Team Automata Satisfying Compositionality

Maurice H. ter Beek[1,*] and Jetty Kleijn[2]

[1] Istituto di Scienza e Tecnologie dell'Informazione, CNR
Area della Ricerca di Pisa, Via G. Moruzzi 1, 56124 Pisa, Italy,
mtbeek@isti.cnr.it

[2] Leiden Institute of Advanced Computer Science, Universiteit Leiden,
P.O. Box 9512, 2300 RA Leiden, The Netherlands
kleijn@liacs.nl

Abstract. A team automaton is said to satisfy compositionality if its behaviour can be described in terms of the behaviour of its constituting component automata. As an initial investigation of the conditions under which team automata satisfy compositionality, we study their computations and behaviour in relation to those of their constituting component automata. We show that the construction of team automata according to certain natural types of synchronization guarantees compositionality.
Keywords: team automata, compositionality, computations, behaviour, synchronizations, shuffles.

1 Introduction

Component-based system design is a complicated task, for which stepwise development may thus be of help. 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 specification model chosen is compositional, i.e. a specification of a composite system can be obtained from specifications of its components [16]. In case of automata-based specification models, compositionality requires that the relevant behaviour of a composite automaton can be obtained from the behaviour of its constituting component 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 sharing this action simultaneously execute it. In [3] this type of synchronization of shared actions is coined *maximal-action-indispensable* (*maximal-ai* for short). Examples

* This author's research was supported by an ERCIM postdoctoral fellowship and was partly carried out during his stays at the Leiden Institute of Advanced Computer Science of Leiden University and at the Computer and Automation Research Institute of the Hungarian Academy of Sciences.

of automata-based specification models with composition based on *maximal-ai* synchronizations include I/O automata [20, 26], I/O systems [15, 16], Cooperating (Pushdown) Automata [8, 12], Timed Cooperating Automata [18], Reactive Transition Systems [5, 6], and Interacting State Machines [21, 22]. This *maximal-ai* type of synchronization also appears in disguise in non-automata-based specification models like CSP [13] and Statecharts [11].

Team automata were introduced for the specification of groupware systems and their interconnections and were shown to provide a flexible framework for modelling collaboration between system components [2, 3, 9]. Inspired by I/O automata, 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. Team automata model the logical architecture of a design, while abstracting from concrete data, configurations, and actions. They describe a system solely in terms of an automaton, the role of actions, and synchronizations.

The crux of composing a team automaton is to define the way its constituting components interact through synchronizations. While it is noted in [26] that a single notion of composition is rather restrictive and may hinder a realistic modelling of certain types of interactions, composition of I/O automata nevertheless is unique. Within a team automaton, however, a component automaton is not forced to participate in every synchronization of an action it shares. Hence 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.

In [3] we introduced a variety of fixed strategies for choosing the synchronizations of a team automaton, thus leading to uniquely defined team automata. 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 [3]. This paper provides an initial investigation of the conditions under which these strategies lead to team automata satisfying compositionality. To this aim, we study the relation between the computations and behaviour of team automata defined according to the *maximal-ai* and *maximal-free* strategies and those of their constituting components. Since a team automaton's distinction of input, output, and internal actions is irrelevant for the results presented in this paper, it is ignored from now on.

We begin this paper by fixing some notation, followed by some definitions and results concerning team automata. Subsequently we investigate the behavioural relation between team automata and their constituting components, only interrupted by some definitions and results concerning (synchronized) shuffles. We conclude this paper with a discussion of the obtained results.

2 Preliminaries

We assume familiarity with the basic notions from formal language theory [24].

We have the following conventions. Set inclusion is denoted by $\subseteq$, whereas $\subset$ denotes a strict inclusion. Set difference of sets V and W is denoted by $V \setminus W$. For a finite set V, its cardinality is denoted by $\#V$. The empty set is denoted by $\varnothing$. For convenience we denote the set $\{1, 2, \ldots, n\}$ by $[n]$. Then $[0] = \varnothing$. The cartesian product of sets V_i, with $i \in [n]$, is denoted by $\prod_{i\in[n]} V_i$. For $j \in [n]$, $\mathrm{proj}_j : \prod_{i\in[n]} V_i \to V_j$ is defined by $\mathrm{proj}_j((a_1, a_2, \ldots, a_n)) = a_j$. The empty word is denoted by λ. The set of all finite words over an alphabet Σ (including λ) is denoted by Σ^*.

Let $f : A \to A'$ and $g : B \to B'$ be functions. Then $f \times g : A \times B \to A' \times B'$ is defined as $(f \times g)(a, b) = (f(a), g(b))$. We use $f^{[2]}$ as shorthand for $f \times f$ (not to be confused with iterated function application). Thus $f^{[2]}(a, b) = (f(a), f(b))$.

Let $h : \Sigma \to \Gamma^*$ be a function assigning to each symbol of Σ a finite word over the alphabet Γ. The homomorphic extension of h to Σ^*, also denoted by h, is defined in the usual way by $h(\lambda) = \lambda$ and $h(xy) = h(x)h(y)$ for all $x, y \in \Sigma^*$.

The function $\mathrm{pres}_\Gamma : \Sigma \to \Gamma^*$, defined by $\mathrm{pres}_\Gamma(a) = a$ if $a \in \Gamma$ and $\mathrm{pres}_\Gamma(a) = \lambda$ otherwise, preserves the symbols from Γ and erases all other symbols.

3 Component Automata and Team Automata

In this section we recall some definitions and results concerning team automata from [3], while ignoring their distinction of input, output, and internal actions.

Definition 1. *A* component automaton *is a quadruple $\mathcal{C} = (Q, \Sigma, \delta, I)$, with Q its (possibly infinite) set of* states, *Σ its set of* actions, *$Q \cap \Sigma = \varnothing$, $\delta \subseteq Q \times \Sigma \times Q$ its set of* labelled transitions, *and $I \subseteq Q$ its set of* initial states. □

Let $a \in \Sigma$. Then the set δ_a of *a-transitions* of $\mathcal{C}$ is defined as $\delta_a = \{(q, q') \mid (q, a, q') \in \delta\}$. An a-transition $(q, q) \in \delta_a$ is called a *loop* (on a).

The dynamics of a component automaton is given through its computations, while focussing on (certain) actions leads to a notion of behaviour.

Definition 2. *Let $\mathcal{C} = (Q, \Sigma, \delta, I)$ be a component automaton.*
The set $\mathbf{C}_\mathcal{C}$ of its computations *is defined as consisting of all finite sequences $\alpha = q_0a_1q_1a_2q_2 \cdots a_nq_n$, with $n \geq 0$, $q_i \in Q$ for $0 \leq i \leq n$, and $a_j \in \Sigma$ for $1 \leq j \leq n$ such that $q_0 \in I$ and $(q_i, a_{i+1}, q_{i+1}) \in \delta$ for all $0 \leq i < n$.*

Let Θ be an alphabet disjoint from Q.
*The Θ-*behaviour *$\mathbf{B}^\Theta_\mathcal{C}$ of $\mathcal{C}$ is defined as $\mathbf{B}^\Theta_\mathcal{C} = pres_\Theta(\mathbf{C}_\mathcal{C})$.* □

The Σ-behaviour of $\mathcal{C}$ is also simply called its behaviour.

For the rest of this paper we consider an arbitrary but fixed set $\mathcal{S} = \{\mathcal{C}_i \mid i \in [n]\}$ of component automata, where $n \geq 0$ and each $\mathcal{C}_i$ is specified as $\mathcal{C}_i = (Q_i, \Sigma_i, \delta_i, I_i)$. A team automaton over $\mathcal{S}$ has the cartesian product $\prod_{i\in[n]} Q_i$ of the state spaces of its components as its state space, while its actions are the actions of its components. Its transition relation, however, is based on but not fixed by the transitions of its components. More precisely, the transition relation of a team automaton over $\mathcal{S}$ is defined by choosing certain synchronizations of actions of its components while excluding others.

Definition 3. *Let $a \in \bigcup_{i\in[n]} \Sigma_i$. The set $\Delta_a(\mathcal{S})$ of* synchronizations *of a in $\mathcal{S}$ is defined as $\Delta_a(\mathcal{S}) = \{(q,q') \in \prod_{i\in[n]} Q_i \times \prod_{i\in[n]} Q_i \mid \exists j \in [n] : proj_j^{[2]}(q,q') \in \delta_{j,a}$ and $\forall i \in [n] : proj_i^{[2]}(q,q') \in \delta_{i,a}$ or $proj_i(q) = proj_i(q')\}$.* □

Let $a \in \bigcup_{i\in[n]} \Sigma_i$. Then $\Delta_a(\mathcal{S})$ thus consists of all possible combinations of a-transitions of components from $\mathcal{S}$, with all non-participating components remaining idle. It is explicitly required that at least one component is *active*, i.e. executes an a-transition. The transformation of a state of a team automaton $\mathcal{T}$ over $\mathcal{S}$ is defined by the local state changes of the components from $\mathcal{S}$ participating in the action of $\mathcal{T}$ being executed. When defining $\mathcal{T}$, a specific subset δ_a of $\Delta_a(\mathcal{S})$ thus must be chosen for each action a. This enforces a certain kind of interaction between the components constituting the team automaton.

Definition 4. *A* team automaton *over $\mathcal{S}$ is a quadruple $\mathcal{T} = (Q, \Sigma, \delta, I)$, with $Q = \prod_{i\in[n]} Q_i$, $\Sigma = \bigcup_{i\in[n]} \Sigma_i$, $\delta \subseteq \prod_{i\in[n]} Q_i \times \Sigma \times \prod_{i\in[n]} Q_i$ such that $\{(q,q') \mid (q,a,q') \in \delta\} \subseteq \Delta_a(\mathcal{S})$, for all $a \in \Sigma$, and $I = \prod_{i\in[n]} I_i$.* □

Each choice of synchronizations thus defines a team automaton. Clearly every team automaton is again a component automaton, which in its turn can be used as a component in an iteratively composed hierarchical system.

Within the formalization of a team automaton, no explicit information on loops is provided. In general one thus cannot distinguish whether a component with a loop on an action a in its local state participates in a synchronization on a by the team: this component may have been idle or—after having participated in the execution of a starting from the global state—it may have returned to its original local state. In order to relate the computations of a team to those of its components we nevertheless resort to projections. The problem of loops is resolved by implicitly assuming that the presence of a component's loop in a transition of a team implies execution of that loop. This is a maximal interpretation of the components' participation in synchronizations.

