Theoretical Computer Science 484 (2013) 1–15

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

# Compatibility in a multi-component environment

Josep Carmona [a,*], Jetty Kleijn [b]

[a] Software Department, Universitat Politècnica de Catalunya, c/Jordi Girona, 1–3, 08034, Barcelona, Spain
[b] LIACS, Leiden University, P.O. Box 9512, NL-2300 RA, Leiden, The Netherlands

ARTICLE INFO

*Article history:*
Received 28 October 2011
Received in revised form 22 October 2012
Accepted 4 March 2013
Communicated by P.A. Abdulla

ABSTRACT

A distributed environment where many components interact may be functioning in a suboptimal manner due to two main factors: message loss and deadlocks. Message loss occurs when a component is not ready to receive as input a message sent to it. In the case of a deadlock, a system is indefinitely waiting for a message that never arrives. In Carmona and Cortadella (2002) [12] a theory has been presented for characterizing when a pair of systems is compatible in the sense that they can engage in a dialog free from these two problems. The theory developed was restricted to only two components, a particular mode of synchronization and a closed environment. In this paper we lift all these assumptions to define a general notion of compatibility in a multi-component environment. For the extended definition of compatibility, we use team automata as a modeling formalism which allows arbitrary synchronization strategies and iterative/hierarchical composition. Moreover, it is shown how the general definition of compatibility presented in this paper can be used to determine the compatibility problems that arise in a team automaton built on the basis of an arbitrary synchronization strategy.



## 1. Introduction

### 1.1. Reactive systems and compatibility

A reactive system interacts with its environment by means of output and input signals (active and passive actions). In [11,12] the notion of Input/Output (I/O) compatibility is used to model the fact that two systems (in particular a system and its environment) can interact in such a way that a correct dialog is established between their input and output actions. In a distributed system the environment of a component consists of other components of the system, and correct interaction is not necessarily a binary relation.

Compatibility can be seen as a first step in establishing the correctness of a distributed system. A compatibility failure, detected in e.g., an automaton model of the system, may reveal important problems in the design of a component that can then be repaired before implementing it.

In this paper we establish conditions guaranteeing the feasibility of constructing a properly functioning system from a set of given modules (subsystems). Therefore we consider distributed systems consisting of several reactive components which cooperate through synchronizations on common actions. In summary, the two goals of this paper are:

- *To define a general notion of compatibility in a multi-component environment, by extending the binary I/O compatibility of* [11,12]. There are several issues that arise in the generalization and were not considered before due to the more restricted setting. First, since the input or output domain of an action is in general not a single component, the execution

* Corresponding author. Tel.: +34 934137853.
*E-mail addresses:* jcarmona@lsi.upc.edu (J. Carmona), kleijn@liacs.nl (J. Kleijn).

0304-3975/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.tcs.2013.03.006

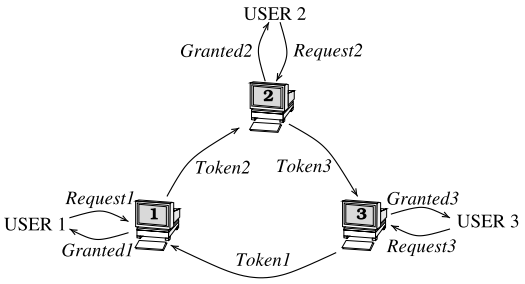



**Fig. 1.** Ring of stations sharing a bus.

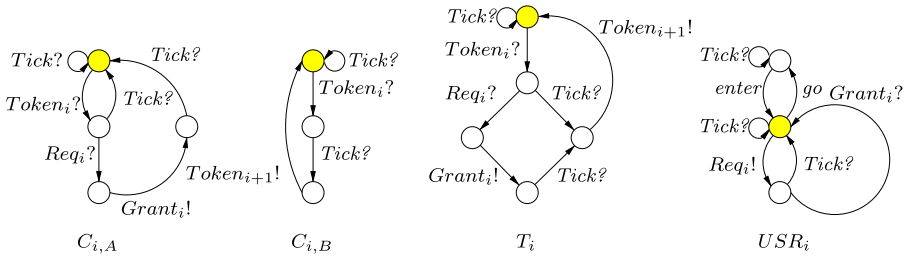



**Fig. 2.** The Station components ($\mathcal{C}_{i,A}$, $\mathcal{C}_{i,B}$), the Station ($\mathcal{T}_i$) and the User ($\mathcal{USR}_i$). Initial states of each automaton are shaded.

of an action may involve very different components depending on the state considered. Second, a system does not need to be complete, i.e., the components might have external actions that may become communicating actions only after more components have been incorporated into the system. Finally, the role of actions is a dynamic feature if iterative construction is considered, e.g., an action which is input to some components of a system is considered satisfied within the system (in the sense that it is not an input action of the system as a whole) when it appears as an output action of another component (see Definition 2).

- *To accommodate different synchronization strategies*. The framework of team automata, that we use in this paper to capture the general concept of compatibility in the case of synchronous product (a standard synchronization strategy, e.g., [5,24,25,2]), also allows us to deviate from this synchronization strategy. There are several examples both in hardware and in software where the communication is not mandatory synchronous. In hardware, asynchronous communication has been proved to be a meaningful mechanism and several paradigms for computation have appeared like *Globally Asynchronous Locally Synchronous* systems [16], or the more recent *Elastic Systems* [13]. In software, the *Erlang* [4] programming language may be seen as a prominent example of the asynchronous mode of communication between processes. Since compatibility appears to have been defined until now essentially for systems composed by synchronous product like *Interface Automata* [2] and *I/O Automata* [24,25], such a more general approach could fill a gap.

The next subsections give some intuition for the issues addressed in this paper. We will use as a running example a ring of stations sharing a bus. The example is inspired by the Esterel program presented in [10].

### 1.2. Example

**Example 1.** Consider a system where three identical stations are hooked forming a ring (see Fig. 1). Each station has its own user who can perform requests for accessing a common resource, e.g., a bus. Moreover, every request made by a user can be granted or not, depending on whether the corresponding station has the right to grant, which is implemented by means of tokens that flow along the ring. When station $i$ receives the token, it has the right to grant. To ensure fairness, a grant to the user only lasts one clock tick, after which the token is passed on. So a global clock is assumed (not shown explicitly) that only produces ticks and lets all the components synchronize.

