Modularity for Teams of I/O Automata

Maurice H. ter Beek [a] and Jetty Kleijn [b]

[a] *ISTI, Area della Ricerca CNR di Pisa, Via G. Moruzzi 1, 56124 Pisa, Italy*

[b] *LIACS, Universiteit Leiden, P.O. Box 9512, 2300 RA Leiden, The Netherlands*

Abstract

It is shown how *Input/Output automata* fit in the framework of *team automata*, thus making it possible to view certain notions and results regarding their modular structure as special instances of more general observations.

Key words: formal methods, I/O automata, team automata, modularity

1 Introduction

Input/Output automata (or I/O automata) have originally been introduced in [16,22] as a model for distributed computations in asynchronous networks and a means to construct correctness proofs of distributed algorithms. Basically, an I/O automaton is a transition system with action names labeling its transitions. A distinction is made between internal actions and external (i.e. input and output) actions used for communication with the environment, which may consist of other I/O automata. I/O automata can be composed using a synchronous product construction yielding a new I/O automaton. Since their introduction I/O automata have been augmented with, e.g., time and probability [18,21] and, together with its many variants, the model is now widely used for describing reactive, distributed systems [6–13,16–23].

Inspired by I/O automata, team automata were introduced in [5] to model components of groupware systems and their interconnections. They were further developed as a formal model in, e.g., [1–3]. In particular, in [1] they have been shown to provide a solid and generic theoretical framework for the study of synchronization mechanisms in automata models. To allow a flexible modeling of various ways of collaboration in groupware systems, a number of the restrictions of I/O automata have been dropped. Again internal, input, and output actions are distinguished. Composition of automata is however not based on an a priori fixed mode of synchronization, but flexible. This makes it possible to define a wide variety of protocols for the collaboration and communication between a system and its environment. Team automata impose hardly any restrictions on the role of the actions in the various components and are naturally suited to describe hierarchically constructed systems.

It is our aim here to show how I/O automata fit in the framework of team automata, thus placing certain known results for I/O automata in a broader context and allowing to transfer notions and results from team automata to the more specific I/O automata. The emphasis is on their modular structure. In [1] we also studied the behavior of team automata and its implications for I/O automata, which we intend to be the subject of a forthcoming paper.

Notations

Let $\mathcal{I} \subseteq \mathbb{N}$ be a possibly infinite set of indices. Assume that $\mathcal{I}$ is given by $\mathcal{I} = \{i_1, i_2, \ldots\}$, with $i_j < i_k$ if $j < k$. For a collection of sets V_i, with $i \in \mathcal{I}$, we denote by $\prod_{i\in\mathcal{I}} V_i$ the cartesian product consisting of the elements $(v_{i_1}, v_{i_2}, \ldots)$ with $v_i \in V_i$ for each $i \in \mathcal{I}$. If $v_i \in V_i$ for each $i \in \mathcal{I}$, then $\prod_{i\in\mathcal{I}} v_i$ denotes the element $(v_{i_1}, v_{i_2}, \ldots)$ of $\prod_{i\in\mathcal{I}} V_i$. If $\mathcal{I} = \varnothing$, then $\prod_{i\in\mathcal{I}} V_i = \varnothing$. For each $j \in \mathcal{I}$ and $(v_{i_1}, v_{i_2}, \ldots) \in \prod_{i\in\mathcal{I}} V_i$, we define $\mathrm{proj}_j((v_{i_1}, v_{i_2}, \ldots)) = v_j$. If $\mathcal{J} \subseteq \mathcal{I}$, then $\mathrm{proj}_{\mathcal{J}}((v_{i_1}, v_{i_2}, \ldots)) = \prod_{j\in\mathcal{J}} v_j$.

2 Team Automata and I/O Automata

Component automata are the basic building blocks of team automata. A component automaton is a labeled transition system. The labels represent the actions of the automaton. Three types of actions are distinguished.

Definition 1 *A* component automaton *is a (labeled) transition system* $\mathcal{C} = (P, (\Gamma_{inp}, \Gamma_{out}, \Gamma_{int}), \gamma, J)$*, with* P *its set of* states*;* $\Gamma = \Gamma_{inp} \cup \Gamma_{out} \cup \Gamma_{int}$ *its set of* actions *specified by three mutually disjoint sets* Γ_{inp}*,* Γ_{out}*, and* Γ_{int} *of* input, output *and* internal *actions, respectively, and* $P \cap \Gamma = \varnothing$*;* $\gamma \subseteq P \times \Gamma \times P$ *its set of (labeled)* transitions*; and* $J \subseteq P$ *its set of* initial states.
Let $a \in \Gamma$*. Then* $\gamma_a = \gamma \cap (P \times \{a\} \times P)$ *is the set of* a-transitions *of* $\mathcal{C}$*;* a *is* enabled *in* $\mathcal{C}$ *at state* $p \in P$*, if there exists a* $p' \in P$ *such that* $(p, a, p') \in \gamma$*; and* $\mathcal{C}$ *is* a-enabling *if* a *is enabled at every state* p *of* $\mathcal{C}$*.*

For the sequel, we let $\mathcal{S} = \{\mathcal{C}_i \mid i \in \mathcal{I}\}$ with $\mathcal{I} \subseteq \mathbb{N}$, be a fixed set of component automata, in which each $\mathcal{C}_i$ is specified as $(Q_i, (\Sigma_{i,inp}, \Sigma_{i,out}, \Sigma_{i,int}), \delta^i, I_i)$, with $\Sigma_i = \Sigma_{i,inp} \cup \Sigma_{i,out} \cup \Sigma_{i,int}$ as set of actions. $\Sigma = \bigcup_{i\in\mathcal{I}} \Sigma_i$ is the set of actions of $\mathcal{S}$ and $Q = \prod_{i\in\mathcal{I}} Q_i$ is the state space of $\mathcal{S}$. To avoid technical anomalities we assume that none of the Q_i is empty. Thus Q is empty if and only if $\mathcal{I} = \varnothing$.

Component automata cooperate by synchronizing on common actions. Together they define a complete transition space, consisting of all possible combinations of identically labeled transitions with all non-participating components remaining idle.

