\!\sqcup ba) \cap (ab \sqcup\!\sqcup bca) = \{abbca, babca, abcba\} \neq \{abbca, babca\} = u \sqcup\!\sqcup \overset{S}{\Gamma} v$.*

Based on the above relation between the SS-shuffle and shuffle operations, we obtain an alternative definition of the SS-shuffle operation in terms of morphisms. Let $u \in \Sigma_1^*$, $v \in \Sigma_2^*$, and $\Gamma \subseteq \Sigma_1 \cap \Sigma_2$ be such that $(\alpha(u) \setminus \Gamma) \cap (\alpha(v) \setminus \Gamma) = \varnothing$. Then

$$u \sqcup\!\sqcup \overset{S}{\Gamma} v = \{\, w \in (\Sigma_1 \cup \Sigma_2)^* \mid \mathrm{pres}_{\Sigma_1}(w) = u,\ \mathrm{pres}_{\Sigma_2}(w) = v \,\}.$$

Note that the condition $(\alpha(u) \setminus \Gamma) \cap (\alpha(v) \setminus \Gamma) = \varnothing$ is satisfied whenever $\Gamma = \Sigma_1 \cap \Sigma_2$.

Let $u \in \Sigma_1^*$ and let $v \in \Sigma_2^*$. If $\Gamma \subseteq \Sigma_1 \cap \Sigma_2$, then the SS-shuffle on Γ of u and v equals the rS-shuffle (S-shuffle) on Γ of u and v as defined in [5]. Hence $u \sqcup\!\sqcup \overset{S}{\Gamma} v = u \underline{\|}^{\Gamma}_{\Sigma_2} v = u \|^{\Gamma} v$ whenever $\Gamma \subseteq \Sigma_1 \cap \Sigma_2$. In its turn, the S-shuffle on a arbitrary alphabet Γ of u and v is a slight generalization of both the concurrent composition as defined in [16], which requires $\Gamma \subseteq \Sigma_1 = \Sigma_2$, and the synchronized shuffle operation as defined in [9, 18], which requires $\Gamma = \alpha(u) \cap \alpha(v)$. If $\Gamma = \Sigma_1 \cap \Sigma_2$, finally, then the SS-shuffle on Γ of u and v equals the fS-shuffle (S-shuffle) on Γ of u and v as defined in [5] as well as the weave of u and v as defined in [25]. Hence $u \sqcup\!\sqcup \overset{S}{\Gamma} v = u \underline{\|}_{\Sigma_2} v = u \|^{\Gamma} v = u \underline{w} v$ whenever $\Gamma = \Sigma_1 \cap \Sigma_2$. To the best of our knowledge, however, the WS-shuffle and the AS-shuffle are new types of synchronized shuffle operations.

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 operation rather than the shuffle operation is considered. In fact, for an alphabet Γ and a word u, we note that $u \sqcup\!\sqcup \overset{S}{\Gamma} \lambda = \lambda \sqcup\!\sqcup \overset{S}{\Gamma} u$ equals $\{u\}$ if and only if $\Gamma \cap \alpha(u) = \varnothing$ or $\Gamma = \varnothing$. On the other hand, $u \sqcup\!\sqcup \overset{X}{\Gamma} \lambda = \lambda \sqcup\!\sqcup \overset{X}{\Gamma} u = \{u\}$, for any word u and $X \in \{W, A\}$.

We now discuss the associativity of the three synchronized shuffle operations we have introduced, restricted to words over the same alphabet and the same alphabet of letters subject to synchronization.

Proposition 1 *1. For any alphabet Σ and $\Gamma \subseteq \Sigma$, $(\Sigma^*, \sqcup\!\sqcup \overset{S}{\Gamma})$ is a commutative semigroup. Moreover, if $\Gamma = \Sigma$, then this semigroup is idempotent.*

2. The WS-shuffle has a unit element, but it is not associative.

3. For any alphabet Σ and $\Gamma \subseteq \Sigma$, $(\Sigma^*, \sqcup\!\sqcup \overset{A}{\Gamma}, \lambda)$ is a commutative monoid.

Proof. 1. The commutativity of all three operations follows easily from the definitions and the commutativity of the basic shuffle operation. Since 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 operation is associative, it follows that the SS-shuffle operation is associative. Furthermore, it is obvious that $u \sqcup\!\sqcup \overset{S}{\Sigma} u = \{u\}$, for any word $u \in \Sigma^*$.

