Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Processing Letters 95 (2005) 487–495
Information Processing Letters
www.elsevier.com/locate/ipl

Modularity for teams of I/O automata ☆

Maurice H. ter Beek [a,*], 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*

Received 21 September 2004; received in revised form 9 May 2005; accepted 20 May 2005

Available online 21 June 2005

Communicated by J.L. Fiadeiro

Keywords: Formal methods; Distributed systems; I/O automata; Team automata

1. Introduction

Input/Output automata (or I/O automata) were originally introduced by Tuttle and Lynch [10,13] as a model for distributed computations in asynchronous networks and as a means of constructing 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. They can be composed using a synchronous product construction yielding a new I/O automaton. Many variants of I/O automata were considered and the model is now widely used for describing reactive, distributed systems [6–8,10–13].

Inspired by I/O automata, Ellis introduced team automata 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 the first author's PhD Thesis [1] they were shown to provide a solid and general theoretical framework for the study of synchronization mechanisms in automata models. Ellis dropped a number of the restrictions of I/O automata to allow flexible modeling of various kinds of collaboration in groupware systems: Team automata impose hardly any restrictions on the role of the actions in the various components and their composition is not based on an a priori fixed way of synchronizing their actions. This makes it possible to define a wide variety of protocols for the interaction between a system and its environment.

It is our aim to establish a formal connection between I/O automata and team automata, easing the transfer of notions, techniques, and results. In Section 2 we first introduce team automata. Then, in Section 3, we demonstrate how I/O automata fit in the framework of team automata. We do this by studying

☆ This work was partially supported by the European Research Training Network SegraVis (Syntactic and Semantic Integration of Visual Modeling Techniques).

* Corresponding author.
E-mail address: maurice.terbeek@isti.cnr.it (M.H. ter Beek).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2005.05.012

separately several notions that are always used in combination for I/O automata: input-enabledness, compatibility, and synchronous product. This embedding of I/O automata in the framework of team automata is then used in Section 4 to investigate the modularity of iteratively defined I/O automata. In particular, we consider the notions of subteam and superteam as introduced for team automata [2] and investigate to what extent they can be applied to I/O automata. So far, these notions have not been considered explicitly for I/O automata. Section 5 concludes the paper.

2. Team 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. We recall some notions from [2].

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

To avoid technical anomalies, we always assume in this paper that the set of states and the set of initial states of a component automaton are nonempty. Let $\mathcal{I} \subseteq \mathbb{N}$ be a nonempty, possibly infinite, countable 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 $\prod_{i\in\mathcal{I}} V_i$. For each $j \in \mathcal{I}$ and $(v_{i_1}, v_{i_2}, \ldots) \in \prod_{i\in\mathcal{I}} V_i$, we define $\text{proj}_j((v_{i_1}, v_{i_2}, \ldots)) = v_j$. If $\emptyset \neq \mathcal{J} \subseteq \mathcal{I}$, then $\text{proj}_{\mathcal{J}}((v_{i_1}, v_{i_2}, \ldots)) = \prod_{j\in\mathcal{J}} v_j$.

For the sequel, we let $\mathcal{S} = \{\mathcal{C}_i \mid i \in \mathcal{I}\}$ with $\mathcal{I} \subseteq \mathbb{N}$ be a fixed nonempty, indexed set of component automata, in which each $\mathcal{C}_i$ is specified as $(Q_i, (\Sigma_{i,\text{inp}}, \Sigma_{i,\text{out}}, \Sigma_{i,\text{int}}), \delta^i, I_i)$, with $\Sigma_i = \Sigma_{i,\text{inp}} \cup \Sigma_{i,\text{out}} \cup \Sigma_{i,\text{int}}$ 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 $\mathcal{S}$.

Component automata interact by synchronizing on common actions. Not all automata sharing an action have to participate in each synchronization on that action. This leads to the notion of a complete transition space, consisting of all possible combinations of identically labeled transitions.