Definition 2 *Let* $a \in \Sigma$. *Then* $\Delta_a(\mathcal{S}) = \{(q,a,q') \mid q,q' \in Q \wedge [\exists i \in \mathcal{I} : (proj_i(q), a, proj_i(q')) \in \delta^i] \wedge [\forall i \in \mathcal{I} : (proj_i(q), a, proj_i(q')) \in \delta^i \vee proj_i(q) = proj_i(q')]\}$ *is the set of all possible* synchronizations *on* a *in* $\mathcal{S}$.
$\Delta(\mathcal{S}) = \bigcup_{a \in \Sigma} \Delta_a(\mathcal{S})$ *is the set of all synchronizations of* $\mathcal{S}$.

Given a set of component automata, different collaboration protocols can be modeled by choosing different synchronizations as the transitions of a composed automaton. Such an automaton has the cartesian product of the (initial) states of the components as its (initial) states. To allow hierarchically constructed systems within the setup of team automata, a composed automaton has again internal, input, and output actions. The internal actions are the internal actions of the components, the output actions are those appearing at least once as an output action in a component, and the input actions are those external actions that are never output. This reflects the idea that an action occurring both as input and as output in a system, remains observable as output to the environment. It is assumed that internal actions are not externally observable and thus not available for synchronizations. This is not imposed by a restriction on the synchronizations allowed, but rather by the syntactical requirement that each internal action must belong to a unique component:

$\mathcal{S}$ is said to be *composable* if $\Sigma_{i,int} \cap \bigcup_{j \in \mathcal{I} \setminus \{i\}} \Sigma_j = \varnothing$ for all $i \in \mathcal{I}$.

Moreover, within a team automaton each internal action can be executed (from a global state) whenever it can be executed by the component to which it belongs (at the current local state).

Definition 3 *Let* $\mathcal{S}$ *be composable. Then a* team automaton *over* $\mathcal{S}$ *is a transition system* $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$, *with set of states* $Q = \prod_{i \in \mathcal{I}} Q_i$ *and set of initial states* $I = \prod_{i \in \mathcal{I}} I_i$; *actions* $\Sigma = \bigcup_{i \in \mathcal{I}} \Sigma_i$ *specified by* $\Sigma_{int} = \bigcup_{i \in \mathcal{I}} \Sigma_{i,int}$, $\Sigma_{out} = \bigcup_{i \in \mathcal{I}} \Sigma_{i,out}$, *and* $\Sigma_{inp} = (\bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}) \setminus \Sigma_{out}$; *and transitions* $\delta \subseteq Q \times \Sigma \times Q$ *such that* $\delta \subseteq \Delta(\mathcal{S})$ *and moreover* $\delta_a = \Delta_a(\mathcal{S})$ *for all* $a \in \Sigma_{int}$.

It is immediate that every team automaton is again a component automaton, which in its turn can be used in a higher-level team.

The team automata framework is fairly general and flexible. An external action may occur in different components as input or output and choosing a particular mode of synchronization, which may also depend on the different roles an action has, is left to the designer of an application. [1]

[1] See, e.g., [2] for models of different forms of collaboration and cooperation between components, like *peer-to-peer* synchronizations and *master-slave* synchronizations.

Also the theory of I/O automata uses component automata as basic units. The construction of a composed system is however based on assumptions on how reactive, distributed systems behave and interact. Composition is intended to model *communication* between components rather than arbitrary collaborations. There is no choice as to which transitions to include in a composition. A fixed synchronous product construction formalizes the idea of components proceeding independently, but subject to the restriction that any common action is executed synchronously by all components sharing that action.

In general, for a set $\mathcal{S}$ of component automata as specified before, the *synchronous product* $\chi^{\mathcal{S}}$ of the transitions from the components in $\mathcal{S}$ is defined by:

$$\chi^{\mathcal{S}} = \{(q, a, q') \in \Delta(\mathcal{S}) \mid \forall i \in \mathcal{I} : [a \in \Sigma_i \Rightarrow (\mathrm{proj}_i(q), a, \mathrm{proj}_i(q')) \in \delta^i]\}.$$

If $\mathcal{S}$ is composable, then the *synchronous product automaton* over $\mathcal{S}$, denoted by $\mathcal{X}(\mathcal{S})$, is the team automaton over $\mathcal{S}$ which has $\chi^{\mathcal{S}}$ as its set of transitions. Note that $\chi^{\mathcal{S}}$ satisfies the requirements of Def. 3. In particular, we have $(\chi^{\mathcal{S}})_a = \Delta_a(\mathcal{S})$ for every internal action a.

A synchronous product automaton thus has as its transitions all and only those synchronizations on an action which involve all components sharing that action. A synchronization on an action which is both input and output models a communication. Now a basic assumption within the theory of I/O automata is that input actions are controlled by the environment, whereas output actions are locally controlled. This implies that—similar to the case of internal actions—whenever a component can execute an output action (at its current local state), then it should be able to do so (from the global state) in the synchronous product automaton. In particular it should never be blocked by component automata that are currently not ready for this communication. Consequently, only component automata are considered which are *input-enabled*: each input action is enabled at each state. Input-enabledness guarantees that a component automaton is always ready to receive input, whatever its current local state. This is a necessary condition for the local control of output actions, as can be proven in a more general setting.

First, we formalize the concept of local control using the notion of omnipresence from [1]. Let $\delta \subseteq \Delta(\mathcal{S})$ be a subset of the set of synchronizations of $\mathcal{S}$. A transition $(p, a, p') \in \delta^i$ of a component automaton $\mathcal{C}_i$ from $\mathcal{S}$ is said to be *i-omnipresent in δ* if for all $q \in Q$ such that $\mathrm{proj}_i(q) = p$, there exists a $(q, a, q') \in \delta$ such that $\mathrm{proj}_i(q') = p'$. Assume now that $\mathcal{S}$ is composable and let $\mathcal{T}$ be the team automaton over $\mathcal{S}$ with set of transitions δ. Then an action a of component automaton $\mathcal{C}_i$ is *locally controlled* by $\mathcal{C}_i$ in $\mathcal{T}$, if all a-transitions of $\mathcal{C}_i$ are i-omnipresent in δ. Hence whatever the global state of the team, the ith component can execute any of its currently available local a-transitions. Note that indeed this definition implies that the internal actions of any team automaton are under the local control of the component to which they belong.