The behavior of a station is defined by the two components on the left in Fig. 2. Exclamation-marks and question-marks are used to clarify whether an action has an output role (!) or an input role (?) for each component in which it occurs. Station $\mathcal{T}_i$ has two component automata $\mathcal{C}_{i,A}$, and $\mathcal{C}_{i,B}$. Informally, the main behavior of the station is expressed by component $\mathcal{C}_{i,A}$:

*$\mathcal{C}_{i,A}$: if a token is present then check whether the user has made a request within the current tick. If so, grant the bus to the user. Otherwise, go back to the initial state when receiving the next tick.*

Component $\mathcal{C}_{i,B}$ provides fairness to the whole system, by forcing the continuous movement of the tokens:

*$\mathcal{C}_{i,B}$: if a token is present, wait for the next tick and then pass the token.*

The complete behavior of the $i$-th station is depicted in the third automaton of Fig. 2, named $\mathcal{T}_i$. The automaton $\mathcal{T}_i$ is obtained as the synchronous product of $\mathcal{C}_{i,A}$ and $\mathcal{C}_{i,B}$ (as formally defined in Section 2), meaning that the components have to synchronize on shared actions. Finally, the model for a simple user is also presented in Fig. 2 (right-most). The informal behavior is:

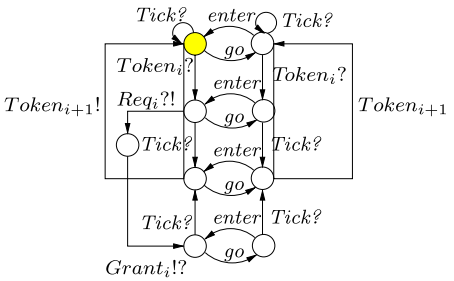



**Fig. 3.** Synchronous product of the Station $\mathcal{T}_i$ and the User $\mathcal{USR}_i$. We write $x?!$ when action $x$ in role $x?$ and $x!$ is executed synchronously. Notice that the station and the user collaborate in the execution of *Tick*?.

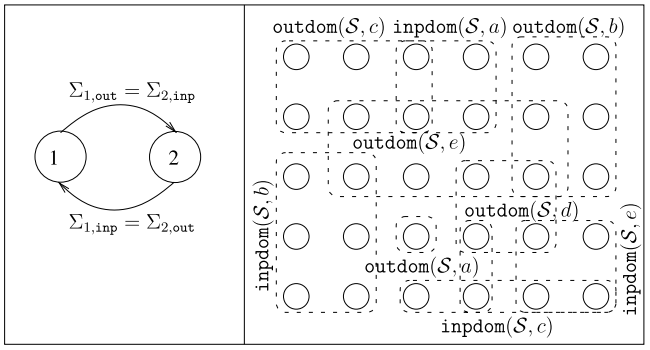



**Fig. 4.** Lifting the compatibility to the $n$-dimensional case: from components to domains. The terms $\texttt{outdom}(S, x)$ ($\texttt{inpdom}(S, x)$) stand for the output (input) domain of action $x$ in the system $\mathcal{S}$.

*$\mathcal{USR}_i$: a user can at any time perform a request or leave temporarily the system. Once a request has been formulated, if a tick is received before the request is granted, go back to the initial state.*

The analysis of a distributed system or algorithm is typically done at the level of message-interchange or distributed execution of actions. How actions by different components are executed within the system is a matter of design. A very common strategy is to use the *synchronous product* (see e.g., [5,24,25,2]), according to which a joint action can be executed only if all components sharing the action are ready for it, and execute the action simultaneously.

Returning to the example, the joint behavior of the station and the user according to the synchronous product is the automaton depicted in Fig. 3. The joint signals $Req_i$ and $Grant_i$ as well as *Tick* appear in the automaton if the user and the station execute them synchronously. Other behavior is not allowed in the synchronous product: for instance, attempts made from the user which do not reach the station (e.g., a network failure, an incorrect address) are not modeled in this automaton.

### 1.3. Compatibility in a multi-component environment

In a reactive system a component's actions that are not internal are either input or output actions for that component. Input actions are generally [24,25,19] considered as being passive in the sense that in order to occur they should be triggered by output events. For instance in the intended interaction between the user and the station in Example 1, it is the user who activates the communication, and the station should follow. For such communicating behavior, the notion of I/O compatibility was presented in [12] to characterize *pairs* of reactive systems that can communicate successfully. Compatible behavior means especially that outputs will be received and there are no deadlocks leaving components waiting for input. Compatibility is introduced as a bisimulation relation between the participating components. The easy extension suggested in that paper for the $n$-dimensional case allows actions to occur only in one component as input and one component as output, i.e. no two or more components share an action as input or output. This paper demonstrates that if this assumption is dropped, i.e. components may *collaborate* when executing an action (i.e., the action has the same role in these components), the extension to the $n$-dimensional case is more challenging. In Fig. 4 we illustrate the difficulties that arise when moving to the $n$-dimensional case: in general, in an $n$-component system, it is not the components but *domains* that send/receive actions. Moreover, since an $n$-dimensional distributed system evolves by the interaction of its components, different actions become enabled, and thus the domains that can/should communicate depend on the current state of the system and the actions it enables locally.

Our approach is to lift I/O compatibility to the general *team automata* model (see Section 1.4), which allows us to express different synchronization strategies between any number of components. Motivated by the construction that will be presented in Section 4 which provides a faithful representation of arbitrary synchronization strategies based on the synchronous product, we will generalize the notion of compatibility in a multi-component environment.

In summary, this paper presents a generalized notion of I/O compatibility (in the rest of the paper referred to as compatibility) in the context of the team automata framework.

### 1.4. Team automata

As a formal and general model for multi-component systems we use team automata (TA) which were originally introduced to model components of groupware systems and their interconnections [19,9]. TA are inspired by the I/O automata of [24]. Both distinguish between input, output, and internal actions. TA however impose less restrictions on the role of the actions in the various components and do not fix an *a priori* mode of synchronization, like the *synchronous product*. TA are models for distributed systems, which do not describe physical or software systems but instead specify intended behavior. This implies that there is not a single synchronization strategy for components.

### 1.5. Organization of the paper

Section 2 provides the necessary background on TA, the framework used in this paper to model reactive systems. Section 3 presents the first contribution of the paper, a general notion of compatibility of composable systems. Safety and liveness properties are proven for compatible systems. Section 4 contains the conditions under which a restricted synchronization mechanism can unambiguously represent an arbitrary synchronization scheme, in order to facilitate the analysis of arbitrarily composed team automata. The results obtained in this section allow us to reason about the compatibility of arbitrary TA. Section 5 discusses related work and reports conclusions and future work. We provide the proofs for some of the results of the paper in an Appendix.

## 2. Modeling with team automata

In this section we introduce the TA model [19,8,9]. Systems are modeled as (possibly infinite) transition systems. They use a finite set of abstract action names, which may be empty.

**Definition 1.** A *transition system* (or *TS*) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, I)$, where $Q$ is a set of *states*, possibly infinite; $\Sigma$ is an alphabet of *actions*, such that $Q \cap \Sigma = \emptyset$; $\delta \subseteq Q \times \Sigma \times Q$ is a set of *(labeled) transitions*; and $I \subseteq Q$ is a set of *initial states*. $\mathcal{A}$ is *reactive* (an *RTS*) if its set of actions $\Sigma$ is the union of three pairwise disjoint subsets $\Sigma_{\mathtt{inp}}$, $\Sigma_{\mathtt{out}}$, and $\Sigma_{\mathtt{int}}$ of *input*, *output* and *internal* actions, respectively. In addition, the elements of $\Sigma_{\mathtt{ext}} = \Sigma_{\mathtt{inp}} \cup \Sigma_{\mathtt{out}}$ are referred to as the *external* actions of $\mathcal{A}$.