Definition 2. A transition $(q, a, q') \in Q \times \Sigma \times Q$ is a *synchronization* on a in $\mathcal{S}$ if for all $i \in \mathcal{I}$, $(\text{proj}_i(q), a, \text{proj}_i(q')) \in \delta^i$ or $\text{proj}_i(q) = \text{proj}_i(q')$, and there exists $i \in \mathcal{I}$ such that $(\text{proj}_i(q), a, \text{proj}_i(q')) \in \delta^i$.

For $a \in \Sigma$, $\Delta_a(\mathcal{S})$ is the set of all synchronizations on a in $\mathcal{S}$.

Finally, $\Delta(\mathcal{S}) = \bigcup_{a\in\Sigma} \Delta_a(\mathcal{S})$ is the set of all synchronizations of $\mathcal{S}$.

Given a set of component automata, different synchronizations can be chosen for the set of transitions of a composed automaton. Such an automaton has the Cartesian product of the states of the components as its states. To allow hierarchically constructed systems within the setup of team automata, a composed automaton also has internal, input, and output actions. 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 *composable* if

$$\Sigma_{i,\text{int}} \cap \bigcup_{j\in\mathcal{I}\setminus\{i\}} \Sigma_j = \emptyset \quad \text{for all } i \in \mathcal{I}.$$

Moreover, within a team automaton each internal action can be executed from a global state whenever it can be executed by its component at the current local state. All this is formalized as follows.

Definition 3. Let $\mathcal{S}$ be composable. Then a *team automaton* over $\mathcal{S}$ is a transition system $\mathcal{T} = (Q, (\Sigma_{\text{inp}}, \Sigma_{\text{out}}, \Sigma_{\text{int}}), \delta, I)$, 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 $\Sigma_{\text{int}} = \bigcup_{i\in\mathcal{I}} \Sigma_{i,\text{int}}$, $\Sigma_{\text{out}} = \bigcup_{i\in\mathcal{I}} \Sigma_{i,\text{out}}$, and $\Sigma_{\text{inp}} = (\bigcup_{i\in\mathcal{I}} \Sigma_{i,\text{inp}}) \setminus \Sigma_{\text{out}}$; 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 $a \in \Sigma_{\text{int}}$.

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 particular synchronizations on that action, possibly depending on its different roles, is left to the designer of an application.[1]

3. I/O automata as team automata

The theory of I/O automata also uses component automata as basic units [7,11,13]. Composition is, however, based on assumptions about how reactive, distributed systems behave and interact. It is *intended* (see, e.g., the Introduction of [11]) to model communication among components rather than arbitrary interaction. This leads to restrictions on the input and output roles an external action may have while assuming an a priori fixed way of synchronizing these actions. We first define this fixed composition, which 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:

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

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 Definition 3. In particular, we have $(\chi^{\mathcal{S}})_a = \Delta_a(\mathcal{S})$ for every internal action a.

[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.

A synchronous product automaton thus has as its transitions all and only those synchronizations on an action that involve all components sharing that action. A synchronization on an action that 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* [13]. This means 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 components that are not ready for this communication.

Example 4. Let $\mathcal{C}_1$ be a component automaton with output action a and the single transition (p,a,p). Let $\mathcal{C}_2$ be a component automaton with initial state q, input actions $\{a,b\}$, and transitions $\{(q,b,r),(r,a,q)\}$. Then $\mathcal{S} = \{\mathcal{C}_1,\mathcal{C}_2\}$ is composable and $\chi^{\mathcal{S}} = \{((p,q), b, (p,r)), ((p,r), a, (p,q))\}$. Now $\mathcal{C}_1$ has no local control over its output action a: It cannot execute a when $\mathcal{C}_2$ is in state q and unable to receive input a.

Consequently, only component automata are considered which are *input-enabled*: each input action is enabled at each state.

Definition 5. An *I/O automaton* is a component automaton that is input-enabled.