Next we establish a general relationship between omnipresence and enabling:

Lemma 4 *Let $i \in \mathcal{I}$ and $(p, a, p') \in \delta^i$. Then (p, a, p') is i-omnipresent in $\chi^{\mathcal{S}}$ if and only if for all $j \in \mathcal{I}$ such that $j \neq i$, $\mathcal{C}_j$ is a-enabling whenever $a \in \Sigma_j$.*

PROOF. The only-if-direction is immediate. Only the if-direction is proven. Let $q \in Q$ be such that $\mathrm{proj}_i(q) = p$. For all $j \in \mathcal{I}$ such that $j \neq i$ and $a \in \Sigma_j$, the fact that $\mathcal{C}_j$ is a-enabling implies that there exists a state $p_j \in Q_j$ such that $(\mathrm{proj}_j(q), a, p_j) \in \delta^j$. Thus there exists a state $q' \in Q$ with $\mathrm{proj}_i(q') = p'$ and, for all $j \in \mathcal{I}$ such that $j \neq i$ and $a \in \Sigma_j$, $\mathrm{proj}_j(q') = p_j$ where p_j is as above, and $\mathrm{proj}_k(q') = \mathrm{proj}_k(q)$ for all $k \in \mathcal{I}$ such that $a \notin \Sigma_k$. Then, by definition, $(q, a, q') \in \chi^{\mathcal{S}}$ and hence (p, a, p') is i-omnipresent in $\chi^{\mathcal{S}}$. $\square$

Consequently, in a synchronous product automaton composed of input-enabled component automata, it is guaranteed that each of the components locally controls its output actions if it does not have to synchronize with another component sharing that action as output. Clearly, this can easily be avoided in the framework of team automata by excluding this type of synchronizations from the set of transitions of the composed automaton. Within the theory of I/O automata however it is simply beforehand forbidden that components share output actions by imposing an additional restriction on the components:

$\mathcal{S}$ is said to be *compatible* if it is composable and

$$\Sigma_{i,out} \cap \bigcup_{j \in \mathcal{I} \setminus \{i\}} \Sigma_{j,out} = \varnothing \text{ for all } i \in \mathcal{I}.$$

Here an *I/O automaton* is now formally defined as a component automaton which is input-enabled. [2]

From Lemma 4 it follows immediately that the internal and output actions in the synchronous product automaton over a compatible set of I/O automata are indeed locally controlled. From a behavioral point of view, this was found to be an important property already in the earliest versions of I/O automata [14,15,22]. Here we have a more structural version of this property:

Theorem 5 *Let $\mathcal{S}$ be a compatible set of I/O automata. Let $i \in \mathcal{I}$ and a an internal or output action of $\mathcal{C}_i$. Then a is locally controlled by $\mathcal{C}_i$ in $\mathcal{X}(\mathcal{S})$.*

In order to obtain an I/O automaton when composing I/O automata, the synchronous product construction should preserve input-enabledness. In fact, a more general property can be derived from Lemma 4.

Lemma 6 *Let $a \in \Sigma$ be such that, for all $i \in \mathcal{I}$, whenever $a \in \Sigma_i$, then $\mathcal{C}_i$ is a-enabling. Let $\delta \subseteq \Delta(\mathcal{S})$ be such that $(\chi^{\mathcal{S}})_a \subseteq \delta_a$. Then for every $q \in Q$ there exists a $q' \in Q$ such that $(q, a, q') \in \delta$.*

[2] This definition corresponds to I/O automata that are called *unfair* [7] or *safe* [22].

PROOF. Let $q \in Q$ and let $i \in \mathcal{I}$ be such that $a \in \Sigma_i$. Since $\mathcal{C}_i$ is a-enabling, there exists a state $p' \in Q_i$ such that $(\mathrm{proj}_i(q), a, p') \in \delta^i$. By Lemma 4 such a $(\mathrm{proj}_i(q), a, p')$ is i-omnipresent in $\chi^{\mathcal{S}}$. Consequently, there exists a state $q' \in Q$ such that $(q, a, q') \in \chi^{\mathcal{S}}$. Since $(\chi^{\mathcal{S}})_a \subseteq \delta_a$ it follows that $(q, a, q') \in \delta$. $\square$

By observing that the input actions of a team automaton occur only as input actions in its components, we obtain as a corollary of Lemma 6 that the synchronous product construction preserves input-enabledness:

Theorem 7 *Let $\mathcal{S}$ be composable. If all component automata in $\mathcal{S}$ are input-enabled, then $\mathcal{X}(\mathcal{S})$ is also input-enabled.*

Consequently, if all component automata in a composable set are I/O automata, then their synchronous product automaton is also an I/O automaton. However, as observed earlier, within the theory of I/O automata only compatible sets of I/O automata are used to compose new I/O automata.

If $\mathcal{S}$ is a compatible set of I/O automata, then $\mathcal{X}(\mathcal{S})$ is now referred to here as the *team I/O automaton* over $\mathcal{S}$.

3 Modular Constructions: Subteams and Superteams

From the previous section we know that both team automata and I/O automata can be used as components to define higher-level team (I/O) automata. Under the assumption that certain natural properties like commutativity and associativity are guaranteed, it thus becomes feasible to design systems in a modular, iterative fashion. Conversely, an appropriate notion of subautomaton would make it possible to decompose systems constructed as team (I/O) automata into separate automata, which in their turn may again be composed systems. Hence, subautomata form an integral part of a modular approach by which one would deduce recursively properties of hierarchically defined systems from their components. Within the framework of team automata, subteams and iterated teams have been defined and related to one another in, e.g., [2]. So far, within the theory of I/O automata, the concept of subautomaton has not been considered explicitly as a structural notion. In this section we investigate to what extent the definitions of subautomaton and iterated composition for team automata can be applied successfully to I/O automata.

3.1 Subteams

By focusing on a subset of the component automata forming a team automaton, a subteam automaton can be distinguished. Its transitions are restricted versions of those transitions of the team automaton in which at least one of the components under consideration is actively involved. Its actions are the actions of these component automata. Their classification as input, output, or internal is based on their roles in the component automata defining the subteam: An external action which only occurs as an input action in the components considered, is an input action of the subteam even if it is an output action of one of the remaining component automata. This makes it possible to view a subteam as an independent team automaton without the context of the full team. Note that every subset of a composable set of component automata is again composable.