2. We prove that the WS-shuffle operation is not associative. To this aim, let $\Sigma = \Gamma = \{a, b, c\}$ and consider the three words $x = acb$, $y = bca$, and $z = abc$. One can easily check by direct calculus that $bacbabc \in ((x \sqcup\!\sqcup \overset{W}{\Gamma} y) \sqcup\!\sqcup \overset{W}{\Gamma} z) \setminus (x \sqcup\!\sqcup \overset{W}{\Gamma} (y \sqcup\!\sqcup \overset{W}{\Gamma} z))$.

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

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

Let us assume that $w \in u \sqcup\!\sqcup \overset{A}{\Gamma} z$, for some $u \in x \sqcup\!\sqcup \overset{A}{\Gamma} y$. 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 $u \in (x_1 \sqcup\!\sqcup y_1)v_1(x_2 \sqcup\!\sqcup y_2)v_2 \cdots v_{n-1}(x_n \sqcup\!\sqcup y_n)$, with $n \geq 1$, $v_1, v_2, \ldots, v_{n-1} \in \Gamma$, $x_1 v_1 x_2 v_2 \cdots v_{n-1} x_n = x$, and $y_1 v_1 y_2 v_2 \cdots v_{n-1} y_n = y$, the word u' constructed from u as follows:

- (*i*) each letter a of all x_i, with $i \in [n]$, is replaced by $\langle a, 1, 0, 0\rangle$,
- (*ii*) each letter a of all y_i, with $i \in [n]$, is replaced by $\langle a, 0, 1, 0\rangle$, and
- (*iii*) each letter v_i, with $i \in [n]$, 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 $m \geq 1$, $t_1, t_2, \ldots, t_{m-1} \in \Gamma$, $u_1 t_1 u_2 t_2 \cdots t_{n-1} u_n = u$, and $z_1 t_1 z_2 t_2 \cdots t_{n-1} z_n = z$, 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 $\langle t_i, 1, 1, 0\rangle$.

By 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

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

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

4 Synchronized Shuffles on Languages

In this section we naturally extend the three synchronized shuffle operations defined in the previous section to languages and we present some properties of some families of languages with respect to these operations.

Let $X \in \{S, W, A\}$. Then the XS-shuffle on Γ of languages $L_i \subseteq \Sigma_i^*$, with $i \in [2]$, is denoted by $L_1 \sqcup\!\sqcup_{\Gamma}^{X} L_2$ 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

$$L_1 \sqcup\!\sqcup {}^{X}_{\Gamma} L_2 = \bigcup_{u \in L_1,\, v \in L_2} u \sqcup\!\sqcup {}^{X}_{\Gamma \cap \alpha(u) \cap \alpha(v)} v.$$

We write $\sqcup\!\sqcup {}^{X}_{\Gamma}(L)$ instead of $L \sqcup\!\sqcup {}^{X}_{\Gamma} L$. 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 $\sqcup\!\sqcup {}^{X}_{\Gamma}(L) \subseteq L$. 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, $\sqcup\!\sqcup {}^{S}_{\alpha(L)}(L) = L$.

It is worth mentioning that all three synchronized shuffle operations are distributive over the union. Hence

Proposition 2 *1. For any $\Gamma \subseteq \Sigma$, $(\mathcal{P}(\Sigma^*), \cup, \sqcup\!\sqcup {}^{S}_{\Gamma}, \varnothing)$ is a commutative hemiring (i.e. a semiring without a unit element).*

2. For any $\Gamma \subseteq \Sigma$, $(\mathcal{P}(\Sigma^*), \cup, \sqcup\!\sqcup {}^{A}_{\Gamma}, \varnothing, \lambda)$ 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 $\Gamma \subseteq \alpha(L_1) \cap \alpha(L_2)$, $L_1 \sqcup\!\sqcup {}^{X}_{\Gamma} L_2 \in \mathcal{F}$. Moreover, $\mathcal{F}$ is said to be *closed* under XS-shuffle if for any two languages $L_1, L_2 \in \mathcal{F}$, $L_1 \sqcup\!\sqcup {}^{X}_{\alpha(L_1) \cap \alpha(L_2)} L_2 \in \mathcal{F}$.