This is the standard definition of an I/O automaton, except for the omission of an equivalence relation used only to deal with fairness in computations but otherwise ignored. I/O automata as defined here are sometimes called *unfair* [6] or *safe* [13].

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 we now prove in a more general setting. First we formalize the intuitive concept of local control by 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 $\text{proj}_i(q) = p$, there exists $(q,a,q') \in \delta$ such that $\text{proj}_i(q') = p'$. 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 $\mathcal{C}_i$ 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 this definition implies that the internal actions of any team automaton are under the local control of the component to which they belong. We now show a general relationship between omnipresence and enabling:

Lemma 6. *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. We prove only the if-direction. 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 $\chi^{\mathcal{S}}$. □

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 case of team automata by not imposing this restriction on the set of transitions of the composed automaton. Within the theory of I/O automata, however, components are simply beforehand forbidden to share output actions by imposing an additional restriction on the set of component automata:

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

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

Lemma 6 implies that the internal and output actions in the synchronous product automaton over a compatible set of I/O automata are indeed locally controlled:

Theorem 7. *Let $\mathcal{S}$ be a compatible set of I/O automata. Let $i \in \mathcal{I}$ and a be an internal or output action of $\mathcal{C}_i$. Then a is locally controlled by $\mathcal{C}_i$ in $\mathcal{X}(\mathcal{S})$.*

This was found to be an important property already in the earliest versions of I/O automata [9,13].

In order to ensure that the composition of I/O automata is again an I/O automaton, the synchronous product construction should preserve input-enabledness. In fact, a more general property can be derived from Lemma 6: Any set of transitions which contains the synchronous product preserves the enabling property and compatibility is not needed.

Theorem 8. *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 $q' \in Q$ such that $(q, a, q') \in \delta$.*

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 6 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 $(q, a, q') \in \delta$. □

By observing that the input actions of a team automaton occur only as input actions in its components, we obtain as a corollary of Theorem 8 that the synchronous product construction preserves input-enabledness:

Corollary 9. *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. This generalizes the approach of the theory of I/O automata, where only compatible sets of I/O automata are used to compose new I/O automata.

Definition 10. Let $\mathcal{S}$ be a compatible set of I/O automata. Then $\mathcal{X}(\mathcal{S})$ is the *team I/O automaton* over $\mathcal{S}$.

Clearly every team I/O automaton is an I/O automaton by Corollary 9.

4. Modular constructions: subteams and superteams

From the previous section we know that both team automata and team I/O automata can be used as components to define higher-level teams. It thus is feasible to design systems in a modular, iterative fashion. Conversely, an appropriate notion of subautomaton would make it possible to decompose systems 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 properties of hierarchically defined systems from their components. This is done for the framework of team automata in [1–3], where subteams and iterated teams have been defined and related to one another. In this section we investigate to what extent the definitions of subteam and iterated composition for team automata can be applied successfully to I/O automata.

4.1. 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 $\mathcal{K} \subseteq \mathcal{I}$ be nonempty. 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

$$\delta^\mathcal{K} = \big\{\big(\mathrm{proj}_\mathcal{K}(q), a, \mathrm{proj}_\mathcal{K}(q')\big) \mid (q, a, q') \in \delta\big\} \cap \Delta(\mathcal{S}_\mathcal{K}).$$

For $(\delta^\mathcal{K})_a = (\delta_a)^\mathcal{K}$ we simply write $\delta_a^\mathcal{K}$. For the rest of this section we let $\mathcal{K}$ be an arbitrary, but fixed nonempty subset of $\mathcal{I}$.

Definition 11. Let $\mathcal{S}$ be composable and $\mathcal{T}$ be a team automaton over $\mathcal{S}$, with set of transitions δ. The *subteam* $\mathrm{SUB}_\mathcal{K}(\mathcal{T})$ 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 $\mathrm{SUB}_{\{k\}}(\mathcal{T})$ is not the same as the component automaton $\mathcal{C}_k$. For one, $\mathrm{SUB}_{\{k\}}(\mathcal{T})$ 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 $\mathrm{SUB}_{\{k\}}(\mathcal{T})$ would only be a subset of δ^k (see also Example 13).