Let $\mathcal{K} \subseteq \mathcal{I}$. Then $\mathcal{S}_\mathcal{K} = \{\mathcal{C}_k \mid k \in \mathcal{K}\}$, $\Sigma_\mathcal{K} = \bigcup_{k\in\mathcal{K}} \Sigma_k$, and $Q_\mathcal{K} = \prod_{k\in\mathcal{K}} Q_k$. For a set of synchronizations $\delta \subseteq \Delta(\mathcal{S})$, its restriction $\delta^\mathcal{K} \subseteq Q_\mathcal{K} \times \Sigma_\mathcal{K} \times Q_\mathcal{K}$ to $\mathcal{S}_\mathcal{K}$ is defined by $\delta^\mathcal{K} = \{(\mathrm{proj}_\mathcal{K}(q), a, \mathrm{proj}_\mathcal{K}(q')) \mid (q,a,q') \in \delta\} \cap \Delta(\mathcal{S}_\mathcal{K})$. For $(\delta^\mathcal{K})_a = (\delta_a)^\mathcal{K}$ we will simply write $\delta^\mathcal{K}_a$.

For the rest of this section we let $\mathcal{K}$ be an arbitrary, but fixed subset of $\mathcal{I}$.

Definition 8 *Let $\mathcal{S}$ be composable and $\mathcal{T}$ a team automaton over $\mathcal{S}$, with set of transitions δ. The* subteam $SUB_\mathcal{K}(\mathcal{T})$ *of $\mathcal{T}$* determined by $\mathcal{K}$ *is the team automaton over $\mathcal{S}_\mathcal{K}$ with set of transitions $\delta^\mathcal{K}$.*

Note that for a singleton set $\{k\}$, with $k \in \mathcal{I}$, the subteam $SUB_{\{k\}}(\mathcal{T})$ is not the same as the component automaton C_k. For one, $SUB_{\{k\}}(\mathcal{T})$ has $\prod_{j\in\{k\}} Q_j$ rather than Q_k as its set of states. Still, even if we identified a singleton cartesian product with its element, then in general the set of transitions $\delta^{\{k\}}$ of $SUB_{\{k\}}(\mathcal{T})$ would only be a subset of δ^k (see also Example 10).

When applying the idea of a subteam to a team I/O automaton, in order to be useful the result should again be a team I/O automaton. It is immediate that every subset of a compatible set of component automata is again compatible. Moreover, if all component automata are input-enabled, then obviously also every subset consists of input-enabled components. Hence, the only thing left to establish is that the transitions inherited by a subteam of a synchronous product automaton form again a synchronous product, now of the transitions from the components considered: $(\chi^\mathcal{S})^\mathcal{K} = \chi^{\mathcal{S}_\mathcal{K}}$.

In [2] it has been shown, using a different terminology, that at least the transitions of a subteam of a synchronous product automaton will always be of the right type. Hence:

Lemma 9 $(\chi^\mathcal{S})^\mathcal{K} \subseteq \chi^{\mathcal{S}_\mathcal{K}}$.

An equality does not necessarily hold, as is demonstrated by the next example.

Example 10 *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be component automata sharing the external ac-*

tion a. $\mathcal{C}_1$ has a transition (p,a,p) while $\mathcal{C}_2$ does not have any a-transitions. Let $\mathcal{Z} = \{\mathcal{C}_1, \mathcal{C}_2\}$. Then clearly $(\chi^{\mathcal{Z}})_a = \varnothing$. Consequently, $(\chi^{\mathcal{Z}})_a^{\{1\}} = \varnothing$, but $(\chi^{\{\mathcal{C}_1\}})_a = \{(p,a,p')\}$.

An auxiliary condition is needed to guarantee that all necessary transitions are present in the subteam. The example shows that the subteam may miss a transition because it has no extension in the full synchronous product. As proven next, this can only be the case if the corresponding action is shared with some component from outside the subteam that does not use that action (i.e. it does not have a transition labeled by that action).

Lemma 11 *Let $a \in \Sigma_{\mathcal{K}}$ be such that $(\chi^{\mathcal{S}_{\mathcal{K}}})_a \neq \varnothing$. Then $(\chi^{\mathcal{S}_{\mathcal{K}}})_a \subseteq (\chi^{\mathcal{S}})_a^{\mathcal{K}}$ if and only if $(\delta^j)_a \neq \varnothing$ for all $j \in \mathcal{I} \setminus \mathcal{K}$ such that $a \in \Sigma_j$.*

PROOF. If there exists a $j \in \mathcal{I} \setminus \mathcal{K}$ such that $a \in \Sigma_j$ and $(\delta^j)_a = \varnothing$, then $(\chi^{\mathcal{S}})_a = \varnothing$ and thus also $(\chi^{\mathcal{S}})_a^{\mathcal{K}} = \varnothing$. Hence $(\chi^{\mathcal{S}_{\mathcal{K}}})_a \nsubseteq (\chi^{\mathcal{S}})_a^{\mathcal{K}}$. To prove the if-direction, let $(p,a,p') \in \chi^{\mathcal{S}_{\mathcal{K}}}$. Let $\mathcal{J} = \{j \in \mathcal{K} \mid a \in \Sigma_j\}$. Then, by definition of the synchronous product, $(\mathrm{proj}_j(p), a, \mathrm{proj}_j(p')) \in \delta^j$, for all $j \in \mathcal{J}$, and $\mathrm{proj}_i(p) = \mathrm{proj}_i(p')$, for all $i \in \mathcal{K} \setminus \mathcal{J}$. Let $\mathcal{J}' = \{j \in \mathcal{I} \setminus \mathcal{K} \mid a \in \Sigma_j\}$. Assume $(\delta^j)_a \neq \varnothing$, for all $j \in \mathcal{J}'$. Then, for each $j \in \mathcal{J}'$, we can fix a pair $p_j, p'_j \in Q_j$ such that $(p_j, a, p'_j) \in \delta^j$. Let $q, q' \in \prod_{i \in \mathcal{I}} Q_i$ be such that $\mathrm{proj}_j(q) = \mathrm{proj}_j(p)$ and $\mathrm{proj}_j(q') = \mathrm{proj}_j(p')$ for all $j \in \mathcal{K}$; $\mathrm{proj}_j(q) = p_j$ and $\mathrm{proj}_j(q') = p'_j$ for all $j \in \mathcal{J}'$; and $\mathrm{proj}_i(q) = \mathrm{proj}_i(q')$ for all $i \in \mathcal{I} \setminus \mathcal{K}$ such that $a \notin \Sigma_i$. Hence $(q,a,q') \in (\chi^{\mathcal{S}})_a$ and $(p,a,p') = (\mathrm{proj}_{\mathcal{K}}(q), a, \mathrm{proj}_{\mathcal{K}}(q')) \in (\chi^{\mathcal{S}})_a^{\mathcal{K}}$. □

Combining Lemmata 9 and 11 yields the following result.