If $\mathcal{A}$ is an RTS as above, then it can also be represented, with the explicit partition of its actions, as $(Q, (\Sigma_{\mathtt{inp}}, \Sigma_{\mathtt{out}}, \Sigma_{\mathtt{int}}), \delta, I)$. Graphically, output and input actions are suffixed with the symbol ! and ? respectively, while internal actions are denoted by Greek letters. For a TS $\mathcal{A} = (Q, \Sigma, \delta, I)$ and an action $a \in \Sigma$, the set of $a$-transitions in $\mathcal{A}$ is $\delta_a = \delta \cap (Q \times \{a\} \times Q)$. Action $a$ is said to be *enabled* in $\mathcal{A}$ at state $q \in Q$, denoted by $a\ \mathtt{en}_{\mathcal{A}}\ q$, if there exists a $q'$ such that $(q, a, q') \in \delta$. We say that $q$ is *input-enabled* if $\Sigma_{\mathtt{inp}} \subseteq \{a | a\ \mathtt{en}_{\mathcal{A}}\ q\}$.

A *computation of* $\mathcal{A}$ is a (possibly infinite) sequence $\sigma = q_0 a_1 q_1 a_2 \cdots$ where $q_0 \in I$ and $q_i \in Q$ for all $1 \leq i$, and $(q_i, a_{i+1}, q_{i+1}) \in \delta$, for all $0 \leq i$. All $q_i$ in such a sequence are said to be *reachable* states of $\mathcal{A}$. The set of computations of $\mathcal{A}$ is denoted by $\mathcal{C}_{\mathcal{A}}$. A state $q \in Q$ is *terminating* if no actions at all are enabled at $q$ and $\mathcal{A}$ is terminating if it has at least one reachable terminating state.

A TA is constructed from a finite number of components which are RTSs. These may cooperate by synchronization on common (external) actions. Internal actions are not available for synchronization, i.e. they are private actions in the sense that they belong uniquely to a component. Thus a collection $\mathcal{S} = \{\mathcal{A}_i \mid 1 \leq i \leq n\}$ of *component automata* (RTSs) $\mathcal{A}_i = (Q_i, \Sigma_i, \delta_i, I_i)$, with $\Sigma_i$ partitioned into internal, input and output actions $(\Sigma_{i,\mathtt{int}}, \Sigma_{i,\mathtt{inp}}, \Sigma_{i,\mathtt{out}})$, respectively, is said to form a *composable system* if $\Sigma_{i,\mathtt{int}} \cap \bigcup_{j=1, j\neq i}^{n} \Sigma_j = \emptyset$ for all $1 \leq j \leq n$. Given a composable system, a TA is constructed by *selecting* synchronizations on common actions of its components, while retaining the local effect of each action. The state space of such TA is $\Pi_{i=1}^{n} Q_i$. The function $\mathtt{proj}_i$ maps global states into local ones, i.e., for $p = (p_1, \ldots, p_n) \in \Pi_{i=1}^{n} Q_i$, we have $\mathtt{proj}_i(p) = p_i$ for $1 \leq i \leq n$.

**Definition 2.** Let $\mathcal{S}$ be a composable system as specified above. Let $a \in \bigcup_{i=1}^{n} \Sigma_i$. The *complete transition space of $a$ in $\mathcal{S}$* is $\Delta_a(\mathcal{S}) = \{(p, a, p') \mid p, p' \in \Pi_{i=1}^{n} Q_i,\ (\mathtt{proj}_i(p), a, \mathtt{proj}_i(p')) \in \delta_i$ for some $1 \leq i \leq n$, and for all $1 \leq i \leq n$ either $(\mathtt{proj}_i(p), a, \mathtt{proj}_i(p')) \in \delta_i$ or $\mathtt{proj}_i(p) = \mathtt{proj}_i(p')\}$. A *TA over $\mathcal{S}$* is an RTS $\mathcal{T} = (\Pi_{i=1}^{n} Q_i, \Sigma, \delta, \Pi_{i=1}^{n} I_i)$ with $\delta_a \subseteq \Delta_a(\mathcal{S})$ for all $a \in \Sigma_{\mathtt{ext}}$, and $\delta_a = \Delta_a(\mathcal{S})$ for all $a \in \Sigma_{\mathtt{int}}$. Its set of actions is $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$ with $\Sigma_{\mathtt{int}} = \bigcup_{i=1}^{n} \Sigma_{i,\mathtt{int}}$, $\Sigma_{\mathtt{out}} = \bigcup_{i=1}^{n} \Sigma_{i,\mathtt{out}}$, and $\Sigma_{\mathtt{inp}} = \bigcup_{i=1}^{n} \Sigma_{i,\mathtt{inp}} - \bigcup_{i=1}^{n} \Sigma_{i,\mathtt{out}}$.

Thus, the internal actions of a TA over $\mathcal{S}$ are the internal actions of its components; its output actions are the output actions of its components, while its input actions are the remaining external actions (those that do not appear in any component as output). Consequently, each TA is again an RTS which can be used in its turn as a component in the construction of a TA. Note that from a given composable system, many TA may be constructed differing only w.r.t. their transitions. Internal actions can always be executed in any team whenever they can be executed by their component. However not all possible synchronizations on an external action as given in its complete transition space have to be present in a TA.

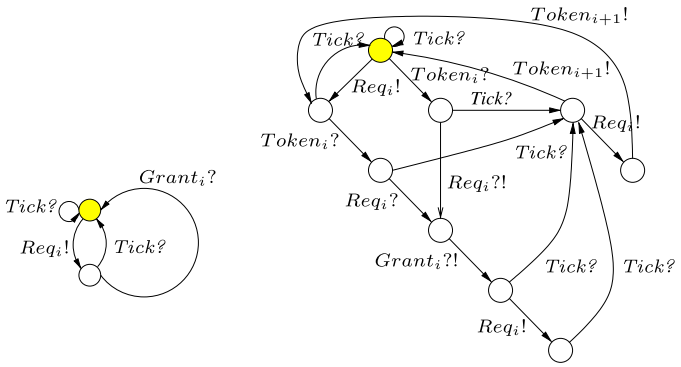



**Fig. 5.** Joint behavior (shown on the right) of the simplified user (left) and the station $\mathcal{T}_i$ of Fig. 2. Requests can be: (a) lost: in the initial state, the sequence $Req_i!$ *Tick*? leads back to the initial state with a non-granted request, meaning that the user made a request, but the clock tick occurred before the station reacted; (b) pending: the sequence $Req_i!$ $Token_i?$ is followed by the request (input) action of the station; (c) synchronized: when a token has arrived in the initial state, the request action is executed synchronously by the user and the station.

Let $\mathcal{S} = \{\mathcal{A}_i \mid 1 \leq i \leq n\}$ be a composable system as specified before. $\mathcal{S}$ represents a collection of interacting components. Let $a \in \bigcup_{i=1}^{n} \Sigma_i$ be an action of $\mathcal{S}$. Then $\mathtt{dom}(\mathcal{S}, a) = \{i \mid a \in \Sigma_i\}$ is the *domain* of $a$ in $\mathcal{S}$. The *output domain* of $a$ in $\mathcal{S}$ is $\mathtt{outdom}(\mathcal{S}, a) = \{i \mid a \in \Sigma_{i,\mathtt{out}}\}$. Similarly, $\mathtt{inpdom}(\mathcal{S}, a) = \{i \mid a \in \Sigma_{i,\mathtt{inp}}\}$ is the *input domain* of $a$ in $\mathcal{S}$.