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, though—that at least the transitions of a subteam of a synchronous product automaton will always be of the right type. Hence:

Lemma 12. $(\chi^\mathcal{S})^\mathcal{K} \subseteq \chi^{\mathcal{S}_\mathcal{K}}$.

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

Example 13. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be component automata sharing the external action 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 $(\chi^\mathcal{Z})_a = \emptyset$. Consequently, $(\chi^\mathcal{Z})_a^{\{1\}} = \emptyset$, 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 14. *Let $a \in \Sigma_{\mathcal{K}}$ be such that $(\chi^{\mathcal{S}_{\mathcal{K}}})_a \neq \emptyset$. Then $(\chi^{\mathcal{S}_{\mathcal{K}}})_a \subseteq (\chi^{\mathcal{S}})^{\mathcal{K}}_a$ if and only if $(\delta^j)_a \neq \emptyset$ for all $j \in \mathcal{I} \setminus \mathcal{K}$ such that $a \in \Sigma_j$.*

Proof. If there exists $j \in \mathcal{I} \setminus \mathcal{K}$ such that $a \in \Sigma_j$ and $(\delta^j)_a = \emptyset$, then $(\chi^{\mathcal{S}})_a = \emptyset$ and thus also $(\chi^{\mathcal{S}})^{\mathcal{K}}_a = \emptyset$. Hence $(\chi^{\mathcal{S}_{\mathcal{K}}})_a \not\subseteq (\chi^{\mathcal{S}})^{\mathcal{K}}_a$. 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 $(\delta^j)_a \neq \emptyset$, 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 $(p, a, p') = (\mathrm{proj}_{\mathcal{K}}(q), a, \mathrm{proj}_{\mathcal{K}}(q')) \in (\chi^{\mathcal{S}})^{\mathcal{K}}_a$. □

Combining Lemmas 12 and 14 yields the following result.

Theorem 15. *$(\chi^{\mathcal{S}})^{\mathcal{K}} = \chi^{\mathcal{S}_{\mathcal{K}}}$ if and only if for all $a \in \Sigma_{\mathcal{K}}$, $(\chi^{\mathcal{S}_{\mathcal{K}}})_a = \emptyset$ or $(\delta^j)_a \neq \emptyset$ 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 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 conclude that:

Corollary 16. *Let $\mathcal{S}$ be a compatible set of I/O automata.*

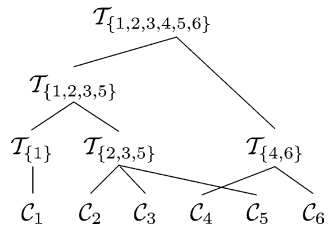
Then $\mathrm{SUB}_{\mathcal{K}}(\mathcal{X}(\mathcal{S})) = \mathcal{X}(\mathcal{S}_{\mathcal{K}})$, the team I/O automaton over $\mathcal{S}_{\mathcal{K}}$, if and only if $\delta^j_a \neq \emptyset$, for all $a \in \Sigma_{\mathcal{K}} \cap \Sigma_{\mathrm{out}}$ and $j \in \mathcal{I} \setminus \mathcal{K}$ such that $a \in \Sigma_{j,\mathrm{out}}$.

If $\mathcal{K} \neq \mathcal{I}$, then $\mathrm{SUB}_{\mathcal{K}}(\mathcal{X}(\mathcal{S}))$ is the team I/O automaton over $\mathcal{S}_{\mathcal{K}}$ and $\mathrm{SUB}_{\mathcal{I}\setminus\mathcal{K}}(\mathcal{X}(\mathcal{S}))$ is the team I/O automaton over $\mathcal{S}_{\mathcal{I}\setminus\mathcal{K}}$ if and only if $\delta^j_a \neq \emptyset$, for all $a \in \Sigma_{\mathcal{K}} \cap \Sigma_{\mathcal{I}\setminus\mathcal{K}} \cap \Sigma_{\mathrm{out}}$ and $j \in \mathcal{I}$ such that $a \in \Sigma_{j,\mathrm{out}}$.