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 shuffle operations defined in this paper implies the closure under all the others.

Proposition 3 *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 $L_i \subseteq \Sigma_i^*$, with $i \in [2]$, be two languages in $\mathcal{F}$. Furthermore, let $\Gamma \subseteq \Sigma_1 \cap \Sigma_2$. We define the new alphabet $\widehat{\Sigma}_2 = \{\, \widehat{a} \mid a \in \Sigma_2 \,\}$ and consider the morphism $h : \Sigma_2^* \longrightarrow \widehat{\Sigma}_2^*$, $h(a) = \widehat{a}$, $a \in \Sigma_2$. Let M be a sequential transducer which defines a finite transduction from $(\Sigma_1 \cup \widehat{\Sigma}_2)^*$ onto $(\Sigma_1 \cup \Sigma_2)^*$ and which works as follows:

1. M checks the following two conditions to be satisfied:
   (i) each occurrence of a letter a from Γ in the input word w either is immediately followed by $\widehat{a}$ or $\widehat{a}$ does not appear at all in w, and
   (ii) each occurrence of a letter $\widehat{a}$, with a from Γ, in the input word w either is immediately preceded by a or a does not appear at all in w.
2. When reading the subword $a\widehat{a}$, M outputs a.
3. When reading a and $\widehat{a}$, M outputs a.

By these explanations,

$$L_1 \sqcup\!\sqcup {}^{S}_{\Gamma}\ L_2 = T_M(L_1 \sqcup\!\sqcup\ h(L_2)).$$

Since any trio is closed under transducer mappings, it follows that $\mathcal{F}$ is fully closed under WS-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 aim, for two languages $L_i \subseteq \Sigma_i^*$, with $i \in [2]$, in $\mathcal{F}$ we define the same alphabet $\widehat{\Sigma}_2$ and morphism h as above. Furthermore, we consider the morphisms:

$$\begin{aligned}
&g_1 : (\Sigma_1 \cup \{c\})^* \longrightarrow \Sigma_1^*,\ g_1(a) = a,\ a \in \Sigma_1,\ g_1(c) = \lambda,\\
&g_2 : (\widehat{\Sigma}_2 \cup \{c\})^* \longrightarrow \widehat{\Sigma}_2^*,\ g_2(\widehat{a}) = \widehat{a},\ a \in \Sigma_2,\ g_2(c) = \lambda,\ \text{and}\\
&f : (\Sigma_1 \cup \widehat{\Sigma}_2 \cup \{c\})^* \longrightarrow (\Sigma_1 \cup \Sigma_2)^*,\ f(a) = a,\ a \in \Sigma_1,\ f(\widehat{a}) = a,\ a \in \Sigma_2,\\
&\quad f(c) = \lambda.
\end{aligned}$$

It is plain that $\Gamma = \alpha(g_1^{-1}(L_1)) \cap \alpha(g_2^{-1}(h(L_2))) = \{c\}$. Therefore

$$L_1 \sqcup\!\sqcup L_2 = f(g_1^{-1}(L_1) \sqcup\!\sqcup {}^{S}_{\Gamma}\ g_2^{-1}(h(L_2))),$$

which concludes the proof. □

Proposition 4 *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 $L_i \subseteq \Sigma_i^*$, with $i \in [2]$, be two languages in $\mathcal{F}$. Furthermore, let $\Gamma \subseteq \Sigma_1 \cap \Sigma_2$. We define the new alphabets $\widehat{\Sigma}_2 = \{\, \widehat{a} \mid a \in \Sigma_2 \,\}$ and $\overline{\Sigma}_2 = \{\, \overline{a} \mid a \in \Sigma_2 \,\}$. Let the finite substitution $s : \Sigma_2^* \longrightarrow \mathcal{P}(\widehat{\Sigma}_2 \cup \overline{\Sigma}_2)$ be defined by $s(a) = \{\overline{a}\}$, for any $a \in \Sigma_2 \setminus \Gamma$, and $s(a) = \{\overline{a}, \widehat{a}\}$, for any $a \in \Gamma$. Now we construct a sequential transducer M which reads words from $(\Sigma_1 \cup \widehat{\Sigma}_2 \cup \overline{\Sigma}_2)^*$ and outputs words in $(\Sigma_1 \cup \Sigma_2)^*$. 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 met or the input word is completely read. Along this computation, when reading a letter c from Σ_1 and $\overline{b}$ from $\overline{\Sigma}_2$, M writes c and b, respectively. However, if the currently scanned subword of the input word contains a pair of letters $(b, \overline{b})$, for some $b \in \Gamma$, 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, checks whether or not 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 ones, except the error state and the states of the second phase.