Theorem 12 $(\chi^{\mathcal{S}})^{\mathcal{K}} = \chi^{\mathcal{S}_{\mathcal{K}}}$ *if and only if for all $a \in \Sigma_{\mathcal{K}}$, either $(\chi^{\mathcal{S}_{\mathcal{K}}})_a = \varnothing$ or $(\delta^j)_a \neq \varnothing$ for all $j \in \mathcal{I} \setminus \mathcal{K}$ such that $a \in \Sigma_j$.*

Hence the synchronous product defined by a subset of the components coincides with the restriction of the full synchronous product, if every action shared with some "outside" components either has no synchronizations in the subset itself or is used by each of these outside components. For a composable set of component automata it thus follows that the synchronous product is always preserved for the internal actions, since they are never shared. If the component automata are moreover input-enabled (I/O automata), then the synchronous product is also preserved for those input actions of a subteam which have transitions in each of the outside components in which they occur as output. Since in a compatible set of I/O automata, every output action occurs in only one component as output, we can derive the following conclusion:

Theorem 13 *Let $\mathcal{S}$ be a compatible set of I/O automata.*
Then $SUB_{\mathcal{K}}(\mathcal{X}(\mathcal{S})) = \mathcal{X}(\mathcal{S}_{\mathcal{K}})$, the team I/O automaton over $\mathcal{S}_{\mathcal{K}}$, if and only if $\delta^j_a \neq \varnothing$, for all $a \in \Sigma_{\mathcal{K}} \cap \Sigma_{out}$ and $j \in \mathcal{I} \setminus \mathcal{K}$ such that $a \in \Sigma_{j,out}$.
Also, $SUB_{\mathcal{K}}(\mathcal{X}(\mathcal{S}))$ is the team I/O automaton over $\mathcal{S}_{\mathcal{K}}$ and $SUB_{\mathcal{I} \setminus \mathcal{K}}(\mathcal{X}(\mathcal{S}))$

is the team I/O automaton over $\mathcal{S}_{\mathcal{I}\setminus\mathcal{K}}$ if and only if $\delta_a^j \neq \varnothing$, for all $a \in \bigcup_{i\in\mathcal{I}} \Sigma_{i,inp} \cap \Sigma_{out}$ and $j \in \mathcal{I}$ such that $a \in \Sigma_{j,out}$.

In other words, a subteam of a team I/O automaton is again a team I/O automaton if and only if output of the full team corresponding to input for the subteam is used by the component in which it occurs as an output action. Moreover, to guarantee that every subteam of a team I/O automaton is a team I/O automaton, each output action intended for communication within the team (as it also occurs as input for some components), should have a transition in the component in which it occurs in its output role. While this condition may seem mathematically involved, it turns out to be so natural that for a design to make sense it is practically always satisfied.

3.2 Superteams

In [2] it is shown how team automata can be used to describe hierarchical systems through iteration and how subteams can be considered as component automata in such iteratively defined team automata. Moreover, iteration in the construction of a team automaton does not lead to additional possibilities for synchronization, i.e. every iterated team automaton over a composable set of component automata can be interpreted as being directly defined from those components. From Theorem 7 we know that also team I/O automata can be used as components in a hierarchical construction. In the remainder we outline the approach followed for team automata and then restrict it to the case of I/O automata. This yields precise structural definitions for iterated team I/O automata, which contrast with the usual approach that focuses on behavior and is mostly restricted to binary rather than arbitrary iterations.

Given a composable set of component automata, there are in general different routes possible to form a team automaton. Rather than directly defining a team over all components available, one can also first describe collaborations between certain components before merging these into a higher-level construct. Hence first teams over (disjoint) subsets of components are defined and then these can be used (iteratively) as components in new teams, until after a finite number of such iterations all components have been used. This implies that the composable set is partitioned into subsets, each of which forms the basis of an (iterated) team automaton.

A *partition* of a set W is a collection of sets $\{W_j \mid j \in \mathcal{J}\}$, with $\mathcal{J} \subseteq \mathbb{N}$, such that the W_j are nonempty and pairwise disjoint, and $\bigcup_{j\in\mathcal{J}} W_j = W$.

As observed before, any subset of a composable set is again composable. Furthermore, in [2] it was shown that since composition does not introduce new internal actions and internal actions are never shared, team automata over disjoint subsets of a composable set also form a composable set:

Lemma 14 *Let $\mathcal{S}$ be composable and $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ a partition of $\mathcal{I}$. For all $j \in \mathcal{J}$, let $\mathcal{T}_j$ be a team automaton over $\mathcal{S}_j = \{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$. Then $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ is composable.*

Team automata can thus be defined iteratively from a given composable set.

Definition 15 *Let $\mathcal{S}$ be composable. $\mathcal{T}$ is an* iterated team automaton *over $\mathcal{S}$ if either (1) $\mathcal{T}$ is a team automaton over $\mathcal{S}$ or (2) there exists a partition $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ of $\mathcal{I}$ such that $\mathcal{T}$ is a team automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, where each $\mathcal{T}_j$ is an iterated team automaton over $\{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$.*

In [2] it is shown that any iterated team automaton over a composable set can be viewed as a (directly defined) team automaton over that set: Its set of actions—including the distribution over input, output, and internal actions—is the one defined by the composable set, just as for any ordinary team automaton over that same set of components. Its (initial) states are "essentially" the (initial) states defined by the composable set, where "essentially" means upto (unpacking and) reordering the nested components. Similarly, its transitions are "essentially" synchronizations of the composable set in the sense that only their state components have to be reordered. Rearranging the state space is thus sufficient to consider an iterated team automaton as an ordinary team automaton. Before continuing, we give an outline of the process of reordering. [3]