Definition 5. *Let $\mathcal{T} = (Q, \Sigma, \delta, I)$ be a team automaton over $\mathcal{S}$ and let $j \in [n]$. The* projection *$\pi_{\mathcal{C}_j}(\alpha)$ on $\mathcal{C}_j$ of a computation $\alpha \in \mathbf{C}_{\mathcal{T}}$ is defined as $\pi_{\mathcal{C}_j}(\alpha) = proj_j(q)$ in case $\alpha = q \in I$, while in case $\alpha = \beta q a q'$, for some $\beta q \in \mathbf{C}_{\mathcal{T}}$, $q, q' \in Q$, and $a \in \Sigma$, then $\pi_{\mathcal{C}_j}(\alpha) = \pi_{\mathcal{C}_j}(\beta q)$ if $a \notin \Sigma_j$ or $proj_j^{[2]}(q,q') \notin \delta_{j,a}$, and $\pi_{\mathcal{C}_j}(\alpha) = \pi_{\mathcal{C}_j}(\beta q) a\, proj_j(q')$ if $proj_j^{[2]}(q,q') \in \delta_{j,a}$.* □

Computations of team automata correspond to sequences of synchronizations and the projection on the jth component yields a computation of that component. However, since the transitions of a team automaton are only required to be subsets of all possible synchronizations, not every computation of a component of a team is part of a computation of that team.

Theorem 1. $\pi_{\mathcal{C}_j}(\mathbf{C}_{\mathcal{T}}) \subseteq \mathbf{C}_{\mathcal{C}_j}$ *for all* $j \in [n]$ *and all team automata* $\mathcal{T}$ *over* $\mathcal{S}$. $\square$

In [3] we defined several strategies for choosing the synchronizations of a team automaton, each leading to a uniquely defined team automaton. These strategies fix the synchronizations of a team automaton by defining, per action a, certain conditions on the a-transitions to be chosen from $\Delta_a(\mathcal{S})$, thus determining a unique subset of $\Delta_a(\mathcal{S})$ as the set of a-transitions of the team automaton. We refer to such subsets as *predicates* for a. Once predicates have been chosen for all actions in $\bigcup_{i \in [n]} \Sigma_i$, the team automaton over $\mathcal{S}$ defined by these predicates is unique.

Definition 6. *Let* $\mathcal{R}_a(\mathcal{S}) \subseteq \Delta_a(\mathcal{S})$*, for all* $a \in \Sigma$*, and let* $\mathcal{R} = \{\mathcal{R}_a(\mathcal{S}) \mid a \in \Sigma\}$*. Then* $\mathcal{T} = (Q, \Sigma, \delta, I)$ *is the* $\mathcal{R}$-team automaton *over* $\mathcal{S}$ *if* $\delta_a = \mathcal{R}_a(\mathcal{S})$*, for all* $a \in \Sigma$. $\square$

The predicates *is-free* and *is-ai* are based on those actions of $\mathcal{T}$ that are *free* and *ai*, respectively. An action a is *free* in $\mathcal{T}$ if none of its a-transitions is brought about by a synchronization of a by two or more components from $\mathcal{S}$, while a is *action-indispensable* (*ai* for short) in $\mathcal{T}$ if all its a-transitions are brought about by a synchronization of all components from $\mathcal{S}$ sharing a.

Definition 7. *Let* $a \in \bigcup_{i \in [n]} \Sigma_i$*. The predicate* is-free *in* $\mathcal{S}$ *for* a *is defined as* $\mathcal{R}_a^{free}(\mathcal{S}) = \{(q, q') \in \Delta_a(\mathcal{S}) \mid \#\{i \in [n] \mid a \in \Sigma_i \text{ and } proj_i^{[2]}(q, q') \in \delta_{i,a}\} = 1\}$ *and the predicate* is-action-indispensable *(*is-ai *for short) in* $\mathcal{S}$ *for* a *is defined as* $\mathcal{R}_a^{ai}(\mathcal{S}) = \{(q, q') \in \Delta_a(\mathcal{S}) \mid \forall i \in [n] : \text{ if } a \in \Sigma_i, \text{ then } proj_i^{[2]}(q, q') \in \delta_{i,a}\}$. $\square$

The predicate *is-free* thus contains *all* a-transitions from $\Delta_a(\mathcal{S})$ in which only one component participates, while the predicate *is-ai* contains *all* a-transitions from $\Delta_a(\mathcal{S})$ in which all components with a as an action participate.

The $\mathcal{R}^{free}$-team automaton ($\mathcal{R}^{ai}$-team automaton) over $\mathcal{S}$ is also called *the maximal-free* (*maximal-ai*) team automaton over $\mathcal{S}$ because it is the unique team automaton with the property that adding any synchronization yields a team automaton with an action that is not *free* (*ai*).

Note that whenever none of the components from $\mathcal{S}$ share an action, i.e. for all $i \in [n]$, $\Sigma_i \cap \bigcup_{k \in [n] \setminus \{i\}} \Sigma_k = \varnothing$, then the *maximal-free* team automaton over $\mathcal{S}$ equals the *maximal-ai* team automaton over $\mathcal{S}$.

For the rest of this paper we fix $\mathcal{T} = (Q, \Sigma, \delta, I)$ as a team automaton over $\mathcal{S}$ and we fix an alphabet Θ disjoint from Q. We also fix an element $j \in [n]$.

4 From Team Automata to Component Automata

In this section we start out from the computations and behaviour of a team automaton which we want to relate to the computations and behaviour of its constituting component automata. We address this issue element-wise, i.e. given one particular computation (behaviour) of a team automaton we consider how to extract from it the underlying computation (behaviour) of one of its constituting component automata.

According to Theorem 1 we can apply projections on the computations of the team automaton in order to obtain computations of its components. By filtering out the state information from these computations we subsequently obtain their behaviour. We thus have the situation depicted by the diagram in Fig. 1.

$$\begin{tikzcd}
\alpha \in \mathbf{C}_{\mathcal{T}} \arrow[rr, "\pi_{\mathcal{C}_j}"] \arrow[d, "\mathrm{pres}_{\Theta}"'] & & \pi_{\mathcal{C}_j}(\alpha) \in \mathbf{C}_{\mathcal{C}_j} \arrow[d, "\mathrm{pres}_{\Theta}"] \\
\mathrm{pres}_{\Theta}(\alpha) \in \mathbf{B}^{\Theta}_{\mathcal{T}} \arrow[rr, dashed, "?"] & & \mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha)) \in \mathbf{B}^{\Theta}_{\mathcal{C}_j}
\end{tikzcd}$$

Fig. 1. Extracting behaviour from team automata to component automata.

In addition we are interested in an operation that yields the Θ-behaviour of $\mathcal{C}_j$ directly from the Θ-behaviour $\mathrm{pres}_{\Theta}(\alpha)$ of $\mathcal{T}$, i.e. an operation that makes the diagram depicted in Fig. 1 commute. A natural candidate is the homomorphism pres_{Σ_j} preserving only those actions from $\mathrm{pres}_{\Theta}(\alpha)$ that belong to $\mathcal{C}_j$. However, $\mathrm{pres}_{\Sigma_j}(\mathrm{pres}_{\Theta}(\alpha)) = \mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha))$ in general does not hold.

Example 1. Consider component automata $\mathcal{C}_1$ and $\mathcal{C}_2$ as depicted in Fig. 2 and team automaton $\mathcal{T}$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$ as depicted in Fig. 3. We have $\Sigma_1 = \Sigma_2 = \{a, b\}$.

Fig. 2. Component automata $\mathcal{C}_1$ and $\mathcal{C}_2$.

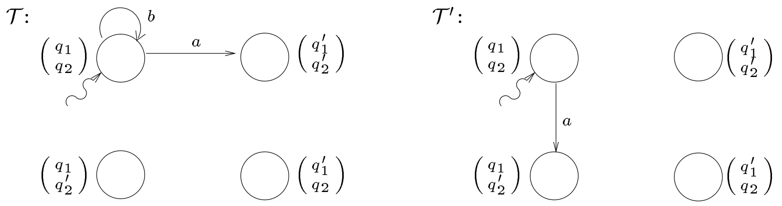

Fig. 3. Team automata $\mathcal{T}$ and $\mathcal{T}'$.

Now let $\Theta = \{a, b\}$ and let $\alpha = (q_1, q_2)b(q_1, q_2)a(q_1', q_2') \in \mathbf{C}_{\mathcal{T}}$. Then we have $\mathrm{pres}_{\Sigma_2}(\mathrm{pres}_{\Theta}(\alpha)) = ba \neq a = \mathrm{pres}_{\Theta}(q_2 a q_2') = \mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_2}(\alpha))$. □

This example shows that we cannot assume that a component participates in a synchronization *just* because it has the action that is being synchronized upon as one of its actions. There is thus no a priori relation between a component's set of actions and its participation in synchronizations of those actions. However, there exists a necessary and sufficient condition which guarantees that $\mathrm{pres}_{\Sigma_j}(\mathrm{pres}_{\Theta}(\alpha)) = \mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha))$. This condition is based on the notion of *ai* actions and guarantees the participation of all components that share the action of a synchronization, but only for transitions that are actually used in team computations. A transition $(q,q') \in \delta_a$ is *useful* (*in* $\mathcal{T}$) if there exists a computation $\alpha \in \mathbf{C}_\mathcal{T}$ such that $\alpha = \beta q a q' \gamma$ for some $\beta \in (Q\Sigma)^*$ and $\gamma \in (\Sigma Q)^*$.

Definition 8. *The set* $uAI_j(\mathcal{T})$ *of* useful j-ai *actions is defined as* $uAI_j(\mathcal{T}) = \{a \in \Sigma_j \mid \forall q,q' \in Q: \text{ if } (q,q') \in \delta_a \text{ is useful, then } proj_j^{[2]}(q,q') \in \delta_{j,a}\}$. □

This leads to the following sufficient condition under which the preserving homomorphism pres_{Σ_j} makes the diagram of Fig. 1 commute.

Lemma 1. *If* $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$*, then for all* $\alpha \in \mathbf{C}_\mathcal{T}$*,* $pres_{\Sigma_j}(pres_\Theta(\alpha)) = pres_\Theta(\pi_{\mathcal{C}_j}(\alpha))$.

Proof. Let $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$. We begin by considering $\alpha = q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n \in \mathbf{C}_\mathcal{T}$. By induction on n we prove that $\mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\alpha)) = \mathrm{pres}_\Theta(\pi_{\mathcal{C}_j}(\alpha))$.

If $n = 0$, then $\alpha = q_0$ and thus $\mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(q_0)) = \mathrm{pres}_\Theta(\pi_{\mathcal{C}_j}(q_0)) = \lambda$.

Next assume that $n = k+1$, for some $k \geq 0$, and that $\mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\beta)) = \mathrm{pres}_\Theta(\pi_{\mathcal{C}_j}(\beta))$, where $\beta = q_0 a_1 q_1 a_2 q_2 \cdots a_k q_k$. Hence $\alpha = \beta a_n q_n$. This implies $\mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\alpha)) = \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\beta))a_n$ if $a_n \in \Theta \cap \Sigma_j$ and $\mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\alpha)) = \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\beta))$ if $a_n \notin \Theta \cap \Sigma_j$.

First let $a_n \in \Theta \cap \Sigma_j$. Then $\mathrm{proj}_j^{[2]}(q_n, q_{n+1}) \in \delta_{j,a_n}$ since $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$ and thus $\mathrm{pres}_\Theta(\pi_{\mathcal{C}_j}(\alpha)) = \mathrm{pres}_\Theta(\pi_{\mathcal{C}_j}(\beta) a_n \mathrm{proj}_j(q_{n+1})) = \mathrm{pres}_\Theta(\pi_{\mathcal{C}_j}(\beta))a_n = \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\beta a_n q_n))$ by the induction hypothesis. Hence $\mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\alpha)) = \mathrm{pres}_\Theta(\pi_{\mathcal{C}_j}(\alpha))$.