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 actually is nothing but the natural assumption that an action included as an output signal will indeed be used as such.

4.2. Superteams

In [2] it is shown how team automata can be used to construct higher-level automata 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 Corollary 9 we know that also team I/O automata can be used as components in a hierarchical construction. In the remainder of this section 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 contrasts with the usual approach [7,11,13] that focuses on behavior and is mostly restricted to binary compositions.



$\mathcal{L}'' = \{1,2,3,4,5,6\}$, $\mathcal{L}''_1 = \{1,2,3,5\}$, $\mathcal{L}''_2 = \{4,6\}$,
$\langle((q_2,q_3,q_5),(q_1)),(q_4,q_6))\rangle_{Q_{\mathcal{L}''}} = (q_1,q_2,q_3,q_4,q_5,q_6)$,
$\mathcal{L}' = \{1,2,3,5\}$, $\mathcal{L}'_1 = \{2,3,5\}$, $\mathcal{L}'_2 = \{1\}$,
$\langle((q_2,q_3,q_5),(q_1))\rangle_{Q_{\mathcal{L}'}} = (q_1,q_2,q_3,q_5)$,
$\mathcal{L} = \{2,3,5\}$,
$\langle(q_2,q_3,q_5)\rangle_{Q_{\mathcal{L}}} = (q_2,q_3,q_5)$

Fig. 1. Sketch of the process of reordering.

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 an indexed 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. Formally, $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ is composable whenever $\mathcal{S}$ is composable and each $\mathcal{T}_j$ is a team automaton over $\{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$, where $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ is a partition of $\mathcal{I}$. Consequently, we recall the following recursive definition for iterated team automata over a given composable set.

Definition 17. 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 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 states are "essentially" the states defined by the composable set, where "essentially" means up to 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. Hence rearranging the state space is sufficient to consider an iterated team automaton as an ordinary team automaton. The process of reordering outlined below is a simpler and more concrete description of the abstract reordering from [2]. It is sketched in Fig. 1.

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 its recursive structure 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 $\mathcal{T}_\mathcal{L}$ be 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 $\mathcal{L} \subseteq \mathcal{I}$ 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 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 18. *Let $\mathcal{S}$ be composable and $\mathcal{T}$ be an iterated team automaton over $\mathcal{S}$. Then its reordered version $\langle\!\langle \mathcal{T}\rangle\!\rangle_{\mathcal{S}}$ is a team automaton over $\mathcal{S}$.*

Conversely, every team automaton can be seen—after reordering—as composed of any disjoint combination of its subteams, i.e., any 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—up to 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 19. *Let $\mathcal{S}$ be composable and $\mathcal{T}$ be a team automaton over $\mathcal{S}$. Let $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ be a partition of $\mathcal{I}$ and, for all $j \in \mathcal{J}$, $\mathcal{T}_j$ be 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 19 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.

Theorem 20. *Let $\mathcal{S}$ be composable and $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ be 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 12 and Theorem 19, 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 $\mathcal{C}_i$ 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 $(p, a, p') \in \chi^{\{\mathcal{T}_j \mid j\in\mathcal{J}\}}$. 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. □

Note that, as remarked before, input-enabledness is preserved by synchronous products and hence by iterated synchronous product constructions (cf. Corollary 9). Moreover, like composability obviously also compatibility is preserved when iteratively forming teams. Formally, $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ is compatible whenever $\mathcal{S}$ is compatible and each $\mathcal{T}_j$ is a team automaton over $\{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$, where $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ is a partition of $\mathcal{I}$. Thus we can propose a definition of iterated team I/O automata similar to that of iterated team automata (cf. Definition 17).