Assume that $\mathcal{S}$ is composable and let $\mathcal{T}$ be an iterated team automaton over $\mathcal{S}$. Reordering is defined bottom-up from leaves to root in the ordered tree describing the iteration and assumes that the relevant partitions are given. At the lowest level we have the leaves corresponding to component automata. One level higher we have team automata over subsets of $\mathcal{S}$. Note that the order of the component automata as used in the definition of these team automata still corresponds to their original order in $\mathcal{S}$. Hence, at this level, reordering has no effect: Let $\mathcal{L} \subseteq \mathcal{I}$ and $\mathcal{T}_\mathcal{L}$ a team automaton over $\mathcal{S}_\mathcal{L} = \{\mathcal{C}_\ell \mid \ell \in \mathcal{L}\}$. For $p \in Q_\mathcal{L} = \prod_{\ell \in \mathcal{L}} Q_\ell$, its reordered version with respect to $Q_\mathcal{L}$ is $\langle p \rangle_{Q_\mathcal{L}} = p$ and the reordered version of $\mathcal{T}_\mathcal{L}$ with respect to $\mathcal{S}_\mathcal{L}$ is $\langle\langle \mathcal{T}_\mathcal{L} \rangle\rangle_{\mathcal{S}_\mathcal{L}} = \mathcal{T}_\mathcal{L}$. At all higher levels we deal with an $\mathcal{L} \subseteq \mathcal{I}$ and an iterated team automaton $\mathcal{T}_\mathcal{L}$ over $\mathcal{S}_\mathcal{L} = \{\mathcal{C}_\ell \mid \ell \in \mathcal{L}\}$, a subset of $\mathcal{S}$. Moreover, we have a partition $\{\mathcal{L}_j \mid j \in \mathcal{J}\}$ of $\mathcal{L}$ such that $\mathcal{T}_\mathcal{L}$ is a team automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, where each $\mathcal{T}_j$ is an iterated team automaton over $\mathcal{S}_{\mathcal{L}_j} = \{\mathcal{C}_\ell \mid \ell \in \mathcal{L}_j\}$. On basis of the construction we may assume that, for all $j \in \mathcal{J}$, for each of the states p of $\mathcal{T}_j$, its reordered version $\langle p \rangle_{Q_{\mathcal{L}_j}} \in Q_{\mathcal{L}_j} = \prod_{\ell \in \mathcal{L}_j} Q_\ell$ has been established. Then, for each state $q = \prod_{j \in \mathcal{J}} q_j$ of $\mathcal{T}_\mathcal{L}$, where for each j, q_j is a state of $\mathcal{T}_j$ we define the reordered version of q with respect to $Q_\mathcal{L}$ as $\langle q \rangle_{Q_\mathcal{L}} = \prod_{\ell \in \mathcal{L}} p_\ell$ with $p_\ell = \mathrm{proj}_\ell(\langle q_j \rangle_{Q_{\mathcal{L}_j}})$, where j is such that $\ell \in \mathcal{L}_j$. The reordered version $\langle\langle \mathcal{T}_\mathcal{L} \rangle\rangle_{\mathcal{S}_\mathcal{L}}$ of $\mathcal{T}_\mathcal{L}$ with respect to $\mathcal{S}_\mathcal{L}$ is the team automaton over $\mathcal{S}_\mathcal{L}$, with set

[3] A simpler and more concrete description of the abstract reordering defined in [2].

of transitions $\langle\delta\rangle_{Q_\mathcal{L}} = \{(\langle q\rangle_{Q_\mathcal{L}}, a, \langle q'\rangle_{Q_\mathcal{L}}) \mid (q,a,q') \in \delta\}$, where δ is the set of transitions of $\mathcal{T}_\mathcal{L}$.

All this leads to the following formalization, as proven in [2], of our earlier observation that every iterated team automaton over a composable set of component automata can be interpreted as a team automaton over that set by reordering its state space:

Theorem 16 *Let $\mathcal{S}$ be composable and $\widehat{\mathcal{T}}$ an iterated team automaton over $\mathcal{S}$. Then $\widehat{\mathcal{T}}$ is a transition system $(\widehat{Q}, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, \widehat{I})$ and its reordered version $\langle\langle\widehat{\mathcal{T}}\rangle\rangle_\mathcal{S} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \langle\delta\rangle_Q, I)$ is a team automaton over $\mathcal{S}$.*

Conversely, every team automaton can be seen—after reordering—as being composed of any (disjoint) combination of its subteams, i.e. any (iterated) decomposition can be used as a constructive description of the team. Obviously, in such iterative constructions care should be taken to include "enough" transitions. Formally, a composable set $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ consisting of iterated team automata over subsets $\mathcal{S}_j$ of the composable set $\mathcal{S}$, where $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ is a partition of $\mathcal{I}$, may provide less transitions for the forming of a team than $\mathcal{S}$ does. To define a given team $\mathcal{T}$ it is however always sufficient to require that each of the $\mathcal{T}_j$ has at least all transitions—upto reordering—of the subteam of $\mathcal{T}$ determined by $\mathcal{I}_j$. In that case, as proven in [2], one can define the team automaton $\widehat{\mathcal{T}}$ over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ such that $\langle\langle\widehat{\mathcal{T}}\rangle\rangle_\mathcal{S} = \mathcal{T}$:

Theorem 17 *Let $\mathcal{S}$ be composable and $\mathcal{T}$ a team automaton over $\mathcal{S}$. Let $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ be a partition of $\mathcal{I}$ and, for all $j \in \mathcal{J}$, $\mathcal{T}_j$ an iterated team automaton over $\mathcal{S}_{\mathcal{I}_j} = \{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$ with set γ^j of transitions such that $\delta^{\mathcal{I}_j} \subseteq \langle\gamma^j\rangle_{Q_{\mathcal{I}_j}}$. Then there exists a team automaton $\widehat{\mathcal{T}}$ over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ such that $\langle\langle\widehat{\mathcal{T}}\rangle\rangle_\mathcal{S} = \mathcal{T}$.*

Now we turn to synchronous products as a first step towards the iterated construction of team I/O automata. Recall that a subteam of a synchronous product automaton need not be a synchronous product automaton itself, because some transitions may be missing. Theorem 17 however allows us to prove that synchronous product automata can still be seen as constructed iteratively as a synchronous product of components that are synchronous products.