Next let $a_n \notin \Theta \cap \Sigma_j$. Then $a_n \notin \Theta$ or $a_n \notin \Sigma_j$.
If $a_n \notin \Sigma_j$, then $\pi_{\mathcal{C}_j}(\alpha) = \pi_{\mathcal{C}_j}(\beta)$ and thus, by the induction hypothesis, $\mathrm{pres}_\Theta(\pi_{\mathcal{C}_j}(\alpha)) = \mathrm{pres}_\Theta(\pi_{\mathcal{C}_j}(\beta)) = \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\beta))$. Since $\mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\beta)) = \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\beta a_n q_n))$ it follows that $\mathrm{pres}_\Theta(\pi_{\mathcal{C}_j}(\alpha)) = \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\alpha))$.
If $a_n \notin \Theta$, then $\mathrm{pres}_\Theta(\pi_{\mathcal{C}_j}(\alpha)) = \mathrm{pres}_\Theta(\pi_{\mathcal{C}_j}(\beta))$ and thus, by the induction hypothesis, $\mathrm{pres}_\Theta(\pi_{\mathcal{C}_j}(\alpha)) = \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\beta)) = \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\alpha))$. □

This condition is also necessary.

Lemma 2. *If* $(\Theta \cap \Sigma_j) \setminus uAI_j(\mathcal{T}) \neq \varnothing$*, then there exists an* $\alpha \in \mathbf{C}_\mathcal{T}$ *such that* $pres_{\Sigma_j}(pres_\Theta(\alpha)) \neq pres_\Theta(\pi_{\mathcal{C}_j}(\alpha))$.

Proof. Let $(\Theta \cap \Sigma_j) \setminus uAI_j(\mathcal{T}) \neq \varnothing$. Then the following situation must exist. Let $\alpha = q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n \in \mathbf{C}_\mathcal{T}$ be such that for all $1 \leq i < n$, either $a_i \notin \Theta$, or $a_i \notin \Sigma_j$, or $\mathrm{proj}_j^{[2]}(q_{i-1}, q_i) \in \delta_{j,a_i}$, while $\mathrm{proj}_j^{[2]}(q_{n-1}, q_n) \notin \delta_{j,a_n}$, with $a_n \in \Theta \cap \Sigma_j$. Thus $\mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\alpha)) = \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(a_1 a_2 \cdots a_{n-1}))a_n$. As $\mathrm{proj}_j^{[2]}(q_{n-1}, q_n) \notin \delta_{j,a_n}$ we however have $\mathrm{pres}_\Theta(\pi_{\mathcal{C}_j}(\alpha)) = \mathrm{pres}_\Theta(\pi_{\mathcal{C}_j}(q_0 a_1 q_1 a_2 q_2 \cdots a_{n-1} q_{n-1})) \neq \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(a_1 a_2 \cdots a_{n-1}))a_n = \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_\Theta(\alpha))$. □

Theorem 2. *For all* $\alpha \in \mathbf{C}_\mathcal{T}$*,* $pres_{\Sigma_j}(pres_\Theta(\alpha)) = pres_\Theta(\pi_{\mathcal{C}_j}(\alpha))$ *if and only if* $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$. □

Until now we extracted the behaviour of the component automata of a team automaton from its computations. The above results however also provide us with a sufficient condition for obtaining behaviour of components directly from the behaviour of the team automaton.

Theorem 3. *If $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$, then $\mathbf{B}_{\mathcal{T}}^{\Theta \cap \Sigma_j} \subseteq \mathbf{B}_{\mathcal{C}_j}^{\Theta}$.*

Proof. Let $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$ and let $v \in \mathbf{B}_{\mathcal{T}}^{\Theta \cap \Sigma_j}$. Then $v \in \mathrm{pres}_{\Theta \cap \Sigma_j}(\mathbf{C}_{\mathcal{T}})$. Now let $\alpha \in \mathbf{C}_{\mathcal{T}}$ be such that $\mathrm{pres}_{\Theta \cap \Sigma_j}(\alpha) = v$. By Theorem 1, $\pi_{\mathcal{C}_j}(\alpha) \in \mathbf{C}_{\mathcal{C}_j}$. Since $\Theta \cap \Sigma_j \subseteq uAI_j(\mathcal{T})$, Lemma 1 implies that $\mathrm{pres}_{\Sigma_j}(\mathrm{pres}_{\Theta}(\alpha)) = \mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha))$. Hence $v = \mathrm{pres}_{\Theta \cap \Sigma_j}(\alpha) = \mathrm{pres}_{\Sigma_j}(\mathrm{pres}_{\Theta}(\alpha)) = \mathrm{pres}_{\Theta}(\pi_{\mathcal{C}_j}(\alpha)) \in \mathbf{B}_{\mathcal{C}_j}^{\Theta}$. □

Contrary to what might be expected from Theorem 2, the statement of Theorem 3 cannot be reversed.

Example 2. (Ex. 1 cont.) Consider team automaton $\mathcal{T}'$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$ as depicted in Fig. 3. Then $\Theta \cap \Sigma_1 = \{a, b\}$ and $uAI_1(\mathcal{T}') = \{b\}$. However, $\mathbf{B}_{\mathcal{T}'}^{\Theta \cap \Sigma_1} = \{\lambda, a\}$ is included in $\mathbf{B}_{\mathcal{C}_1}^{\Theta} = \{b^n \mid n \geq 0\} \cup \{b^n a \mid n \geq 0\}$. □

Whereas a simple projection $\pi_{\mathcal{C}_j}$ applied to a computation of $\mathcal{T}$ suffices to obtain a computation of $\mathcal{C}_j$, a similarly simple preserving homomorphism pres_{Σ_j} applied to a behaviour of $\mathcal{T}$ need not always yield a behaviour of $\mathcal{C}_j$ *unless* all actions Σ_j of $\mathcal{C}_j$ are useful j-*ai*. The reason for this difference is as follows. In a computation of $\mathcal{T}$ we still have available the information as to which components from $\mathcal{S}$ participated in each synchronization performed during this computation. When we deal with a behaviour of $\mathcal{T}$, however, only the sequence of executed actions is available, i.e. we have lost all information as to which components from $\mathcal{S}$ participated in which execution. This implies that whenever we can be sure of a component's participation in each execution of an action it has as an action itself, then we can simply apply our preserving homomorphism to a team behaviour in order to obtain the behaviour of that component. Since every action of a component from $\mathcal{S}$ is useful j-*ai* in the *maximal-ai* team automaton $\mathcal{T}$ over $\mathcal{S}$, Theorem 3 implies the following result.

Corollary 1. *If $\mathcal{T}$ is the $\mathcal{R}^{ai}$-team automaton over $\mathcal{S}$, then $\mathbf{B}_{\mathcal{T}}^{\Theta \cap \Sigma_j} \subseteq \mathbf{B}_{\mathcal{C}_j}^{\Theta}$.* □

Thus far we studied how to obtain the computations (behaviour) of the components constituting $\mathcal{S}$ from the computations (behaviour) of team automata over $\mathcal{S}$. In the next section we consider the dual approach.

5 From Component Automata to Team Automata

In this section we start out from the computations and behaviour of the component automata constituting $\mathcal{S}$. Consequently we want to describe computations and behaviour of team automata over $\mathcal{S}$. We begin by addressing this issue element-wise, i.e. given a computation (behaviour) of each component in a subset of $\mathcal{S}$ we want to know whether there exists a team automaton over $\mathcal{S}$ with a computation (behaviour) that uses this combination of computations.

Definition 9. *Let $\alpha \in \prod_{i \in [n]} \mathbf{C}_{\mathcal{C}_i}$. Then α is* used in $\mathcal{T}$ *if there exists a $\beta \in \mathbf{C}_{\mathcal{T}}$ such that for all $i \in [n]$, $\pi_{\mathcal{C}_i}(\beta) = proj_i(\alpha)$.* □

Note that any vector of initial states is used in $\mathcal{T}$ since $\prod_{i\in[n]} I_i \subseteq \mathbf{C}_{\mathcal{T}}$. If $K \subseteq [n]$ and $\alpha_k \in \mathbf{C}_{\mathcal{C}_k}$, for all $k \in K$, then we say that $\prod_{k\in K} \alpha_k$ is used in $\mathcal{T}$ whenever there exists a $\gamma \in \prod_{i\in[n]} \mathbf{C}_{\mathcal{C}_i}$ that is used in $\mathcal{T}$ and which is such that $\mathrm{proj}_k(\gamma) = \alpha_k$, for all $k \in K$. Finally, as vectors over $\prod_j \mathbf{C}_{\mathcal{C}_j}$ have one element we identify the vector and its element in those cases.

In general not all vectors of computations of components from $\mathcal{S}$ are used in $\mathcal{T}$. As said before, it may be the case that a computation of a component from $\mathcal{S}$ never participates in a team computation. Moreover, it may happen that a vector over two or more computations of components from $\mathcal{S}$ is not used as such in $\mathcal{T}$, even when each entry of this vector *is* used in $\mathcal{T}$.

Example 3. (Ex. 1 cont.) Let $\alpha' = q_2aq_2'bq_2' \in \mathbf{C}_{\mathcal{C}_2}$. Then α' is not used in $\mathcal{T}$ because there exists no $\beta \in \mathbf{C}_{\mathcal{T}}$ such that $\pi_{\mathcal{C}_2}(\beta) = \alpha'$. Now consider team automaton $\mathcal{T}''$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$ as depicted in Fig. 4.

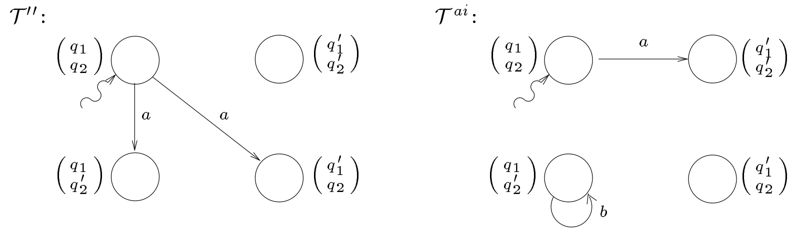

Fig. 4. Team automaton $\mathcal{T}''$ and *maximal-ai* team automaton $\mathcal{T}^{ai}$.

Now let $\alpha_1 = q_1aq_1' \in \mathbf{C}_{\mathcal{C}_1}$ and let $\alpha_2 = q_2aq_2' \in \mathbf{C}_{\mathcal{C}_2}$. Clearly both α_1 and α_2 are used in $\mathcal{T}''$ because $\beta_1 = (q_1, q_2)a(q_1', q_2) \in \mathbf{C}_{\mathcal{T}''}$ and $\beta_2 = (q_1, q_2)a(q_1, q_2') \in \mathbf{C}_{\mathcal{T}''}$. However, β_1 and β_2 are the only two nontrivial computations of $\mathcal{T}''$. Since $\pi_{\mathcal{C}_1}(\beta_2) = q_1$ and $\pi_{\mathcal{C}_2}(\beta_1) = q_2$ this means that there exists no $\beta \in \mathbf{C}_{\mathcal{T}''}$ such that $\pi_{\mathcal{C}_1}(\beta) = \alpha_1$ and $\pi_{\mathcal{C}_2}(\beta) = \alpha_2$. Hence (α_1, α_2) is not used in $\mathcal{T}''$.