By these explanations,

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

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 5 *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 or not a regular language is XS-shuffle closed w.r.t. any given alphabet of letters subject to synchronization.

Proposition 6 *For any regular language $L \subseteq \Sigma^*$, any $\Gamma \subseteq \Sigma$, and any $X \in \{S, W, A\}$, one can algorithmically decide whether or not L is XS-shuffle closed w.r.t. Γ.*

Proof. Since the family of regular languages is a trio closed under shuffle, $\sqcup\!\sqcup {}^{X}_{\Gamma}(L)$ 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 $\sqcup\!\sqcup {}^{X}_{\Gamma}(L)$. On the other hand, the inclusion problem is decidable for regular languages, therefore one can algorithmically decide whether or not $\sqcup\!\sqcup {}^{X}_{\Gamma}(L) \subseteq L$. □

Proposition 7 *Let $\mathcal{F}$ be an arbitrary family of languages.*

1. If $\mathcal{F}$ is (fully) closed under SS-shuffle and intersection with regular sets, then $\mathcal{F}$ is closed under intersection.

2. If $\mathcal{F}$ is closed under morphisms, inverse morphisms, and (fully) closed under XS-shuffle, with $X \in \{S, W, A\}$, then $\mathcal{F}$ is closed under intersection.

Proof. 1. Let $L_i \subseteq \Sigma_i^*$, with $i \in [2]$, be two languages in $\mathcal{F}$. Clearly,

$$L_1 \cap L_2 = (L_1 \sqcup\!\sqcup {}^{S}_{\Sigma_1 \cap \Sigma_2}\ L_2) \cap (\Sigma_1 \cap \Sigma_2)^*.$$

2. As $\mathcal{F}$ is (fully) closed under XS-shuffle, for any $X \in \{S, W, A\}$, 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 shuffle operations.

Proposition 8 *The family of nonempty finite languages is the smallest family containing the language $\{ab\}$, closed under union, closed under projections, and (fully) closed under any synchronized shuffle.*

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

Conversely, the languages $\{\lambda\}$ and $\{a\}$ belong to $\mathcal{F}$. Since $\mathcal{F}$ is closed under union and projections, it suffices to prove that the singleton language consisting of an arbitrary word $w = a_1 a_2 \cdots a_n$, with $a_i \neq a_j$ and $1 \leq i \neq j \leq n$, lies in $\mathcal{F}$. We prove this by induction on n. Obviously, the assertion holds for $n \in \{0, 1, 2\}$. For any $n \geq 3$ and $X \in \{S, W, A\}$, we have $\{w\} = \{a_1 a_2 \cdots a_{n-1}\} \sqcup\!\sqcup^{X}_{\{a_{n-1}\}} \{a_{n-1} a_n\}$. □

5 Conclusion

We have introduced and investigated three synchronized shuffle operations on words and languages. Since the idea underlying such synchronized shuffle operations appears in numerous disguises throughout the computer science literature, e.g. in formal language theory and concurrency theory, the three synchronized shuffle operations studied in this paper may well be applicable to existing research issues in the aforementioned fields. We briefly hint at two such usages.

To begin with, it remains to be investigated whether or not the synchronized shuffle operations introduced in this paper can be used to define the way in which the constituting component automata of a particular team automaton interact through synchronizations of shared actions. This undoubtedly requires these synchronized shuffle operations to satisfy some of the fundamental mathematical properties studied in this paper, even more so when also the infinitary behaviour of team automata is taken into account (as can be concluded from [2]).

It moreover remains to be investigated whether or not the synchronized shuffle operations introduced in this paper 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 [20] and continued in [1].

Acknowledgement

The first author would like to thank Jetty Kleijn for many stimulating and in-depth discussions on synchronized shuffle operations.