Lemma 18 *Let $\mathcal{S}$ be composable and $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ a partition of $\mathcal{I}$. For all $j \in \mathcal{J}$, let $\mathcal{T}_j$ be an iterated team automaton over $\mathcal{S}_{\mathcal{I}_j} = \{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$ such that $\langle\langle\mathcal{T}_j\rangle\rangle_{\mathcal{S}_{\mathcal{I}_j}} = \mathcal{X}(\mathcal{S}_{\mathcal{I}_j})$. Then $\langle\langle\mathcal{X}(\{\mathcal{T}_j \mid j \in \mathcal{J}\})\rangle\rangle_\mathcal{S} = \mathcal{X}(\mathcal{S})$.*

PROOF. By Lemma 9 and Theorem 17, there exists a team automaton $\mathcal{T}$ over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ such that $\langle\langle\mathcal{T}\rangle\rangle_\mathcal{S} = \mathcal{X}(\mathcal{S})$. Hence once we have proven that $\mathcal{T} = \mathcal{X}(\{\mathcal{T}_j \mid j \in \mathcal{J}\})$ must hold, we are done. First assume that $\mathcal{T}$ has a transition (p, a, p') which is not in $\chi^{\{\mathcal{T}_j \mid j \in \mathcal{J}\}}$, i.e. some $\mathcal{T}_j$ is not involved in this transition while a is an action of this $\mathcal{T}_j$. Thus a is an action of a $\mathcal{C}_i$ with $i \in \mathcal{I}_j$, and this $\mathcal{C}_i$ is not involved in the reordered version $(\langle p \rangle_Q, a, \langle p' \rangle_Q)$ of (p, a, p'). Consequently $(\langle p \rangle_Q, a, \langle p' \rangle_Q) \notin \chi^{\mathcal{S}}$, a contradiction with $\langle\langle \mathcal{T} \rangle\rangle_{\mathcal{S}} = \mathcal{X}(\mathcal{S})$. Next assume that $(p, a, p') \in \chi^{\{\mathcal{T}_j | j \in \mathcal{J}\}}$. Let $j \in \mathcal{J}$ be such that a is an action of $\mathcal{T}_j$. Then $\mathcal{T}_j$ is involved in this transition. Moreover, since $\langle\langle \mathcal{T}_j \rangle\rangle_{\mathcal{S}_{\mathcal{I}_j}} = \mathcal{X}(\mathcal{S}_{\mathcal{I}_j})$, we know that every component $\mathcal{C}_i$ with $i \in \mathcal{I}_j$ which has a as an action is involved in $(\mathrm{proj}_j(p), a, \mathrm{proj}_j(p'))$. Consequently, the reordered version of (p, a, p') with respect to Q is in $\chi^{\mathcal{S}}$ and (p, a, p') is thus a transition of $\mathcal{T}$. Hence $\mathcal{T} = \mathcal{X}(\{\mathcal{T}_j \mid j \in \mathcal{J}\})$, as required. □

Now we propose a definition of iterated team I/O automata similar to that of iterated team automata (cf. Def. 15). Note that, as remarked before, input-enabledness is preserved by synchronous products and hence by iterated synchronous product constructions (cf. Theorem 7). Moreover, like composability obviously also compatibility is preserved when iteratively forming teams:

Lemma 19 *Let $\mathcal{S}$ be compatible and $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ a partition of $\mathcal{I}$. For all $j \in \mathcal{J}$, let $\mathcal{T}_j$ be team automaton over $\mathcal{S}_{\mathcal{I}_j} = \{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$. Then $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ is compatible.*

Definition 20 *Let $\mathcal{S}$ be compatible. $\mathcal{T}$ is an* iterated team I/O automaton *over $\mathcal{S}$ if either (1) $\mathcal{T}$ is the team I/O automaton over $\mathcal{S}$ or (2) there exists a partition $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ of $\mathcal{I}$ such that $\mathcal{T}$ is the team I/O automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, where each $\mathcal{T}_j$ is an iterated team I/O automaton over $\{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$.*

From Lemma 18 it follows that iteratively applying a synchronous product to construct higher-level team automata yields—upto reordering—a synchronous product automaton over the original components. This allows us to demonstrate that the reordered version of any (whatever route has been followed) iterated team over a compatible set of I/O automata is *the* team I/O automaton over these components.

Theorem 21 *Let $\mathcal{S}$ be compatible and $\mathcal{T}$ an iterated team I/O automaton over $\mathcal{S}$. Then $\langle\langle \mathcal{T} \rangle\rangle_{\mathcal{S}} = \mathcal{X}(\mathcal{S})$.*

PROOF. If $\mathcal{T}$ is constructed directly from $\mathcal{S}$ without iteration, then $\langle\langle \mathcal{T} \rangle\rangle_{\mathcal{S}} = \mathcal{T} = \mathcal{X}(\mathcal{S})$. Otherwise, $\mathcal{T}$ is the team I/O automaton over a compatible set $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, with $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ a partition of $\mathcal{I}$, such that each $\mathcal{T}_j$ is an iterated team I/O automaton over the compatible set $\mathcal{S}_{\mathcal{I}_j} = \{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$. By induction, we assume that $\langle\langle \mathcal{T}_j \rangle\rangle_{\mathcal{S}_j} = \mathcal{X}(\mathcal{S}_{\mathcal{I}_j})$. Then Lemma 18 implies that $\langle\langle \mathcal{T} \rangle\rangle_{\mathcal{S}} = \mathcal{X}(\mathcal{S})$, as desired. □

As for team automata, Lemma 18 implies that also every team I/O automaton can be seen—after reordering—as being iteratively constructed from its subteams, provided that each of its subteams is itself a team I/O automaton as formally described in Theorem 13.

4 Discussion

We have embedded the (structural) setup of I/O automata in the framework of team automata. Whereas in the theory of I/O automata the notions of input-enabledness, compatibility, and synchronous product are always used in combination—and together reflect a certain viewpoint on the interaction of reactive, distributed systems—they are mathematically independent. In the team automata framework these notions can be studied independently of one another. Among other things this leads to more general results like, e.g., Lemma 6, to the formal distinction between team I/O automata and synchronous products of I/O automata, and to precise notions of sub- and superteams.