Note that (α_1, α_2) *is* used in $\mathcal{T}$ because $\beta = (q_1, q_2)a(q_1', q_2') \in \mathbf{C}_{\mathcal{T}}$ is such that $\pi_{\mathcal{C}_1}(\beta) = \mathrm{proj}_1((\alpha_1, \alpha_2)) = \alpha_1$ and $\pi_{\mathcal{C}_2}(\beta) = \mathrm{proj}_2((\alpha_1, \alpha_2)) = \alpha_2$. □

While in general not every vector of computations of components from $\mathcal{S}$ is used in $\mathcal{T}$, we wonder if the situation improves in case $\mathcal{T}$ is defined in a particular way. In analogy with the previous section we first consider $\mathcal{T}$ to be the *maximal-ai* team automaton over $\mathcal{S}$. However, not even in *maximal-ai* team automata over $\mathcal{S}$ need all vectors of computations of components from $\mathcal{S}$ be used.

Example 4. (Ex. 1,3 cont.) The *maximal-ai* team automaton $\mathcal{T}^{ai}$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$ is depicted in Fig. 4. Consider $q_1 \in \mathbf{C}_{\mathcal{C}_1}$ and recall that $\alpha_2 = q_2aq_2' \in \mathbf{C}_{\mathcal{C}_2}$. Since $(q_1, q_2)a(q_1', q_2')$ is the only nontrivial computation of $\mathcal{T}^{ai}$ there exists no computation $\beta' \in \mathbf{C}_{\mathcal{T}^{ai}}$ such that $\pi_{\mathcal{C}_1}(\beta') = q_1$ and $\pi_{\mathcal{C}_2}(\beta') = \alpha_2$. Hence (q_1, α_2) is not used in $\mathcal{T}^{ai}$. □

The fact that the *maximal-ai* strategy forces components to synchronize on their shared actions provides us with enough information to formulate the conditions under which a vector of computations *is* used in a computation of the *maximal-ai* team automaton over $\mathcal{S}$. To this aim we define a vector α consisting of computations of the components from $\mathcal{S}$—one for each component—to be *ai-consistent* if there exists a word w over Σ with the following property: whenever we preserve from w only the actions of a component from $\mathcal{S}$, then we obtain exactly the behaviour resulting from the computation in α that originates from that component. In an *ai*-consistent vector the computations forming its entries thus "agree" with respect to the behaviour of their respective components.

Definition 10. *Let $\alpha \in \prod_{i\in[n]} \mathbf{C}_{\mathcal{C}_i}$. Then α is* ai-consistent *if there exists a $w \in \Sigma^*$ such that for all $i \in [n]$, $pres_{\Sigma_i}(w) = pres_{\Sigma_i}(proj_i(\alpha))$.* □

We now have a sufficient and necessary condition for a vector of computations of components from $\mathcal{S}$ to be used in the *maximal-ai* team automaton over $\mathcal{S}$.

Theorem 4. *$\alpha \in \prod_{i\in[n]} \mathbf{C}_{\mathcal{C}_i}$ is used in the $\mathcal{R}^{ai}$-team automaton over $\mathcal{S}$ if and only if α is ai-consistent.*

Proof. (If) Let $\alpha \in \prod_{i\in[n]} \mathbf{C}_{\mathcal{C}_i}$ be *ai*-consistent and let $\mathcal{T}$ be the $\mathcal{R}^{ai}$-team automaton over $\mathcal{S}$. Now let $w \in \Sigma^*$ be such that for all $i \in [n]$, $\mathrm{pres}_{\Sigma_i}(w) = \mathrm{pres}_{\Sigma_i}(\mathrm{proj}_i(\alpha))$. Let $w = a_1a_2\cdots a_m$ for some $m \geq 0$ and $a_k \in \Sigma$, for all $k \in [m]$. For each $i \in [n]$, let the indices $i_1, i_2, \ldots, i_{m_i} \in [m]$ be such that $\mathrm{pres}_{\Sigma_i}(w) = a_{i_1}a_{i_2}\cdots a_{i_{m_i}}$. Hence $m_i = 0$ if $\mathrm{pres}_{\Sigma_i}(w) = \lambda$ and $1 \leq i_1 < i_2 < \cdots < i_{m_i} \leq m$ otherwise. Moreover, observe that $\bigcup_{i\in[n]}\{i_1, i_2, \ldots, i_{m_i}\} = [m]$. Since for all $i \in [n]$, $\mathrm{pres}_{\Sigma_i}(w) = \mathrm{pres}_{\Sigma_i}(\mathrm{proj}_i(\alpha))$ and $\mathrm{proj}_i(\alpha) \in \mathbf{C}_{\mathcal{C}_i}$, it follows that for all $i \in [n]$, $\mathrm{proj}_i(\alpha) = q^i_0 a_{i_1} q^i_1 a_{i_2} \cdots a_{i_{m_i}} q^i_{m_i}$ with $q^i_0 \in I_i$ and $q^i_1, q^i_2, \ldots, q^i_{m_i} \in Q_i$.

Now define $\beta = q_0a_1q_1a_2\cdots a_mq_m$, with $q_k \in \prod_{i\in[n]} Q_i$ for all $0 \leq k \leq m$, in such a way that for all $i \in [n]$ and for all $0 \leq k \leq m$, $\mathrm{proj}_i(q_k) = q^i_\ell$ if $i_\ell \leq k < i_{\ell+1}$ with $\ell < m_i$ (by convention, $i_0 = 0$) and $\mathrm{proj}_i(q_k) = q^i_{m_i}$ if $i_{m_i} \leq k \leq m$. Consequently we prove that $\beta \in \mathbf{C}_\mathcal{T}$ while—in one stroke—$\pi_{\mathcal{C}_i}(\beta) = \mathrm{proj}_i(\alpha)$, for all $i \in [n]$, follows from an inductive argument.

By its definition, $q_0 = \prod_{i\in[n]} q^i_0 \in \prod_{i\in[n]} I_i = I$. Next consider (q_{k-1}, a_k, q_k), for some $k \in [m]$. Let $i \in [n]$. We distinguish the following two cases.

If $a_k \in \Sigma_i$, then $k = i_\ell$ for some $\ell \in [m_i]$ and $i_{\ell-1} \leq k-1 < k = i_\ell$. The definitions of q_{k-1} and q_k then yield $\mathrm{proj}_i(q_{k-1}) = q^i_{\ell-1}$ and $\mathrm{proj}_i(q_k) = q^i_\ell$. Since $\mathrm{proj}_i(\alpha) \in \mathbf{C}_{\mathcal{C}_i}$ it follows that $(q^i_{\ell-1}, q^i_\ell) \in \delta_{i,a_{i_\ell}} = \delta_{i,a_k}$.

If $a_k \notin \Sigma_i$, then $k \neq i_\ell$ for some $\ell \in [m_i]$.
If $k < i_{m_i}$, then there exists an $\ell \geq 1$ such that $i_{\ell-1} \leq k-1 < k < i_\ell$ and thus $\mathrm{proj}_i(q_{k-1}) = \mathrm{proj}_i(q_k) = q^i_{\ell-1}$.
Conversely, if $k \geq i_{m_i}$, then $i_{m_i} \leq k-1 < k \leq m$ and thus again $\mathrm{proj}_i(q_{k-1}) = \mathrm{proj}_i(q_k)$.

Since $\bigcup_{i\in[n]}\{i_1, i_2, \ldots, i_{m_i}\} = [m]$, it follows that $a_k \in \Sigma_i$ for at least one $i \in [n]$ and hence $(q_{k-1}, q_k) \in \mathcal{R}^{ai}_{a_k}(\mathcal{S}) = \delta_{a_k}$. This implies that for all $k \in [m]$, $q_0a_1q_1a_2\cdots a_kq_k \in \mathbf{C}_\mathcal{T}$ and for all $i \in [n]$, $\pi_{\mathcal{C}_i}(q_0a_1q_1a_2\cdots a_kq_k) \in \mathbf{C}_{\mathcal{C}_i}$. Hence for all $i \in [n]$, $\pi_{\mathcal{C}_i}(\beta) = \pi_{\mathcal{C}_i}(q_0a_1q_1a_2\cdots a_mq_m) = \mathrm{proj}_i(\alpha)$ and α is thus used in the *maximal-ai* team automaton $\mathcal{T}$.

(Only if) Let $\alpha \in \prod_{i\in[n]} \mathbf{C}_{\mathcal{C}_i}$ be used in the $\mathcal{R}^{ai}$-team automaton $\mathcal{T}$ over $\mathcal{S}$. Then there exists a $\beta \in \mathbf{C}_\mathcal{T}$ such that $\pi_{\mathcal{C}_i}(\beta) = \mathrm{proj}_i(\alpha)$, for all $i \in [n]$. Now let $w = \mathrm{pres}_\Sigma(\beta) \in \Sigma^*$. Since $\mathcal{T}$ is the $\mathcal{R}^{ai}$-team automaton over $\mathcal{S}$, Lemma 1 implies that $\mathrm{pres}_{\Sigma_i}(w) = \mathrm{pres}_{\Sigma_i}(\mathrm{pres}_\Sigma(\beta)) = \mathrm{pres}_\Sigma(\pi_{\mathcal{C}_i}(\beta)) = \mathrm{pres}_{\Sigma_i}(\pi_{\mathcal{C}_i}(\beta)) = \mathrm{pres}_{\Sigma_i}(\mathrm{proj}_i(\alpha))$, for all $i \in [n]$. Hence α is *ai*-consistent. □

In order to relate the computations of *maximal-ai* team automata to the computations of their constituting components, we define when $\mathcal{S}$ is *ai*-consistent.

Definition 11. *$\mathcal{S}$ is* ai-consistent *if for all $i \in [n]$ and for each $\gamma \in \mathbf{C}_{\mathcal{C}_i}$ there exists an ai-consistent vector $\alpha \in \prod_{i\in[n]} \mathbf{C}_{\mathcal{C}_i}$ such that $proj_i(\alpha) = \gamma$.* □

We have now defined *ai*-consistency both for vectors (of computations) and for $\mathcal{S}$. However, from the context it will always be clear whether we deal with an *ai*-consistent vector or rather with an *ai*-consistent $\mathcal{S}$.

If $\mathcal{S}$ is *ai*-consistent, then this guarantees that for all computations of its constituents there exists a vector of computations which is *ai*-consistent and thus each computation of a component from $\mathcal{S}$ is used in a computation of the *maximal-ai* team automaton $\mathcal{T}$ over $\mathcal{S}$. In that case the set of computations (behaviour) of a component from $\mathcal{S}$ thus equals the set of computations (behaviour) of the *maximal-ai* team automaton over $\mathcal{S}$ *projected* on that component.