We say that an external action $a$ is *output-domain* (*input-domain*) *enabled at* $p$ if $a\ \mathtt{en}_{\mathcal{A}_i}\ \mathtt{proj}_i(p)$ for all $i \in \mathtt{outdom}(\mathcal{S}, a)$ (for all $i \in \mathtt{inpdom}(\mathcal{S}, a)$). It is *domain* enabled if it is both input-domain and output-domain enabled.

An action is said to be *communicating* in $\mathcal{S}$ if it has a non-empty output domain and a non-empty input domain. We write $\Sigma_{\mathtt{com}} = \bigcup_{i=1}^{n} \Sigma_{i,\mathtt{out}} \cap \bigcup_{i=1}^{n} \Sigma_{i,\mathtt{inp}}$. External actions which are not communicating in $\mathcal{S}$ may become so in an extension of $\mathcal{S}$ when components have been added. We say that $\mathcal{S}$ is *complete* if each of its external actions is communicating. Note that, in contrast to I/O automata, it may be the case that an action is output in two or more component automata, just like it is possible that an action appears in several components as input action.

A component $\mathcal{A}_i$ has a *livelock* if there is an infinite sequence $q_1, q_2, q_3, \ldots$ of states in $Q_i$ and an infinite sequence $a_1, a_2, a_3, \ldots$ of internal actions in $\Sigma_i$ such that $(q_j, a_j, q_{j+1}) \in \delta_i$ for all $j \geq 1$. A livelock-free RTS will always ultimately execute an external action, interact with fellow components, or terminate.

The framework of team automata does not presuppose a particular principle underlying the synchronization of component automata. A team automaton over a composable system is defined by a selection of (partial) synchronizations on external actions from the complete transition space of the system. A natural and frequently arising method for synchronizing components' actions is the so-called synchronous product. Then for each action, all and only those synchronizations on that action are included in the transitions of the team in which *all* components that have this action in their alphabet participate. Let the set of transitions $\chi^{\mathcal{S}}$ be defined by $\chi^{\mathcal{S}}_a = \{(p, a, p') \in \Delta_a(\mathcal{S}) \mid (\mathtt{proj}_i(p), a, \mathtt{proj}_i(p')) \in \delta_i \text{ for all } 1 \leq i \leq n \text{ such that } a \in \Sigma_i\}$ for all $a \in \Sigma$.

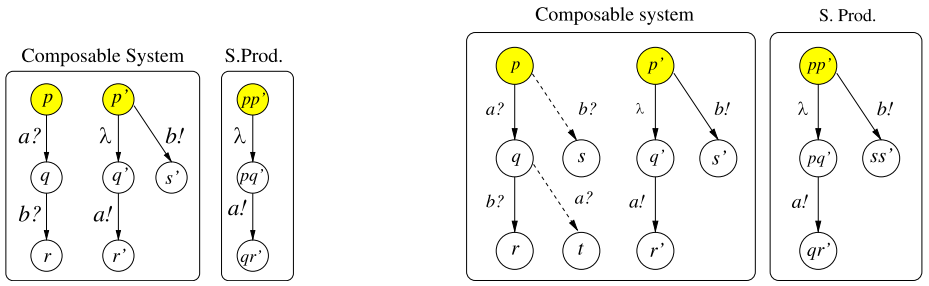
**Definition 3.** The *synchronous product over* a composable system $\mathcal{S}$, denoted by $\mathcal{X}(\mathcal{S})$, is the TA over $\mathcal{S}$ with $\chi^{\mathcal{S}}$ as its set of transitions.

Let us go back to Example 1 where we used the synchronous product construction to model the combined behavior of station and user. A perhaps more realistic model would be one where requests can be lost, i.e. sometimes the station cannot grant the bus to the user, because the token resides in another station. This requires a more flexible way of synchronizing on action $Req_i$. Fig. 5 depicts such a model, where, for the sake of clarity, a simplified model of the user is considered (having no actions *go* and *enter*, the user is now not allowed to leave). This TA models situations where a request $Req_i!$ by the user is not picked up by the station (by a synchronization with $Req_i?$) and thus may be lost or pending. This illustrates the flexibility of team automata as a framework where for each global state of a system and for all locally enabled actions of its components any combination of synchronized executions can be modeled. The choices made could depend on a chosen general principle (broadcasting, handshaking, peer-to-peer or master–slave communication) or be concrete, ad hoc implementations based on certain assumptions or requirements by the designer of a system regarding the environment or the interaction of components (see [8,19]).

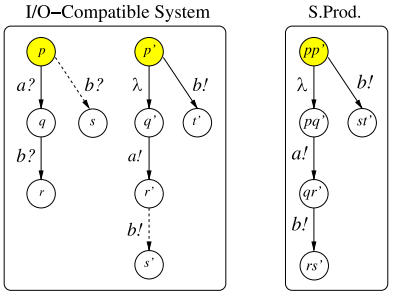
## 3. Compatibility

In this section we lift the notion of I/O compatibility from [11,12] to multi-component systems composed using the synchronous product. Even with this restriction, important differences arise when considering successful communicating and collaborating behavior. The following simple example illustrates this and is also used to motivate compatibility.

**Example 2.** In Fig. 6(a) a composable system (consisting of two RTSs) and the corresponding synchronous product are given. This system is complete (all external actions are communicating).




(a) Composable system and its synchronous product. (b) Synchronous product with two input-enabled states.




(c) I/O-compatible RTSs and their synchronous product.

**Fig. 6.** Comparing compatibility with related notions.

Note that action $b$ in the right component cannot be executed in the synchronous product even though it is an output action for that component. In case of so-called input-enabled states (at which all input actions are enabled, as required by I/O automata for all states [24]), this problem does not arise. In Fig. 6(b), in the left component the two states $p$ and $q$ are input-enabled while the other three states are terminating. Looking at the synchronous product derived from the two components, it can be seen that executing $b$ now is possible in the initial state. However, like in Fig. 6(a), in the state $(q, r')$ the right component has terminated but the left component is still waiting for input. To repair this, one should require components not only to enable inputs but also let the system progress to eventually provide required outputs. This is illustrated in Fig. 6(c), where the right component has an output action $b$ at state $r'$ and the required output can be provided.

Compatibility will take both aspects of the interaction into account: input from other components (produced as output) is accepted and the system will progress in the case of components waiting for input.

### 3.1. Compatibility in a multi-component environment

Informally, the original notion of I/O compatibility from [11,12] is a relation between the states of the components of a complete composable system fulfilling the following conditions (see the Appendix for a formal definition). First, any combination of initial states of the components (i.e. any initial state of the system) should belong to the relation. Second, whenever a system state belongs to the relation, then:

- *Non-communicating Progress*: internal steps do not lead outside the relation.
- *Receptiveness*: every output action sent is received as input. In other words, communication never fails for related states.
- *Deadlock-freeness*: whenever a component is waiting for input, the system cannot terminate.

The following definition generalizes these ideas to a multi-component environment, defined as the synchronous product over a composable system $\mathcal{S}$.