Definition 21. Let $\mathcal{S}$ be a compatible set of I/O automata. $\mathcal{T}$ 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 Theorem 20 it follows that iteratively applying a synchronous product to construct higher-level team automata yields—up to 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 22. *Let $\mathcal{S}$ be a compatible set of I/O automata and $\mathcal{T}$ be 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 Theorem 20 implies that $\langle\langle\mathcal{T}\rangle\rangle_{\mathcal{S}} = \mathcal{X}(\mathcal{S})$, as desired. □

5. Discussion

We have embedded I/O automata in the framework of team automata. 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. As demonstrated in this paper, these notions can be studied independently of one another in the framework of team automata. Among other things, this leads to general results—like Theorems 8 and 20—which can be applied to I/O automata, to the formal distinction between team I/O automata and synchronous products of I/O automata, and to precise notions of sub- and superteams of I/O automata.

Emphasis has been given to the modular structure of team I/O automata. We have not dealt with behavior as expressed through computations and we have not considered *fairness*. Also the notion of *strong* compatibility—by which no action may belong to infinitely many components—is not significant to our exposition. In [1] we did study the behavior of team automata and its implications for I/O automata, which we intend to be the subject of a forthcoming paper.

Finally, we would like to point out that whereas an I/O automaton—in fact, every synchronous product automaton—can immediately be seen as a Petri net, this is not the case for team automata in general [4].

Acknowledgements

We thank the three anonymous referees for suggestions to improve our presentation.

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, G. Rozenberg, Synchronizations in team automata for groupware systems, CSCW 12 (1) (2003) 21–69.
[3] M.H. ter Beek, J. Kleijn, Team automata satisfying compositionality, in: Proc. FM 2003, in: Lecture Notes in Comput. Sci., vol. 2805, Springer, Berlin, 2003, pp. 381–400.
[4] J. Carmona, J. Kleijn, Interactive behaviour of multi-component systems, in: Proc. ToBaCo'04, University of Bologna, 2004, pp. 27–31.
[5] C.A. Ellis, Team automata for groupware systems, in: Proc. GROUP'97, ACM Press, New York, 1997, pp. 415–424.
[6] R. Gawlick, R. Segala, F. Søgaard-Andersen, N.A. Lynch, Liveness in timed and untimed systems, in: Proc. ICALP'94, in: Lecture Notes in Comput. Sci., vol. 820, Springer, Berlin, 1994, pp. 166–177.
[7] N.A. Lynch, Distributed Algorithms, Morgan Kaufmann, San Francisco, CA, 1996.
[8] N.A. Lynch, Input/output automata: basic, timed, hybrid, probabilistic, dynamic, in: Proc. CONCUR'03, in: Lecture Notes in Comput. Sci., vol. 2761, Springer, Berlin, 2003, pp. 191–192.
[9] N.A. Lynch, M. Merritt, Introduction to the theory of nested transactions, Theoret. Comput. Sci. 62 (1–2) (1988) 123–185, Presented at ICDT'86.
[10] N.A. Lynch, M.R. Tuttle, Hierarchical correctness proofs for distributed algorithms, in: Proc. PODC'87, ACM Press, New York, 1987, pp. 137–151.
[11] N.A. Lynch, M.R. Tuttle, An introduction to input/output automata, CWI Quarterly 2 (3) (1989) 219–246.
[12] N.A. Lynch, F. Vaandrager, Forward and backward simulations for timing-based systems, in: Real-Time: Theory in Practice, in: Lecture Notes in Comput. Sci., vol. 600, Springer, Berlin, 2003, pp. 397–446.
[13] M.R. Tuttle, Hierarchical correctness proofs for distributed algorithms, Master's Thesis, Dept. of Electrical Engineering and Computer Science, MIT, 1987.