Theorem 5. *Let $\mathcal{T}$ be the $\mathcal{R}^{ai}$-team automaton over $\mathcal{S}$. Then*

(1) $\mathbf{C}_{\mathcal{C}_i} = \pi_{\mathcal{C}_i}(\mathbf{C}_\mathcal{T})$, for all $i \in [n]$, if and only if $\mathcal{S}$ is ai-consistent, and
(2) if $\mathcal{S}$ is ai-consistent, then for all $i \in [n]$, $\mathbf{B}^{\Sigma_i}_{\mathcal{C}_i} = \mathbf{B}^{\Sigma_i}_{\mathcal{T}}$.

Proof. (1) (Only if) Let $\mathbf{C}_{\mathcal{C}_i} = \pi_{\mathcal{C}_i}(\mathbf{C}_\mathcal{T})$, for all $i \in [n]$. Let $\gamma \in \mathbf{C}_{\mathcal{C}_j}$. Since $\mathbf{C}_{\mathcal{C}_j} = \pi_{\mathcal{C}_j}(\mathbf{C}_\mathcal{T})$ there exists a $\beta \in \mathbf{C}_\mathcal{T}$ such that $\pi_{\mathcal{C}_j}(\beta) = \gamma$. Now let $\alpha \in \prod_{i\in[n]} \mathbf{C}_{\mathcal{C}_i}$ be such that for all $i \in [n]$, $\mathrm{proj}_i(\alpha) = \pi_{\mathcal{C}_i}(\beta)$. Since $\mathbf{C}_{\mathcal{C}_i} = \pi_{\mathcal{C}_i}(\mathbf{C}_\mathcal{T})$, for all $i \in [n]$, this α exists. Furthermore, by Theorem 4, α is *ai*-consistent. Definition 11 then implies that $\mathcal{S}$ is *ai*-consistent.

(If) Let $\mathcal{S}$ be *ai*-consistent. Due to Theorem 1 we need to prove that for all $i \in [n]$, $\mathbf{C}_{\mathcal{C}_i} \subseteq \pi_{\mathcal{C}_i}(\mathbf{C}_\mathcal{T})$. Now let $\gamma \in \mathbf{C}_{\mathcal{C}_j}$. Since $\mathcal{S}$ is *ai*-consistent there exists an *ai*-consistent vector $\alpha \in \prod_{i\in[n]} \mathbf{C}_{\mathcal{C}_i}$ such that $\mathrm{proj}_j(\alpha) = \gamma$. Then by Theorem 4 there exists a $\beta \in \mathbf{C}_\mathcal{T}$ such that $\pi_{\mathcal{C}_j}(\beta) = \mathrm{proj}_j(\alpha) = \gamma$. Hence $\gamma \in \pi_{\mathcal{C}_j}(\mathbf{C}_\mathcal{T})$.

(2) Since $\mathcal{T}$ is the $\mathcal{R}^{ai}$-team automaton over $\mathcal{S}$, Corollary 1 implies that $\mathbf{B}^{\Sigma_i}_{\mathcal{T}} \subseteq \mathbf{B}^{\Sigma_i}_{\mathcal{C}_i}$. Moreover, by (1) and Lemma 1, $\mathbf{B}^{\Sigma_i}_{\mathcal{C}_i} \subseteq \mathbf{B}^{\Sigma_i}_{\mathcal{T}}$. □

The *maximal-free* team automaton $\mathcal{T}$ over $\mathcal{S}$ consists of completely independent, non-synchronizing components. Consequently, our first intuition might be to jump to the conclusion that in that case *every* single computation of a component from $\mathcal{S}$ is used in $\mathcal{T}$. In case the components from $\mathcal{S}$ contain loops, however, a computation of a component from $\mathcal{S}$ need not be used in $\mathcal{T}$. This is due to our maximal interpretation of the component's participation in synchronizations.

Example 5. Consider component automata $\mathcal{C}_3$ and $\mathcal{C}_4$, and the *maximal-free* team automaton $\mathcal{T}^{free}$ over $\{\mathcal{C}_3, \mathcal{C}_4\}$, as depicted in Fig. 5. It is easy to see that $\alpha'' = qbr \in \mathbf{C}_{\mathcal{C}_4}$ and that there does not exist a computation $\beta \in \mathbf{C}_{\mathcal{T}^{free}}$ such that $\pi_{\mathcal{C}_4}(\beta) = \alpha''$. Hence α'' is not used in $\mathcal{T}^{free}$. □

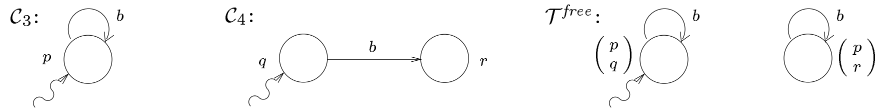

Fig. 5. Component automata $\mathcal{C}_1$ and $\mathcal{C}_2$, and *maximal-free* team automaton $\mathcal{T}^{free}$.

We thus need a condition on the presence of loops in the components from $\mathcal{S}$ to guarantee that all computations of components from $\mathcal{S}$ are used in the *maximal-free* team automaton $\mathcal{T}$ over $\mathcal{S}$.

Definition 12. *$\mathcal{S}$ is Θ-*loop limited *if for all $i \in [n]$ and for all $a \in \Theta \cap \Sigma_i$, whenever $(q,q) \in \delta_{i,a}$ for some $q \in Q_i$, then for all $k \in [n] \setminus \{i\}$, $\delta_{k,a} = \varnothing$.* □

If $\mathcal{S}$ is Σ-loop limited, then we may also simply say that it is loop limited.

In the *maximal-free* team automaton $\mathcal{T}$ over a loop-limited $\mathcal{S}$, each action of $\mathcal{C}_j$ can be executed independently of the current local states that the other components from $\mathcal{S}$ are in, since none of these other components participates in such an execution. It thus comes as no surprise that in that case each computation of a component from $\mathcal{S}$ *is* used in a computation of $\mathcal{T}$.

Lemma 3. *If $\mathcal{S}$ is loop limited, then every $\alpha \in \mathbf{C}_{\mathcal{C}_j}$ is used in the $\mathcal{R}^{free}$-team automaton over $\mathcal{S}$.*

Proof. Let $\mathcal{S}$ be loop limited and let $\mathcal{T}$ be the *maximal-free* team automaton over $\mathcal{S}$. Observe that together with Definitions 3 and 7 this implies that if $(p,p') \in \delta_{j,a}$, then for all $q \in Q$ such that $\mathrm{proj}_j(q) = p$, $(q,q') \in \delta_a = \mathcal{R}^{free}_a(\mathcal{S})$ with $\mathrm{proj}_j(q') = p'$, and for all $i \in [n] \setminus \{j\}$, $\mathrm{proj}_i(q') = \mathrm{proj}_i(q)$. Now let $\alpha = p_0 a_1 p_1 a_2 \cdots a_m p_m \in \mathbf{C}_{\mathcal{C}_j}$, i.e. $(p_{k-1}, p_k) \in \delta_{j,a_k}$, for all $1 \leq k \leq m$. Since $Q = \prod_{i\in[n]} Q_i$ and $I = \prod_{i\in[n]} I_i$, the observation above implies that there exists a computation $\beta = q_0 a_1 q_1 a_2 \cdots a_m q_m \in \mathbf{C}_{\mathcal{T}}$ such that $\mathrm{proj}_j^{[2]}(q_{k-1}, q_k) = (p_{k-1}, p_k) \in \delta_{j,a_k}$, for all $1 \leq k \leq m$. Hence $\pi_{\mathcal{C}_j}(\beta) = \alpha$ and α is thus used in $\mathcal{T}$. □

From Theorem 1 we know that given a computation of a team automaton over $\mathcal{S}$, the projection on a component from $\mathcal{S}$ is included in the set of computations of that component. Together with Lemma 3 this implies that whenever $\mathcal{S}$ is loop limited, then the set of computations of a component from $\mathcal{S}$ equals the set of computations of the *maximal-free* team automaton $\mathcal{T}$ over $\mathcal{S}$ *projected* on that component. Moreover, the behaviour of that component is included in the behaviour of $\mathcal{T}$. Like the proof of Lemma 3, also the proof of this statement is based on the observation that in a *maximal-free* team automaton, each executed action has only one participating component. This implies that the team automaton can always execute any computation of any of its components while keeping all remaining components in an initial state.

Theorem 6. *Let $\mathcal{T}$ be the $\mathcal{R}^{free}$-team automaton over $\mathcal{S}$. Then*

if $\mathcal{S}$ is loop limited, then for all $i \in [n]$, $\mathbf{C}_{\mathcal{C}_i} = \pi_{\mathcal{C}_i}(\mathbf{C}_{\mathcal{T}})$ and $\mathbf{B}^{\Sigma_i}_{\mathcal{C}_i} \subseteq \mathbf{B}^{\Sigma}_{\mathcal{T}}$.

Proof. Let $\mathcal{S}$ be loop limited and let $i \in [n]$. Then Lemma 3 implies that $\mathbf{C}_{\mathcal{C}_i} \subseteq \pi_{\mathcal{C}_i}(\mathbf{C}_{\mathcal{T}})$ and thus, by Theorem 1, $\mathbf{C}_{\mathcal{C}_i} = \pi_{\mathcal{C}_i}(\mathbf{C}_{\mathcal{T}})$. Now let $\alpha \in \mathbf{B}^{\Sigma_i}_{\mathcal{C}_i}$ and let $\beta \in \mathbf{C}_{\mathcal{C}_i}$ be such that $\text{pres}_{\Sigma_i}(\beta) = \alpha$. Since $\mathbf{C}_{\mathcal{C}_i} = \pi_{\mathcal{C}_i}(\mathbf{C}_{\mathcal{T}})$, there must exist a $\gamma \in \mathbf{C}_{\mathcal{T}}$ such that $\beta = \pi_{\mathcal{C}_i}(\gamma)$. Moreover, since $\mathcal{T}$ is the $\mathcal{R}^{free}$-team automaton over $\mathcal{S}$, it follows that we may assume that $\pi_{\mathcal{C}_k}(\gamma) \in I_k$, for all $k \in [n] \setminus \{i\}$. Hence $\text{pres}_{\Sigma}(\gamma) = \text{pres}_{\Sigma}(\pi_{\mathcal{C}_i}(\gamma)) = \text{pres}_{\Sigma_i}(\beta) = \alpha$ and thus $\alpha \in \mathbf{B}^{\Sigma}_{\mathcal{T}}$. □

The behaviour of the *maximal-free* team automaton $\mathcal{T}$ over $\mathcal{S}$ trivially is made up of the behaviour of not just one component from $\mathcal{S}$, but of the behaviour of all of the components from $\mathcal{S}$. Therefore, in general $\mathbf{B}^{\Sigma_j}_{\mathcal{C}_j} \subseteq \mathbf{B}^{\Sigma}_{\mathcal{T}}$ will be proper, even if $\mathcal{S}$ is loop limited. Furthermore, the fact that $\mathbf{C}_{\mathcal{C}_i} = \pi_{\mathcal{C}_i}(\mathbf{C}_{\mathcal{T}})$, for all $i \in [n]$, need not imply that $\mathcal{S}$ is loop limited.