**Definition 4.** Let $\mathcal{S}$ be a composable system as before. Then $\mathcal{R} \subseteq \Pi_{i=1}^{n} Q_i$ is a *compatibility relation for* $\mathcal{S}$ if $\Pi_{i=1}^{n} I_i \subseteq \mathcal{R}$ and for all $p \in \mathcal{R}$ the following conditions are satisfied.

1. *Non-communicating Progress*: For all $a \in \bigcup_{i=1}^{n} \Sigma_i - \Sigma_{\mathtt{com}}$, if $a\, \mathtt{en}_{\mathcal{A}_i}\, \mathtt{proj}_i(p)$ for all $i \in \mathtt{dom}(\mathcal{S}, a)$, then $p' \in \mathcal{R}$, whenever $(p, a, p') \in \chi_a^{\mathcal{S}}$.
2. *Receptiveness*: For all $a \in \Sigma_{\mathtt{com}}$, if $a$ is output-domain enabled at $p$, then $a$ is input-domain enabled at $p$, and $p' \in \mathcal{R}$ whenever $(p, a, p') \in \chi_a^{\mathcal{S}}$.
3. *Deadlock-freeness*: If some action $a \in \Sigma_{\mathtt{com}}$ is input-domain enabled at $p$, then there are $b \in \bigcup_{i=1}^{n} \Sigma_i$ and $p' \in \Pi_{i=1}^{n} Q_i$ such that $(p, b, p') \in \chi^{\mathcal{S}}$.

$\mathcal{S}$ is said to be *compatible* if each of the component automata $\mathcal{A}_i$ is livelock-free and there exists a compatibility relation for $\mathcal{S}$. If $\mathcal{R}$ fulfills conditions 1, 2, or 3 then it is called *progressive*, *receptive* or *deadlock-free*, respectively.

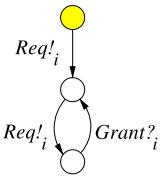



**Fig. 7.** Incorrect model of a user, where two requests are initially needed.

Compatibility reflects a certain concept of successful reactive behavior. In particular, receptiveness is a key notion which resembles the assumption of input-enabledness in I/O automata. Deadlock-freeness in combination with the absence of livelocks guarantees that the system, i.e. its synchronous product, proceeds as long as there are pending input requests (see Theorem 4).

It is immediate that the three conditions of Definition 4 follow the intuition behind the conditions for the binary I/O compatibility. For instance, condition 2 (receptiveness) states that communication never fails for related states: whenever (collaborating) output is possible, the corresponding input domain is ready for it. Moreover, all states that can be reached by such communication are again in the relation (this is expressed using the synchronous product transition relation). Condition 3 (deadlock-freeness) ensures progress when the input domain of a communicating action is ready to receive it. Note that condition 2 (condition 3) does not require progress for actions that are not output-domain (input-domain) enabled (one could say that no subsystem is waiting for communication).

Again, going back to Example 1, the station and the user in Fig. 2 are compatible. However, $\mathcal{T}_i$ and the user depicted in Fig. 7 are not compatible. Using compatibility one can detect incorrect situations in the interaction. Here, in the case of a user that has to send two requests before it can receive a grant, there is a receptiveness violation.

Definition 4 is intended as a generalization of the notion of an I/O-compatible system as defined in [12,11] for complete composable systems consisting of two livelock-free components. The generalization is done in several directions: first, the extension from two to any number of components. Second, generalizing components to input domains and output domains. Third, from complete to open (not necessarily complete) systems. Consequently, hierarchy and iterative construction can now be considered together with compatibility. Finally, termination is allowed when the domain of an action is not totally ready for an input. This deviates from I/O compatibility where domains take the role of components.

The generalized Definition 4 can be considered as a proper extension of the original I/O compatibility.

**Theorem 1.** *Let $\mathcal{S} = \{\mathcal{A}_1, \mathcal{A}_2\}$ be a complete composable system where each component automaton is non-terminating. Then $\mathcal{R} \subseteq Q_1 \times Q_2$ is a compatibility relation for $\mathcal{S}$ if and only if $\mathcal{R}$ is an I/O-compatibility relation for $\mathcal{S}$.*

**Proof.** See the Appendix. □

Before continuing, it is worthwhile to comment on the extra condition in Theorem 1. Both Non-Communicating Progress and the Receptiveness from Definition 4 translate immediately to the original conditions when restricted to complete systems with two components. Deadlock-freeness however requires a more careful approach. Receptiveness of $\mathcal{R}$ is necessary (and guaranteed if $\mathcal{R}$ is a compatibility relation) to demonstrate that the original concept implies the new version of deadlock-freeness when restricted to a complete system of two components. This is because in the original Definition of I/O compatibility (see Appendix) only enabledness is required rather than executability. Moreover, for the proof in the converse direction, it is necessary to require that the components are not terminating. The deadlock-freeness property of Definition 4 does not apply to terminated component states, whereas the original deadlock-freeness property implies that if one component has terminated, the other still has at least one enabled action. Together with receptiveness, this then leads to a livelock for that component. However, if $\mathcal{S}$ is compatible then each component is livelock-free. Consequently, the deadlock-freeness of $\mathcal{R}$ according to the original Definition 7 together with receptiveness implies that no component will ever terminate.

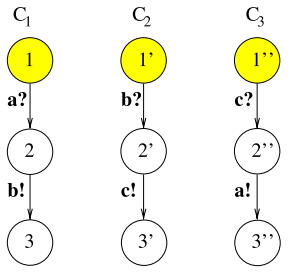
Now let $\mathcal{S}$ be a composable system consisting of livelock-free components. Then, we claim that — just as for I/O compatibility — if $\mathcal{S}$ is compatible with compatibility relation $\mathcal{R}$, then the reachable states of its synchronous product $\mathcal{X}(\mathcal{S})$ are contained in $\mathcal{R}$. Since, for any compatible, composable system, its reachable states form a compatibility relation, this implies that the reachable states of $\mathcal{X}(\mathcal{S})$ are the smallest compatibility relation for $\mathcal{S}$ or $\mathcal{S}$ is not compatible.

**Theorem 2.** *Let $\mathcal{S}$ be compatible with compatibility relation $\mathcal{R}$, and $p$ a reachable state in $\mathcal{X}(\mathcal{S})$. Then $p \in R$.*

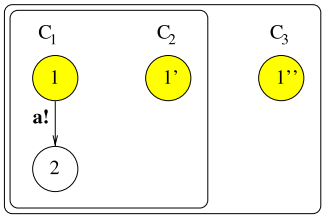
**Proof.** Follows directly from Definition 4. □

Consequently, to check compatibility in a synchronous product one can investigate the reachability relation. In [12] (Section 4, Theorem 4), a procedure is described to check the compatibility using the synchronous product. Since instead of components, domains can be used, a similar approach could be followed in the $n$-dimensional case. Notice that a compatibility relation may also include non-reachable states which could be reachable however in the case of different initial states. This resembles the situation for bisimulation relations.

The livelock-freeness and safety properties discussed in [12] can be extended in a straightforward way to compatible systems: communicating actions are never lost.




(a) Circular dependency. Compatibility for $\{\mathcal{C}_i, \mathcal{C}_j\}$ does not imply compatibility for $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$.




(b) Compatibility for $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$ does not imply compatibility for $\{\mathcal{C}_1, \mathcal{C}_2\}$.