References

1. J.C.M. Baeten and 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, and G. Rozenberg, Team automata for spatial access control. In *ECSCW 2001—Proceedings of the 7th European Conference on Computer Supported Cooperative Work, Bonn, Germany* (W. Prinz, M. Jarke, Y. Rogers, K. Schmidt, and V. Wulf, eds.), Kluwer Academic Publishers, Dordrecht, 2001, 59–77.
4. M.H. ter Beek, C.A. Ellis, J. Kleijn, and G. Rozenberg, Synchronizations in team automata for groupware systems. *Computer Supported Cooperative Work—The Journal of Collaborative Computing* 12, 1 (2003), 21–69.
5. M.H. ter Beek and J. Kleijn, Team automata satisfying compositionality. In *Proceedings of FME 2003: Formal Methods—the 12th International Symposium of Formal Methods Europe, Pisa, Italy* (K. Araki, S. Gnesi, and D. Mandrioli, eds.), LNCS 2805, Springer-Verlag, Berlin, 2003, 381–400.
6. *Handbook of Process Algebra* (J.A. Bergstra, A. Ponse, and S.A. Smolka, eds.), Elsevier Science Publishers, Amsterdam, 2001.
7. S.L. Bloom and Z. Ésik, Free shuffle algebras in language varieties. *Theoretical Computer Science* 163 (1996), 55–98.
8. J. Carmona and J. Cortadella, Input/output compatibility of reactive systems. In *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design* (M.D. Aagaard and J.W. O'Leary, eds.), LNCS 2517, Springer-Verlag, Berlin, 2002, 360–377.
9. R. De Simone, Langages infinitaires et produit de mixage. *Theoretical Computer Science* 31 (1984), 83–100.
10. S. Eilenberg, *Automata, Languages, and Machines*, Vol. A, Academic Press, New York, 1974.
11. C.A. Ellis, Team automata for groupware systems. In *Proceedings of the GROUP'97 International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge, Phoenix, Arizona* (S.C. Hayne and W. Prinz, eds.), ACM Press, New York, 1997, 415–424.
12. S. Ginsburg and E.H. Spanier, Mappings of languages by two-tape devices. *Journal of the ACM* 12, 3 (1965), 423–434.
13. J.L. Gischer, Shuffle languages, Petri nets, and context sensitive grammars. *Communications of the ACM* 24 (1981), 597–605.
14. M. Jantzen, The power of synchronizing operations on strings. *Theoretical Computer Science* 14 (1981), 127–154.
15. B. Jonsson, Compositional specification and verification of distributed systems. *ACM Transactions on Programming Languages and Systems* 16, 2 (1994), 259–303.
16. T. Kimura, An algebraic system for process structuring and interprocess communication. In *Proceedings of the 8th ACM SIGACT Symposium on Theory of Computing, Hershey, Pennsylvania*, ACM Press, New York, 1976, 92–100.
17. R. Lanotte, A. Maggiolo-Schettini, and A. Peron, Timed cooperating automata. *Fundamenta Informaticae* 42 (2000), 1–21.
18. M. Latteux and Y. Roos, Synchronized shuffle and regular languages. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa* (J. Karhumäki, H.A. Maurer, Gh. Păun, and G. Rozenberg, eds.), Springer-Verlag, Berlin, 1999, 35–44.
19. N.A. Lynch, *Distributed Algorithms*, Morgan Kaufmann Publishers, San Mateo, California, 1996.
20. R. Milner, *A Calculus of Communicating Systems*, LNCS 92, Springer-Verlag, Berlin, 1980.

21. D. Park, On the semantics of fair parallelism. LNCS 86, Springer-Verlag, Berlin, 1979, 504–526.
22. A.W. Roscoe, *The Theory and Practice of Concurrency*, Prentice Hall International Series in Computer Science, London, 1997.
23. *Handbook of Formal Languages* (G. Rozenberg and A. Salomaa, eds.), Springer-Verlag, Berlin, 1997.
24. A.C. Shaw, Software descriptions with flow expressions. *IEEE Transactions on Software Engineering* SE-4, 3 (1978), 242–254.
25. J.L.A. van de Snepscheut, *Trace Theory and VLSI Design*, LNCS 200, Springer-Verlag, Berlin, 1985.
26. W. Wechler, *Universal Algebra for Computer Scientists*, EATCS Monographs on Theoretical Computer Science 25, Springer-Verlag, Berlin, 1992.