Example 6. (Ex. 1 cont.) Consider the *maximal-free* team automaton $\mathcal{T}^{1,2}$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$ as depicted in Fig. 6.

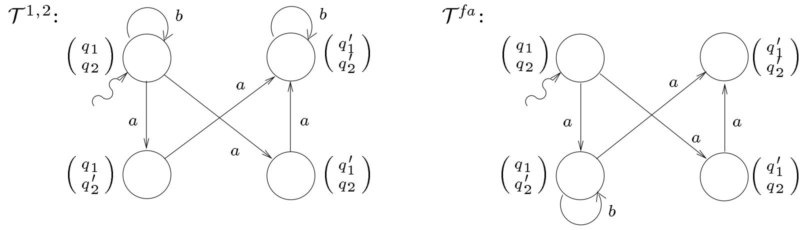

Fig. 6. Team automata $\mathcal{T}^{1,2}$ and $\mathcal{T}^{fa}$.

Recall that $\alpha' = q_2 a q_2' b q_2' \in \mathbf{C}_{\mathcal{C}_2}$. Since $\beta = (q_1, q_2)a(q_1, q_2')a(q_1', q_2')b(q_1', q_2') \in \mathbf{C}_{\mathcal{T}^{1,2}}$, α' is used in $\mathcal{T}^{1,2}$. It is moreover not difficult to see that for all $k \in [2]$, $\mathbf{C}_{\mathcal{C}_k} \subseteq \pi_{\mathcal{C}_k}(\mathbf{C}_{\mathcal{T}^{1,2}})$ and thus, by Theorem 1, $\mathbf{C}_{\mathcal{C}_k} = \pi_{\mathcal{C}_k}(\mathbf{C}_{\mathcal{T}^{1,2}})$. However, $\{\mathcal{C}_1, \mathcal{C}_2\}$ is not loop limited because $(q_1, q_1) \in \delta_{1,b}$ and $(q_2', q_2') \in \delta_{2,b}$. □

Both for *maximal-ai* and for *maximal-free* team automata over $\mathcal{S}$ we have thus found a sufficient condition on $\mathcal{S}$ (*ai*-consistency and loop limitedness, respectively) under which the computations of these team automata can be obtained from those of their constituting components. In case of *maximal-ai* team automata this condition is even necessary. As direct consequences of these results we have subsequently been able to formulate precisely when the behaviour of the components from $\mathcal{S}$ is contained in that of the *maximal-ai* (*maximal-free*) team automaton over $\mathcal{S}$.

In the remainder of this paper we moreover define the behaviour of *maximal-ai* (*maximal-free*) team automata in terms of the behaviour of their constituting components. This requires establishing which combinations of words—if any—from the behaviour of components from $\mathcal{S}$ can be combined—and in particular how—such that a word from the behaviour of the *maximal-ai* (*maximal-free*) team automaton over $\mathcal{S}$ results. This leads us to the *shuffle* (a.k.a. *merge* or *weave*) operation from the theory of formal languages.

6 Shuffles and Synchronized Shuffles

In this section we give definitions and results concerning shuffles and synchronized shuffles. A shuffle of two words is an arbitrary interleaving of the symbol occurrences of the original 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 [10, 14]. However, the underlying idea also appears in many other disguises throughout the computer science literature, e.g. in concurrency theory in the form of parallel operators modelling communication between processes [4, 23].

Definition 13. *Let Δ be an alphabet and let $u, v \in \Delta^*$. Then a word $w \in \Delta^*$ is a* shuffle *of u and v, denoted by $w \in u \,||\, v$, if $w = u_1 v_1 u_2 v_2 \cdots u_n v_n$, with $n \geq 1$, $u_i, v_i \in \Delta^*$ for all $i \in [n]$, $u = u_1 u_2 \cdots u_n$, and $v = v_1 v_2 \cdots v_n$.*
The shuffle *of languages $K, L \subseteq \Delta^*$, denoted by $K \,||\, L$, is defined as $K \,||\, L = \bigcup_{u \in K, v \in L} (u \,||\, v)$.* □

Example 7. Let $\Delta = \{a, b, c, d\}$. Let $u = abc \in \Delta^*$ and let $v = cd \in \Delta^*$. Then $u \,||\, v = \{abccd, acbcd, cabcd, abcdc, acbdc, cabdc, acdbc, cadbc, cdabc\}$. □

The shuffle operation is both commutative and associative: for all $u, v, w \in \Delta^*$, $u \,||\, v = v \,||\, u$ and $(u \,||\, v) \,||\, w = u \,||\, (v \,||\, w)$, and likewise for languages. The shuffle of languages L_i, with $i \in [n]$, can thus be defined as $||_{i \in [n]} L_i = L_1 \,||\, L_2 \,||\, \cdots \,||\, L_n$.

We now generalize the basic shuffle by defining a synchronized shuffle. Rather than just interleaving the symbols of the words being shuffled, some of these symbols are now synchronized. These synchronized symbols, while occurring in each of the words being shuffled, thus occur only once in the resulting words. Furthermore, the sequence of synchronized symbols forms a "backbone" of the resulting words. The idea underlying the various synchronized shuffles we define here appears in numerous disguises throughout the computer science literature, e.g. in concurrency theory as the concurrent composition or weave of synchronizing processes [17, 25] and in formal language theory as the 'produit de mixage' of languages [7, 19]. For the rest of this paper we use Γ to denote an arbitrary but fixed alphabet.

Definition 14. *Let Δ be an alphabet and let $u, v \in \Delta^*$. Then a word $w \in \Delta^*$ is a* synchronized shuffle *(*S-shuffle *for short) on Γ of u and v, denoted by $w \in u \,||^\Gamma\, v$, if $w \in (u_1 \,||\, v_1) x_1 (u_2 \,||\, v_2) x_2 \cdots x_{n-1} (u_n \,||\, v_n)$, with $n \geq 1$, $u_i, v_i \in (\Delta \setminus \Gamma)^*$ for all $i \in [n]$, $x_i \in \Gamma$ for all $i \in [n-1]$, $u = u_1 x_1 u_2 x_2 \cdots x_{n-1} u_n$, and $v = v_1 x_1 v_2 x_2 \cdots x_{n-1} v_n$.*
The S-shuffle *on Γ of languages $K, L \subseteq \Delta^*$, denoted by $K \,||^\Gamma\, L$, is defined as $K \,||^\Gamma\, L = \bigcup_{u \in K, v \in L} (u \,||^\Gamma\, v)$.* □

Note that the S-shuffle is indeed a generalization of the shuffle: for all $u, v \in \Delta^*$, $u \,||^\varnothing\, v = u \,||\, v$, and likewise for languages.

Example 8. (Ex. 7 cont.) Now $u \,||^{\{c\}}\, v = \{abcd\}$, whereas $u \,||^{\{b,c\}}\, v = \varnothing$. □

We proceed by defining two special cases of the S-shuffle, each obtained by varying the set of symbols required to be synchronized. Given two words over two alphabets, the *full S-shuffle* requires all symbols in the intersection of these two alphabets to be synchronized, while the *relaxed S-shuffle* requires only a specified subset of the symbols in this intersection to be synchronized. Both operations are thus defined *with respect to* the alphabets of the words involved.

Definition 15. *Let Δ_1 and Δ_2 be alphabets, let $u \in \Delta_1^*$, let $v \in \Delta_2^*$, and let $w \in (\Delta_1 \cup \Delta_2)^*$. Then*

(1) w is a full S-shuffle *(*fS-shuffle *for short) of u and v w.r.t. Δ_1 and Δ_2, denoted by $w \in u \ {}_{\Delta_1}\underline{\|}_{\Delta_2}\ v$, if w is an S-shuffle on $\Delta_1 \cap \Delta_2$ of u and v, and*
(2) w is a relaxed S-shuffle *(*rS-shuffle *for short)* on *Γ of u and v w.r.t. Δ_1 and Δ_2, denoted by $w \in u \ {}_{\Delta_1}\underline{\|}^{\Gamma}_{\Delta_2}\ v$, if w is an S-shuffle on $\Gamma \cap \Delta_1 \cap \Delta_2$ of u and v.* □

Let $L_1 \subseteq \Delta_1^*$ and let $L_2 \subseteq \Delta_2^*$. Then the fS-shuffle of L_1 and L_2 w.r.t. Δ_1 and Δ_2, denoted by $L_1 \ {}_{\Delta_1}\underline{\|}_{\Delta_2}\ L_2$, is defined as $L_1 \ {}_{\Delta_1}\underline{\|}_{\Delta_2}\ L_2 = \bigcup_{u \in L_1, v \in L_2} (u \ {}_{\Delta_1}\underline{\|}_{\Delta_2}\ v)$ and the rS-shuffle on Γ of L_1 and L_2 w.r.t. Δ_1 and Δ_2, denoted by $L_1 \ {}_{\Delta_1}\underline{\|}^{\Gamma}_{\Delta_2}\ L_2$, is defined as $L_1 \ {}_{\Delta_1}\underline{\|}^{\Gamma}_{\Delta_2}\ L_2 = \bigcup_{u \in L_1, v \in L_2} (u \ {}_{\Delta_1}\underline{\|}^{\Gamma}_{\Delta_2}\ v)$. Note that for all $u \in \Delta_1^*$, $v \in \Delta_2^*$, and $\Gamma \supseteq \Delta_1 \cap \Delta_2$, $u \ {}_{\Delta_1}\underline{\|}^{\Gamma}_{\Delta_2}\ v = u \ {}_{\Delta_1}\underline{\|}_{\Delta_2}\ v$, and likewise for languages.

Example 9. (Ex. 8 cont.) Now $u \ {}_{\Delta}\underline{\|}^{\{c\}}_{\Delta}\ v = \{abcd\}$, whereas $u \ {}_{\Delta}\underline{\|}^{\{b,c\}}_{\Delta}\ v = u \ {}_{\Delta}\underline{\|}_{\Delta}\ v = \varnothing$. Consequently, let $\Delta_1 = \{a,b,c\}$, $\Delta_2 = \{c,d\}$, $u = abc \in \Delta_1^*$, and $v = cd \in \Delta_2^*$. Then $u \ {}_{\Delta_1}\underline{\|}^{\{c\}}_{\Delta_2}\ v = u \ {}_{\Delta_1}\underline{\|}^{\{b,c\}}_{\Delta_2}\ v = u \ {}_{\Delta_1}\underline{\|}_{\Delta_2}\ v = \{abcd\}$. □

Since the S-shuffle is defined in terms of the shuffle, its commutativity follows immediately: for all $u, v \in \Delta^*$, $u \ \|^{\Gamma}\ v = v \ \|^{\Gamma}\ u$, and likewise for languages. The fact that both the fS-shuffle and the rS-shuffle are defined in terms of the S-shuffle subsequently implies that also these operations are commutative in the following sense: for all $u \in \Delta_1^*$ and $v \in \Delta_2^*$, $u \ {}_{\Delta_1}\underline{\|}_{\Delta_2}\ v = v \ {}_{\Delta_2}\underline{\|}_{\Delta_1}\ u$ and $u \ {}_{\Delta_1}\underline{\|}^{\Gamma}_{\Delta_2}\ v = v \ {}_{\Delta_2}\underline{\|}^{\Gamma}_{\Delta_1}\ u$, and likewise for languages.