**Fig. 8.** Compatibility in a multi-component environment.

**Theorem 3** (*Safety*). *Let $\mathcal{S}$ be compatible, $a \in \Sigma_{\mathtt{com}}$ and $p$ a reachable state of $\mathcal{X}(\mathcal{S})$ such that $\mathtt{outdom}(\mathcal{S}, a) \subseteq \{i \mid a\ \mathtt{en}_{\mathcal{A}_i}\ \mathtt{proj}_i(p)\}$. Then $\mathtt{inpdom}(\mathcal{S}, a) \subseteq \{i \mid a\ \mathtt{en}_{\mathcal{A}_i}\ \mathtt{proj}_i(p)\}$.*

**Proof.** Immediate from Theorem 2. □

The liveness property also holds for the $n$-dimensional case:

**Theorem 4** (*Liveness*). *Let $\mathcal{S}$ be compatible, and $p$ a reachable state of $\mathcal{X}(\mathcal{S})$ such that $\mathtt{inpdom}(\mathcal{S}, a) \subseteq \{i \mid a\ \mathtt{en}_{\mathcal{A}_i}\ \mathtt{proj}_i(p)\}$ for some $a \in \Sigma_{\mathtt{com}}$. Then $p$ is non-terminating and in particular, there exists a state $q$ of $\mathcal{X}(\mathcal{S})$ reachable from $p$ and $b \in \Sigma_{\mathtt{ext}}$ such that $b\ \mathtt{en}_{\mathcal{X}(\mathcal{S})}\ q$.*

**Proof.** This follows from the deadlock-freeness at $p$, and livelock-freeness in combination with internal progress and receptiveness. □

Theorem 4 is illustrated in Fig. 6(c): in the state $qr'$, the input domain of action $b$ is enabling it, and therefore some action must be allowed (e.g., $b$) in the synchronous product.

### 3.2. Hierarchical composition and compatibility

This section addresses the construction of a compatible system by iteration. This issue goes beyond the 2-dimensional case and hence was not considered before.

To allow for hierarchically or incrementally defined reactive systems, compatible systems are not required to be complete, i.e. there may be input or output actions that are not communicating in the system, but can be used in a later stage for communication with newly added components. As mentioned before, TA can be used to describe this process. To be precise, any TA can be iteratively constructed from its components (see, e.g., [8]).

The question is now how compatibility of a (complete) composable system relates to compatibility of its subsets (which again form composable systems). An example is shown in Fig. 8(a), where three RTSs are given. It is easy to see that any pair of these component automata forms a compatible system. There exists however no compatibility relation for the three components together. They deadlock due to circular waiting, a well-known phenomenon (for a similar situation see, e.g., [22]). Consequently, compatibility of a system cannot be deduced from the compatibility of its subsystems. The opposite does not hold either: compatibility of part of the system cannot be deduced from the compatibility of the whole system. Fig. 8(b) gives an example for this, where the only transition appears at state 1 with $a$ being an output action of $\mathcal{C}_1$, input in $\mathcal{C}_2$ and output in $\mathcal{C}_3$. It is easy to see that $\mathcal{R} = \{(1, 1', 1'')\}$ is a compatibility relation for $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$ whereas there is no compatibility relation for $\{\mathcal{C}_1, \mathcal{C}_2\}$, due to the receptiveness problem at $(1, 1')$. That problem does not appear at the state $(1, 1', 1'')$ because $a$ is not output-domain enabled.

The next results establish relations between the compatibility of iteratively constructed systems and their non-hierarchical counterparts. The idea is to construct the multi-component environment in a bottom-up manner. In this way, mistakes can be found early and their propagation avoided in the construction of a behaviorally correct multi-component system. First of all, we observe that indeed compatible systems can be constructed in a hierarchical way.

**Theorem 5.** *If $\mathcal{S} = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}$ is compatible then $\mathcal{S}' = \{\mathcal{X}(\{\mathcal{A}_1, \mathcal{A}_2\}), \mathcal{A}_3\}$ is compatible.*

**Proof.** See the Appendix. □

For instance, for the example of the ring of stations in Fig. 2, the system $\{\mathcal{C}_{i,A}, \mathcal{C}_{i,B}, \mathcal{USR}_i\}$ is compatible and therefore $\{\mathcal{T}_i, \mathcal{USR}_i\}$ is compatible as well.

Conversely, under specific conditions, one can conclude that the individual components in a hierarchically constructed system form a compatible set, as will be shown next. To prove the results, we are confronted with the fact that in a hierarchical construction the synchronous product of a subset of the components can hide communication/collaboration problems due to the fact that domains can be incomparable.

We say that two RTSs $\mathcal{A}_1, \mathcal{A}_2$ are *action-complementary* if they have no common output/input actions, e.g., $\Sigma_{1,inp} \cap \Sigma_{2,inp} = \emptyset$ and $\Sigma_{1,out} \cap \Sigma_{2,out} = \emptyset$. Furthermore, $\mathcal{A}_2$ *collaborates* with $\mathcal{A}_1$ if for every state of the Cartesian product, $(p_1, p_2) \in Q_1 \times Q_2$, and every $a \in (\Sigma_{1,inp} \cap \Sigma_{2,inp}) \cup (\Sigma_{1,out} \cap \Sigma_{2,out})$, $a$ is enabled at $p_2$ whenever $a$ is enabled at $p_1$.

**Lemma 6.** *If $\mathcal{S} = \{\mathcal{X}(\{\mathcal{A}_1, \mathcal{A}_2\}), \mathcal{A}_3\}$ is compatible and $\{\mathcal{A}_1, \mathcal{A}_2\}$ is receptive and deadlock-free,[1] then $\mathcal{S}' = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}$ is compatible provided that $\mathcal{A}_3$ collaborates with $\mathcal{X}(\{\mathcal{A}_1, \mathcal{A}_2\})$.*

**Proof.** See the Appendix. □

**Lemma 7.** *The following implications hold:*

- *If $\{\mathcal{A}_1, \mathcal{A}_2\}$ is complete, then $\mathcal{A}_1, \mathcal{A}_2$ are action-complementary.*
- *If $\mathcal{A}_1, \mathcal{A}_2$ are action-complementary, then $\mathcal{A}_1$ collaborates with $\mathcal{A}_2$.*

**Proof.** The two implications follow directly from the definitions of completeness, action-complementarity and collaboration. □

**Theorem 8.** *If $\mathcal{S} = \{\mathcal{X}(\{\mathcal{A}_1, \mathcal{A}_2\}), \mathcal{A}_3\}$ is compatible and $\{\mathcal{A}_1, \mathcal{A}_2\}$ is receptive and deadlock-free, then $\mathcal{S}' = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}$ is compatible if either:*

- *$\mathcal{S}$ is complete, or*
- *$\mathcal{X}(\{\mathcal{A}_1, \mathcal{A}_2\})$ and $\mathcal{A}_3$ are action-complementary, or*
- *$\mathcal{A}_3$ collaborates with $\mathcal{X}(\{\mathcal{A}_1, \mathcal{A}_2\})$.*