The emphasis in this paper has been on the structure of team (I/O) automata. In fact, we have not dealt at all with their behavioral aspects. After all, since the behavior of team automata and I/O automata is based on the same notion of computations (cf., e.g., [2,17]) the structural relation is the more fundamental one. A consequence of our focus on the structural setup of I/O automata is that we have not considered the notion of *fair* computations or behavior. In fact, the definition of I/O automata given here corresponds to I/O automata that are called *unfair* [7] or *safe* [22]. Consequently, also the notion of *strong* compatibility—by which no action may belong to infinitely many components—is not significant to our exposition.

Finally, we conclude by a remark on the relation between Petri nets and team (I/O) automata. I/O automata can automatically be seen as a type of Petri nets, but this is not the case for team automatain general [1,4]. The reason is the fact that I/O automata are *non-state-sharing*, whereas team automata are not. This notion is one of the fundamental concepts underlying Petri nets and it requires that whether or not a synchronization between components can take place, only depends on the local states of the components actively involved in that synchronization (and thus not on the global state as a whole).

References

[1] M.H. ter Beek, *Team Automata—A Formal Approach to the Modeling of Collaboration Between System Components*, PhD thesis (LIACS, Leiden University, 2003).

[2] 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.

[3] M.H. ter Beek and J. Kleijn, Team Automata Satisfying Compositionality, in: K. Araki, S. Gnesi, and D. Mandrioli, eds., *Proceedings FM 2003, LNCS* 2805 (Springer-Verlag, Berlin, 2003) 381–400.

[4] J. Carmona and J. Kleijn, Interactive Behaviour of Multi-Component Systems, in: J. Cortadella and A. Yakovlev, eds., *Proceedings ToBaCo'04* (University of Bologna, 2004) 27–31.

[5] C.A. Ellis, Team Automata for Groupware Systems, in: S.C. Hayne and W. Prinz, eds., *Proceedings GROUP'97* (ACM Press, New York, 1997) 415–424.

[6] S.J. Garland and N.A. Lynch, The IOA Language and Toolset—Support for Designing, Analyzing, and Building Distributed Systems, Technical Report MIT/LCS/TR-762, MIT, 1998.

[7] R. Gawlick, R. Segala, F.F. Søgaard-Andersen, and N.A. Lynch, Liveness in Timed and Untimed Systems, in: S. Abiteboul and E. Shamir, eds., *Proceedings ICALP'94, LNCS* 820 (Springer-Verlag, Berlin, 1994) 166–177.

[8] B. Grobauer and O. Müller, From I/O Automata to Timed I/O Automata—A Solution to the 'Generalized Railroad Crossing' in Isabelle/HOLCF, in: Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, eds., *Proceedings TPHOLs'99, LNCS* 1690 (Springer-Verlag, Berlin, 1999) 273–289.

[9] D.K. Kaynar, N.A. Lynch, R. Segala, and F.W. Vaandrager, Timed I/O Automata: A Mathematical Framework for Modeling and Analyzing Real-Time Systems, in: S.K. Baruah and R. Rajkumar, eds., *Proceedings RTSS'03* (IEEE Computer Society Press, New York, 2003) 166–177.

[10] N.A. Lynch, I/O Automata: A Model for Discrete Event Systems, in: P.J. Ramadge and S. Verdú, eds., *Proceedings CISS'88* (Princeton University Press, Princeton, NJ, 1988) 29–38.

[11] N.A. Lynch, *Distributed Algorithms* (Morgan Kaufmann, San Mateo, CA, 1996).

[12] N.A. Lynch, I/O Automaton Models and Proofs for Shared-Key Communication Systems, in: P. Syverson, ed., *Proceedings CSFW'99* (IEEE Computer Society Press, New York, 1999) 14–31.

[13] N.A. Lynch, Input/Output Automata: Basic, Timed, Hybrid, Probabilistic, Dynamic,..., in: R. Amadio and D. Lugiez, eds., *Proceedings CONCUR'03, LNCS* 2761 (Springer-Verlag, Berlin, 2003) 191–192.

[14] N.A. Lynch and M. Merritt, Introduction to the Theory of Nested Transactions, in: G. Ausiello and P. Atzeni, eds., *Proceedings ICDT'86, LNCS* 243 (Springer-Verlag, Berlin, 1986) 278–305.

[15] N.A. Lynch and M. Merritt, Introduction to the Theory of Nested Transactions. *Theoretical Computer Science* **62, 1-2** (1988) 123–185.

[16] N.A. Lynch and M.R. Tuttle, Hierarchical Correctness Proofs for Distributed Algorithms, in: F.B. Schneider, ed., *Proceedings PODC'87* (ACM Press, New York, 1987) 137–151.

[17] N.A. Lynch and M.R. Tuttle, An Introduction to Input/Output Automata. *CWI Quarterly* **2, 3** (1989) 219–246.

[18] N.A. Lynch and F.W. Vaandrager, Forward and Backward Simulations for Timing-Based Systems, in: J.W. de Bakker, C. Huizing, W.-P. de Roever, and G. Rozenberg, eds., *Real-Time: Theory in Practice, LNCS* 600 (Springer-Verlag, Berlin, 2003) 397–446.

[19] O. Müller, *A Verification Environment for I/O Automata Based on Formalized Meta-Theory*, PhD thesis (Technische Universität München, 1998).

[20] J.M.T. Romijn, Tackling the RPC-Memory specification problem with I/O automata, in: M. Broy, S. Merz, and K. Spies, eds., *Formal Systems Specification—The RPC-Memory Specification Case Study, LNCS* 1169, (Springer-Verlag, Berlin, 1996) 437–476.

[21] R. Segala, *Modeling and Verification of Randomized Distributed Real-Time Systems*, PhD thesis (Department of Electrical Engineering and Computer Science, MIT, 1995).

[22] M.R. Tuttle, *Hierarchical Correctness Proofs for Distributed Algorithms*, Master's thesis (Department of Electrical Engineering and Computer Science, MIT, 1987).

[23] S.-H. Wu, S.A. Smolka, and E.W. Stark, Composition and Behaviors of Probabilistic I/O Automata. *Theoretical Computer Science* **176, 1-2** (1997) 1–38.