The S-shuffle is moreover associative: for all $u, v, w \in \Delta^*$, $\{u\} \ \|^{\Gamma}\ (v \ \|^{\Gamma}\ w) = (u \ \|^{\Gamma}\ v) \ \|^{\Gamma}\ \{w\}$, and likewise for languages. The S-shuffle on Γ of languages L_i, with $i \in [n]$, can thus be defined as $\|^{\Gamma}_{i \in [n]} L_i = L_1 \ \|^{\Gamma}\ L_2 \ \|^{\Gamma}\ \cdots \ \|^{\Gamma}\ L_n$.

Due to the importance of the alphabets w.r.t. which words are fS-shuffled or rS-shuffled, a notion of associativity for the rS-shuffle and fS-shuffle is intuitively not immediate. We do not provide proofs here, but for all $u \in \Delta_1^*$, $v \in \Delta_2^*$, and $w \in \Delta_3^*$, the fS-shuffle satisfies the property $\{u\} \ {}_{\Delta_1}\underline{\|}_{\Delta_2 \cup \Delta_3}\ (v \ {}_{\Delta_2}\underline{\|}_{\Delta_3}\ w) = (u \ {}_{\Delta_1}\underline{\|}_{\Delta_2}\ v) \ {}_{\Delta_1 \cup \Delta_2}\underline{\|}_{\Delta_3}\ \{w\}$, while the rS-shuffle satisfies the property that $\{u\} \ {}_{\Delta_1}\underline{\|}^{\Gamma}_{\Delta_2 \cup \Delta_3}\ (v \ {}_{\Delta_2}\underline{\|}^{\Gamma}_{\Delta_3}\ w) = (u \ {}_{\Delta_1}\underline{\|}^{\Gamma}_{\Delta_2}\ v) \ {}_{\Delta_1 \cup \Delta_2}\underline{\|}^{\Gamma}_{\Delta_3}\ \{w\}$, and likewise for languages. The fS-shuffle of languages $L_i \in \Delta_i^*$, with $i \in [n]$, can thus be defined as $\underline{\|}_{\{\Delta_i \mid i \in [n]\}} L_i = (\cdots((L_1 \ {}_{\Delta_1}\underline{\|}_{\Delta_2}\ L_2) \ {}_{\Delta_1 \cup \Delta_2}\underline{\|}_{\Delta_3}\ L_3) \cdots) \ {}_{\bigcup_{i \in [n-1]} \Delta_i}\underline{\|}_{\Delta_n}\ L_n$, while the rS-shuffle on Γ of languages $L_i \in \Delta_i^*$, with $i \in [n]$, can thus be defined as $\underline{\|}^{\Gamma}_{\{\Delta_i \mid i \in [n]\}} L_i = (\cdots((L_1 \ {}_{\Delta_1}\underline{\|}^{\Gamma}_{\Delta_2}\ L_2) \ {}_{\Delta_1 \cup \Delta_2}\underline{\|}^{\Gamma}_{\Delta_3}\ L_3) \cdots) \ {}_{\bigcup_{i \in [n-1]} \Delta_i}\underline{\|}^{\Gamma}_{\Delta_n}\ L_n$.

The following alternative definition of the fS-shuffle is used in the sequel.

Theorem 7. *If* $w_i \in \Delta_i^*$*, for all* $i \in [n]$*, then* $\underline{||}_{\{\Delta_i | i\in[n]\}}\ w_i = \{w \in (\bigcup_{i\in[n]} \Delta_i)^* \mid$ $pres_{\Delta_i}(w) = w_i,$ *for all* $i \in [n]\}$. □

7 Team Automata Satisfying Compositionality

In this section we identify precisely some types of team automata that satisfy compositonality, i.e. whose behaviour can be obtained from that of their constituting components. As all synchronizations in a *maximal-ai* team automaton require the participation of all its components sharing the action being synchronized, it is not surprising that the behaviour of a *maximal-ai* team automaton equals the fS-shuffle of the behaviour of its constituting components.

Theorem 8. *Let* $\mathcal{T}$ *be the* $\mathcal{R}^{ai}$*-team automaton over* $\mathcal{S}$*. Then*

$$\mathbf{B}_{\mathcal{T}}^{\Sigma} = \underline{||}_{\{\Sigma_i | i\in[n]\}}\ \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i}.$$

Proof. ($\subseteq$) This follows immediately from Corollary 1 and Theorem 7.
($\supseteq$) Let $w \in \underline{||}_{\{\Sigma_i | i\in[n]\}} \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i}$. Then, by Theorem 7, $\mathrm{pres}_{\Sigma_i}(w) \in \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i}$, for all $i \in [n]$. Hence there exist $\alpha_i \in \mathbf{C}_{\mathcal{C}_i}$ such that $\mathrm{pres}_{\Sigma_i}(\alpha_i) = \mathrm{pres}_{\Sigma_i}(w)$, for all $i \in [n]$, and thus $\prod_{i\in[n]} \alpha_i$ is *ai*-consistent. As $w \in (\bigcup_{i\in[n]} \Sigma_i)^*$ is such that $\mathrm{pres}_{\Sigma_i}(w) = \mathrm{pres}_{\Sigma_i}(\alpha_i)$, for all $i \in [n]$, the proof of the (If)-direction of Theorem 4 implies there exists a $\beta \in \mathbf{C}_{\mathcal{T}}$ such that $\mathrm{pres}_{\bigcup_{i\in[n]} \Sigma_i}(\beta) = \mathrm{pres}_{\Sigma}(\beta) = w$. Hence $w \in \mathbf{B}_{\mathcal{T}}^{\Sigma}$. □

Example 10. (Ex. 1, 4 cont.) $\mathbf{B}_{\mathcal{T}^{ai}}^{\Sigma} = \{\lambda, a\} = \{b^n, b^n a \mid n \geq 0\}\ {}_{\Sigma_1}\underline{||}_{\Sigma_2}\ \{\lambda, ab^n \mid n \geq 0\}$ $= \underline{||}_{\{\Sigma_i | i\in[2]\}}\ \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i}$. Note that while $ba \notin \underline{||}_{\{\Sigma_i | i\in[2]\}}\ \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i}$, clearly $ba \in \mathbf{B}_{\mathcal{T}}^{\Sigma}$. □

All synchronizations in a *maximal-free* team automaton are such that only one of its components participates. However, due to the loop problem, the behaviour of a *maximal-free* team automaton equals the shuffle of the behaviour of its constituting components *provided* that this latter set is loop limited. Actually we prove a more general result, viz. that the behaviour of a team automaton that is composed according to a mixture of the *maximal-free* and *maximal-ai* strategies equals the rS-shuffle of the behaviour of its constituting components.

Theorem 9. *Let* $\bar{\Gamma} = \Sigma \setminus \Gamma$ *and let* $\mathcal{T}$ *be the* $\{\mathcal{R}_a^{ai} \mid a \in \Sigma \cap \Gamma\} \cup \{\mathcal{R}_a^{free} \mid a \in \bar{\Gamma}\}$*-team automaton over* $\mathcal{S}$*. Then*

if $\mathcal{S}$ *is* $\bar{\Gamma}$*-loop limited, then* $\mathbf{B}_{\mathcal{T}}^{\Sigma} = \underline{||}^{\Gamma}_{\{\Sigma_i | i\in[n]\}}\ \mathbf{B}_{\mathcal{C}_i}^{\Sigma_i}.$

Proof. Let $\mathcal{T}'$ be the team automaton that is obtained from $\mathcal{T}$ by attaching a label to each action from $\bar{\Gamma}$ depending on the component executing that action, i.e. $\mathcal{T}' = (Q, \Sigma', \delta', I)$ with $\Sigma' = \{[a, i] \mid a \in \bar{\Gamma} \cap \Sigma_i,\ i \in [n]\} \cup (\Sigma \cap \Gamma)$ and $\delta' = \{(q, [a, i], q') \mid a \in \bar{\Gamma},\ (q, a, q') \in \delta,\ \mathrm{proj}_i^{[2]}(q, q') \in \delta_{i,a},\ i \in [n]\} \cup (\delta \cap (Q \times \Gamma \times Q))$. Since all actions from $\bar{\Gamma}$ are *free* in $\mathcal{T}$, the behaviour of $\mathcal{T}$ is an encoding of the behaviour of $\mathcal{T}'$. Let $\psi : (\Sigma')^* \to \Sigma^*$ be the homomorphism defined by $\psi([a,i]) = a$ and $\psi(a) = a$. Then clearly $\mathbf{B}^{\Sigma}_{\mathcal{T}} = \psi(\mathbf{B}^{\Sigma'}_{\mathcal{T}'})$.

For all $i \in [n]$, let $\mathcal{C}'_i$ be the component automaton that is obtained from $\mathcal{C}_i$ by labelling each of its actions from $\bar{\Gamma}$ with i, i.e. $\mathcal{C}'_i = (Q_i, \Sigma'_i, \delta'_i, I_i)$ with $\Sigma'_i = \{[a,i] \mid a \in \bar{\Gamma} \cap \Sigma_i\} \cup (\Gamma \cap \Sigma_i)$ and $\delta'_i = \{(q,[a,i],q') \mid a \in \bar{\Gamma},\ (q,a,q') \in \delta_i\} \cup (\delta_i \cap (Q_i \times \Gamma \times Q_i))$. Obviously, $\mathbf{B}^{\Sigma_i}_{\mathcal{C}_i} = \psi(\mathbf{B}^{\Sigma'_i}_{\mathcal{C}'_i})$, for all $i \in [n]$. Let $\mathcal{S}' = \{\mathcal{C}'_i \mid i \in [n]\}$. Since $\mathcal{S}$ is $\bar{\Gamma}$-loop limited it thus follows that $\delta_{[a,i]} = \mathcal{R}^{free}_{[a,i]}(\mathcal{S}')$, for all $a \in \bar{\Gamma}$ and $i \in [n]$. Hence $\mathcal{T}'$ is the $\{\mathcal{R}^{ai}_a \mid a \in \Sigma \cap \Gamma\} \cup \{\mathcal{R}^{free}_a \mid a \in \Sigma' \setminus \Gamma\}$-team automaton over $\mathcal{S}'$. Moreover, since the components from $\mathcal{S}'$ can share actions from $\Sigma \cap \Gamma$ but not from $\Sigma' \setminus \Gamma$, it follows that for all $K \subseteq [n]$, $\bigcap_{k \in K} \Sigma'_k = \bigcap_{k \in K} \Sigma_k \cap \Gamma$, i.e. the *free* actions of $\mathcal{T}'$ are *ai* and $\mathcal{T}'$ thus equals the *maximal-ai* team automaton over $\mathcal{S}'$. Consequently the relation between the fS-shuffle and the rS-shuffle stated immediately following Definition 15, together with Theorem 8, implies that $\mathbf{B}^{\Sigma}_{\mathcal{T}} = \psi(\mathbf{B}^{\Sigma'}_{\mathcal{T}'}) = \psi(\underline{||}_{\{\Sigma'_i \mid i \in [n]\}} \mathbf{B}^{\Sigma'_i}_{\mathcal{C}'_i}) = \psi(\underline{||}^{\Gamma}_{\{\Sigma'_i \mid i \in [n]\}} \mathbf{B}^{\Sigma'_i}_{\mathcal{C}'_i})$, which is equal to $\underline{||}^{\Gamma}_{\{\psi(\Sigma'_i) \mid i \in [n]\}} \psi(\mathbf{B}^{\Sigma'_i}_{\mathcal{C}'_i}) = \underline{||}^{\Gamma}_{\{\Sigma_i \mid i \in [n]\}} \mathbf{B}^{\Sigma_i}_{\mathcal{C}_i}$ because $\psi(\Sigma' \setminus \Gamma) \cap \Gamma = \varnothing$. $\square$