**Proof.** Follows directly by combining Lemmas 6, 7, and Theorem 2. □

**Corollary 9.** *If $\mathcal{S} = \{\mathcal{X}(\{\mathcal{A}_1, \mathcal{A}_2\}), \mathcal{A}_3\}$ is a compatible and complete system and $\{\mathcal{A}_1, \mathcal{A}_2\}$ is compatible, then $\mathcal{S}' = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}$ is compatible.*

## 4. Arbitrary synchronization strategies

The synchronous product restricts the behavior of components and so potentially interesting behavior may be lost if this synchronization strategy is assumed for common actions. However, to capture more general synchronization strategies, an extension of the compatibility notion is required. In this section we discuss the possibilities for this and how to extend compatibility properties to other synchronization strategies. Rather than trying to adapt Definition 4 to arbitrary synchronizations, we first transform general team automata into synchronous products, with the aim to check compatibility on this transformed representation and then transfer properties back to the initial team automaton. The reasons for this approach are:

1. This representation facilitates an algorithmic approach to check compatibility for arbitrary synchronizations based on the synchronous product. Thus we can still use existing algorithms to analyze and verify systems.
2. The synchronous product provides a uniform way of defining compatibility. In contrast, defining compatibility for arbitrary synchronizations will require ad hoc compatibility definitions for *all* types of synchronization and corresponding theorems as in Section 3.
3. The class of systems for which the transformation preserves behavior faithfully can be characterized (see Theorem 13). This allows us to identify the team automata for which the transfer of properties back to the original system is feasible, thus decoupling compatibility checking from the specific synchronization strategy used.

### 4.1. Uniform representation of arbitrary synchronizations

As the following construction demonstrates, any team automaton $\mathcal{T}$ over a composable system $\mathcal{S}$ can be converted into a synchronous product by using the synchronizations as action names.

Let $\mathcal{S} = \{\mathcal{A}_i \mid 1 \leq i \leq n\}$ be specified by $\mathcal{A}_i = (Q_i, \Sigma_i, \delta_i, I_i)$ for $1 \leq i \leq n$, and $\mathcal{T}$ be the team automaton over $\mathcal{S}$ with set of transitions $\delta$. Each of the component automata $\mathcal{A}_i$ is transformed into a new component automaton $\mathcal{A}'_i = (Q_i, \Sigma'_i, \delta'_i, I_i)$ on the basis of the transitions of $\mathcal{T}$: for $\mathtt{x} = \mathtt{inp}, \mathtt{out}, \mathtt{int}$ we define $\Sigma'_{i,\mathtt{x}} = \{[p, a, p'] \mid a \in \Sigma_{i,\mathtt{x}}, (p, a, p') \in \delta\}$; and the new transitions are $\delta'_i = \{(\mathtt{proj}_i(p), [p, a, p'], \mathtt{proj}_i(p')) \mid (p, a, p') \in \delta \text{ and } (\mathtt{proj}_i(p), a, \mathtt{proj}_i(p')) \in \delta_i\}$. Thus each action is replaced by copies encoding a synchronization in which it occurs in $\mathcal{T}$. Note that actions which are not used in $\mathcal{T}$ disappear in the new transition relation $\delta'_i$, but the alphabet of each component has a representative for each action that can be executed. Moreover, these actions keep their role (internal, input, output) in the new component as in the corresponding original component. This is a crucial fact that will be used at the end of this section for reasoning about the compatibility of components given an arbitrary synchronization strategy.

[1] The reachable states of $\mathcal{X}(\mathcal{A}_1, \mathcal{A}_2)$ form a receptive and deadlock-free relation.

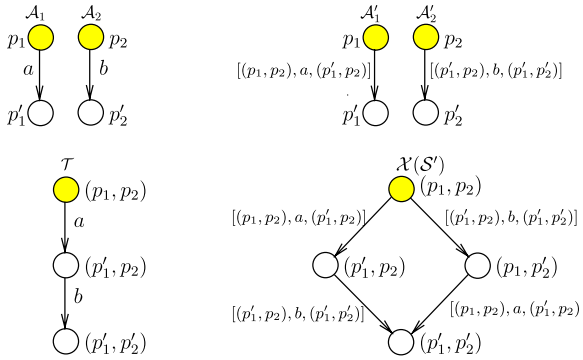



**Fig. 9.** State-sharing property.

It is immediate that $\mathcal{S}' = \{\mathcal{A}'_i \mid 1 \leq i \leq n\}$ is again a composable system. Now consider $\mathcal{X}(\mathcal{S}')$ the synchronous product over $\mathcal{S}'$ with set of transitions $\chi^{\mathcal{S}'}$. This team automaton has the same set of (initial) states as $\mathcal{T}$. Moreover any (finite) computation of $\mathcal{T}$ can be obtained from a computation of $\mathcal{X}(\mathcal{S}')$ using the decoding defined by the (non-injective) morphism $h : (\Pi_{i=1}^n Q_i \cup \bigcup_{i=1}^n \Sigma'_i)^* \to (\Pi_{i=1}^n Q_i \cup \bigcup_{i=1}^n \Sigma_i)^*$ with $h(p) = p$ for all $p \in \Pi_{i=1}^n Q_i$ and $h([p, a, p']) = a$ for all $[p, a, p'] \in \bigcup_{i=1}^n \Sigma'_i$. For infinite computations $h$ is extended in the obvious way.

**Lemma 10.** *Let* $(p, a, p') \in \delta$. *Then* $(p, [p, a, p'], p') \in \chi^{\mathcal{S}'}$.

**Proof.** Let $1 \leq i \leq n$ be such that $[p, a, p'] \in \Sigma'_i$. Since $(p, a, p') \in \delta$, there is at least one such $i$. Then $(\mathtt{proj}_i(p), [p, a, p'], \mathtt{proj}_i(p')) \in \delta'_i$ by the definition of $\delta'_i$. Moreover, whenever $[p, a, p'] \notin \Sigma'_j$ for some $1 \leq j \leq n$, then $\mathtt{proj}_j(p) = \mathtt{proj}_j(p')$. Hence $(p, [p, a, p'], p') \in \chi^{\mathcal{S}'}$. □

Consequently, we can prove now that indeed all elements of $C_{\mathcal{T}}$, the set of computations of $\mathcal{T}$, can be obtained via $h$ from a computation of $\chi(S')$.

**Theorem 11.** $\mathcal{C}_{\mathcal{T}} \subseteq h(\mathcal{C}_{\mathcal{X}(\mathcal{S}')})$.

**Proof.** Let $\sigma = q_0 a_1 q_1 a_2 \cdots$ be a computation of $\mathcal{T}$. Thus, $(q_{i-1}, a_i, q_i) \in \delta$ for all $i \geq 1$. From Lemma 10 it now follows that $\tau = q_0[q_0, a_1, q_1]q_1[q_1, a_2, q_2] \cdots$ is a computation of $\mathcal{X}(\mathcal{S}')$. Since $h(\tau) = \sigma$, the statement follows. □

In general however, not every computation of $\mathcal{X}(\mathcal{S}')$ corresponds via a decoding by $h$ with a computation of $\mathcal{T}$.