Example 11. (Ex. 1, 6 cont.) The $\mathcal{R}^{free}_a \cup \mathcal{R}^{ai}_b$-team automaton $\mathcal{T}^{fa}$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$ is depicted in Fig. 6. Clearly $\{\mathcal{C}_1, \mathcal{C}_2\}$ is $\{a\}$-loop limited and indeed $\mathbf{B}^{\Sigma}_{\mathcal{T}^{fa}} = \underline{||}^{\{b\}}_{\{\Sigma_i \mid i \in [2]\}} \mathbf{B}^{\Sigma_i}_{\mathcal{C}_i}$. $\square$

Theorem 10. *Let $\mathcal{T}$ be the $\mathcal{R}^{free}$-team automaton over $\mathcal{S}$. Then*

if $\mathcal{S}$ is loop limited, then $\mathbf{B}^{\Sigma}_{\mathcal{T}} = ||_{i \in [n]} \mathbf{B}^{\Sigma_i}_{\mathcal{C}_i}$.

Proof. This follows immediately from Theorem 9 with $\Sigma \cap \Gamma = \varnothing$. $\square$

Example 12. (Ex. 1, 6 cont.) Clearly $\{\mathcal{C}_1, \mathcal{C}_2\}$ is $\{a\}$-loop limited and indeed $\mathbf{B}^{\{a\}}_{\mathcal{T}^{1,2}} = \{\lambda, a, aa\} = \{\lambda, a\} \mid\mid \{\lambda, a\} = ||_{i \in [2]} \mathbf{B}^{\{a\}}_{\mathcal{C}_i}$. Since $\{\mathcal{C}_1, \mathcal{C}_2\}$ however is not loop limited, it is no surprise that $ab \notin \mathbf{B}^{\Sigma}_{\mathcal{T}^{1,2}}$, whereas $ab \in ||_{i \in [2]} \mathbf{B}^{\Sigma_i}_{\mathcal{C}_i}$. $\square$

8 Conclusion

In this paper we have shown that—under certain conditions—team automata defined according to the *maximal-ai* and *maximal-free* strategies exhibit a behaviour that equals a certain type of (synchronized) shuffles of the behaviour of their constituting component automata. We have thus identified some types of team automata satisfying compositionality.

In [3] we showed how I/O automata fit in the framework of team automata as a special type of *maximal-ai* team automata. This in fact holds for more automata-based specification models with composition based on the *maximal-ai* strategy [1]. For such models, our results on the relation between the computations and behaviour of team automata and those of their constituting components extend known results: we single out precisely which characteristics of the *maximal-ai* strategy—e.g. the fact that an action is *ai*, useful *j-ai*, or *maximal-ai*—are responsible for a particular behavioural relation, leading to less stringent conditions.

In the context of the *maximal-free* strategy, we have presented new results on the relation between the computations and behaviour of team automata and those of their constituting components. In this case the loop problem popped up, which required a condition in the form of loop limitedness in order to be able to decide whether a component participates in the team execution of a loop. The reason is our maximal interpretation of the component's participation in synchronizations. In [1, 2] we show how to circumvent the loop problem by switching to vectors of actions. In [1] we furthermore extend all results presented in this paper to the case of infinite behaviour.

In order to identify more types of team automata satisfying compositionality, it remains to determine the conditions under which the behaviour of team automata defined according to other strategies can be obtained from the behaviour of their constituting component automata.

While ignored in this paper, team automata distinguish input, output, and internal actions. In [3] we defined an operation that "hides" the input and output actions of a team automaton from other team automata by making them internal. This prohibits their further use in synchronizations on a higher level of an iteratively composed hierarchical system and is thus important for component-based design by means of a step-by-step refinement of specifications. In [3] specifications were refined step-by-step in an example modelling the hierarchical design of a groupware architecture by team automata.

In [3] we furthermore showed—under certain very relaxed conditions—that the order in which a team automaton is iteratively composed is irrelevant. Let the team automata $\mathcal{T}$, $\mathcal{T}'$, and $\mathcal{T}''$ be iteratively composed over the component automata $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ in the way sketched in Fig. 7.

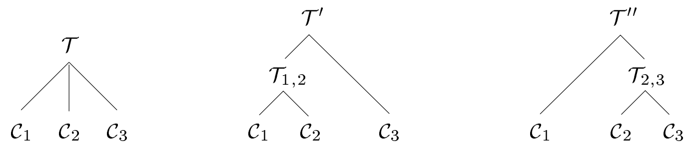

Fig. 7. Team automata $\mathcal{T}$, $\mathcal{T}'$, and $\mathcal{T}''$ composed iteratively over $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$.

If all the team automata depicted in Fig. 7 are composed according to the same strategy, then $\mathcal{T}$, $\mathcal{T}'$, and $\mathcal{T}''$ have the same set of actions and—upto a reordering—the same set of (initial) states and the same transition relation. In this case, they furthermore exhibit the same behaviour. In addition, the results presented in this paper show that if $\mathcal{T}$ is composed according to the *maximal-ai* or *maximal-free* strategy, then its behaviour equals a certain type of (synchronized) shuffles of the behaviour of $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$. It remains to study how these results can be extended to the case of iteratively composed team automata, such as $\mathcal{T}'$ and $\mathcal{T}''$.

Together with the syntactic hierarchical results of [3], the results presented in this paper thus show that the team automata framework is well suited for component-based system design by means of a stepwise development of specifications based on decomposition and refinement.

9 Acknowledgments

We thank Mieke Massink and Josep Carmona for their useful comments on a preliminary version of this paper.

References

1. 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, Universiteit Leiden, 2003.
2. M.H. ter Beek, C.A. Ellis, J. Kleijn, and G. Rozenberg, Team Automata for CSCW. In *Proceedings 2nd International Colloquium on Petri Net Technologies for Modelling Communication Based Systems* (H. Weber, H. Ehrig, and W. Reisig, eds.), Fraunhofer Institute for Software and Systems Engineering, 2001, 1-20.
3. 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.
4. *Handbook of Process Algebra* (J.A. Bergstra, A. Ponse, and S.A. Smolka, eds.), Elsevier Science, 2001.
5. J. Carmona and J. Cortadella, Input/Output Compatibility of Reactive Systems. In *Proceedings 4th International Conference on Formal Methods in Computer-Aided Design* (M.D. Aagaard and J.W. O'Leary, eds.), *LNCS* 2517, Springer-Verlag, 2002, 360-377.
6. J. Carmona, J. Cortadella, and E. Pastor, Synthesis of Reactive Systems: Application to Asynchronous Circuit Design. In *Concurrency and Hardware Design—Advances in Petri Nets* (J. Cortadella, A. Yakovlev, and G. Rozenberg, eds.), Springer-Verlag, 2002, 107-151.
7. R. De Simone, Langages Infinitaires et Produit de Mixage. *Theoretical Computer Science* 31 (1984), 83-100.
8. D. Drusinsky and D. Harel, On the Power of Bounded Concurrency I: Finite Automata. *Journal of the ACM* 41, 3 (1994), 517-539.
9. C.A. Ellis, Team Automata for Groupware Systems. In *Proceedings International Conference on Supporting Group Work: The Integration Challenge* (S.C. Hayne and W. Prinz, eds.), ACM Press, 1997, 415-424.
10. S. Ginsburg and E.H. Spanier, Mappings of Languages by Two-Tape Devices. *Journal of the ACM* 12, 3 (1965), 423-434.
11. D. Harel, Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming* 8 (1987), 231-274.
12. T. Hirst and D. Harel, On the Power of Bounded Concurrency II: Pushdown Automata. *Journal of the ACM* 41, 3 (1994), 540-554.
13. C.A.R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.
14. M. Jantzen, The Power of Synchronizing Operations on Strings. *Theoretical Computer Science* 14 (1981), 127-154.

15. B. Jonsson, Compositional Verification of Distributed Systems. Ph.D. thesis, Department of Computer Systems, Uppsala University, 1987.
16. B. Jonsson, Compositional Specification and Verification of Distributed Systems. *ACM Transactions on Programming Languages and Systems* 16, 2 (1994), 259-303.
17. T. Kimura, An Algebraic System for Process Structuring and Interprocess Communication. In *Proceedings 8th Symposium on Theory of Computing*, ACM Press, 1976, 92-100.
18. R. Lanotte, A. Maggiolo-Schettini, and A. Peron, Timed Cooperating Automata. *Fundamenta Informaticae* 42 (2000), 1-21.
19. M. Latteux and Y. Roos, Synchronized Shuffle and Regular Languages. In *Jewels are Forever* (J. Karhumäki, H.A. Maurer, Gh. Păun, and G. Rozenberg, eds.), Springer-Verlag, 1999, 35-44.
20. N.A. Lynch and M.R. Tuttle, An Introduction to Input/Output Automata. *CWI Quarterly* 2,3 (1989), 219-246.
21. D. von Oheimb, Interacting State Machines: A Stateful Approach to Proving Security. To appear in *Proceedings International Conference on Formal Aspects of Security* (A. Abdallah, P. Ryan, and S. Schneider, eds.), *LNCS* 2629, Springer-Verlag, 2003.
22. D. von Oheimb and V. Lotz, Formal Security Analysis with Interacting State Machines. In *Proceedings 7th European Symposium on Research in Computer Security* (D. Gollmann, G. Karjoth, and M. Waidner, eds.), *LNCS* 2502, Springer-Verlag, 2002, 212-228.
23. A.W. Roscoe, *The Theory and Practice of Concurrency*, Prentice Hall, 1997.
24. A. Salomaa, *Formal Languages*, Academic Press, 1973.
25. J.L.A. van de Snepscheut, *Trace Theory and VLSI Design*, *LNCS* 200, Springer-Verlag, 1985.
26. M.R. Tuttle, *Hierarchical Correctness Proofs for Distributed Algorithms*. Master's thesis, Department of Electrical Engineering and Computer Science, MIT, 1987.