**Example 3.** Let $\mathcal{S} = \{\mathcal{A}_1, \mathcal{A}_2\}$ be the composable system consisting of two automata (shown in Fig. 9) with $\mathcal{A}_1 = (\{p_1, p'_1\}, \{a\}, \delta_1, \{p_1\})$ and similarly $\mathcal{A}_2 = (\{p_2, p'_2\}, \{b\}, \delta_2, \{p_2\})$ with $a$ and $b$ two distinct external actions, and $\delta_1 = \{(p_1, a, p'_1)\}$, $\delta_2 = \{(p_2, b, p'_2)\}$. Consider now the team automaton $\mathcal{T}$ over $\mathcal{S}$ with set of transitions $\delta = \{((p_1, p_2), a, (p'_1, p_2)), ((p'_1, p_2), b, (p'_1, p'_2))\}$. Note that $\mathcal{T}$ can execute action $a$ by $\mathcal{A}_1$ only when the component $\mathcal{A}_2$ is still in its local state $p_2$ and action $b$ can be executed only after $\mathcal{A}_1$ has reached $p'_1$. Now we consider the synchronous product defined by transforming $\mathcal{S}$ and $\mathcal{T}$: $\mathcal{A}'_1$ has a single action $[(p_1, p_2), a, (p'_1, p_2)]$ and $(p_1, [(p_1, p_2), a, (p'_1, p_2)], p'_1)$ as its only transition. $\mathcal{A}'_2$ has action $[(p'_1, p_2), b, (p'_1, p'_2)]$ and $(p_2, [(p'_1, p_2), b, (p'_1, p'_2)], p'_2)$ as its only transition. $\mathcal{X}(\mathcal{S}')$ is shown in Fig. 9. The sequence

$$\tau = (p_1, p_2)[(p'_1, p_2), b, (p'_1, p'_2)](p_1, p'_2)[(p_1, p_2), a, (p'_1, p_2)](p'_1, p'_2)$$

is a computation of $\mathcal{X}(\mathcal{S}')$, but $h(\tau) = (p_1, p_2)b(p_1, p'_2)a(p'_1, p'_2)$ is not a computation of $\mathcal{T}$. □

This is due to a property called *state-sharing* by which in a team automaton the occurrence of a synchronization depends in general also on the current states of the components not involved in the synchronization. The following definition describes the conditions for avoiding this type of global dependency in the synchronization:

**Definition 5** (*[20,9]*)**.** A team automaton $\mathcal{T}$ is *non-state-sharing* if for all actions $a$ whenever $(p, a, p') \in \delta$ then $(q, a, q') \in \delta$ for all states $q, q'$ such that $\mathtt{proj}_i(q) = \mathtt{proj}_i(p)$ and $\mathtt{proj}_i(q') = \mathtt{proj}_i(p')$ for all components $i$ such that $(\mathtt{proj}_i(p), a, \mathtt{proj}_i(p')) \in \delta_i$; and $\mathtt{proj}_i(q) = \mathtt{proj}_i(q')$ for all other $i$.

Thus in a non-state-sharing team automaton the possibility of executing a synchronization depends on local information only and also does not affect local states other than those of the components involved in the synchronization. This is illustrated in Fig. 10. Note the resemblance to the firing rule of Petri nets [26]. In fact, as demonstrated in [9] team automata which are non-state-sharing can be viewed as a particular kind of labeled Petri nets (or *token-based* systems [15] where only components that are actively involved can influence the enabledness of actions). It is easy to see that synchronous products are always non-state-sharing.

**Lemma 12.** *Assume that* $\mathcal{T}$ *is non-state-sharing. Let* $(p, [q, a, q'], p') \in \chi^{\mathcal{S}'}$. *Then* $(p, a, p') \in \delta$.

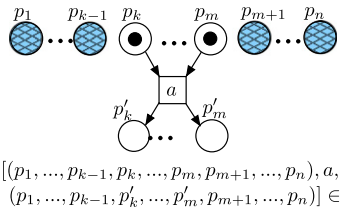



**Fig. 10.** Token-based transitions: shaded places are not involved in the synchronization.

**Proof.** First we observe that $(p, [q, a, q'], p') \in \chi^{\mathcal{S}'}$ implies that $(q, a, q') \in \delta$. Hence, since $\mathcal{T}$ is non-state-sharing, we are done once we have shown that $\mathtt{proj}_i(p) = \mathtt{proj}_i(q)$ and $\mathtt{proj}_i(p') = \mathtt{proj}_i(q')$ for all $1 \leq i \leq n$ such that $(\mathtt{proj}_i(q), a, \mathtt{proj}_i(q')) \in \delta_i$ and $\mathtt{proj}_i(p) = \mathtt{proj}_i(p')$, otherwise. Assume that $i$ is such that $(\mathtt{proj}_i(q), a, \mathtt{proj}_i(q')) \in \delta_i$. Then $[q, a, q'] \in \Sigma'_i$ which – by the definition of synchronous product – means that the $i$-th component participates in the synchronization $(p, [q, a, q'], p')$. Thus $(\mathtt{proj}_i(p), [q, a, q'], \mathtt{proj}_i(p')) \in \delta'_i$. From the definition of $\delta'_i$ it follows that $\mathtt{proj}_i(p) = \mathtt{proj}_i(q)$ and $\mathtt{proj}_i(p') = \mathtt{proj}_i(q')$. Now let $i$ be such that $(\mathtt{proj}_i(q), a, \mathtt{proj}_i(q')) \notin \delta_i$. Then $[q, a, q'] \notin \Sigma'_i$. Consequently, $(\mathtt{proj}_i(p), [q, a, q'], \mathtt{proj}_i(p')) \notin \delta'_i$ which implies that $\mathtt{proj}_i(p) = \mathtt{proj}_i(p')$. □

**Theorem 13.** *If $\mathcal{T}$ is non-state-sharing, then $h(\mathcal{C}_{\mathcal{X}(\mathcal{S}')}) = \mathcal{C}_{\mathcal{T}}$.*

**Proof.** From Theorem 11 we know that $\mathcal{C}_{\mathcal{T}} \subseteq h(\mathcal{C}_{\mathcal{X}(\mathcal{S}')})$. So we only have to prove the converse inclusion. Let $\tau = q_0[p_0, a_1, p_1]q_1[p_1, a_2, p_2] \cdots$ be a computation of $\mathcal{X}(\mathcal{S}')$. Thus $(q_{j-1}, [p_{j-1}, a_j, pj], q_j) \in \chi^{\mathcal{S}'}$ for each $j \geq 1$. Then, according to Lemma 12, $(q_{j-1}, a_j, q_j) \in \delta$ for each $j \geq 1$, which implies that $h(\tau) = q_0 a_1 q_1 a_2 \cdots \in \mathcal{C}_{\mathcal{T}}$ as desired. □

### *4.2. Compatibility of the initial system*

Given the (behavior-preserving) transformation of team automata into synchronous products, we can now investigate to what extent properties can be transferred from the synchronous product representation to the original automaton. Note that one cannot tell for any arbitrary synchronization strategy, whether there is a problem in the interaction unless compatibility for that specific strategy has been defined. Thus we have to be satisfied with detecting situations in the initial system that may represent violations to the correct interaction (compatibility) as investigated in this paper.

Let us return to the example of Fig. 9 and look at the new behavior incorporated in $\mathcal{X}(\mathcal{S}')$ if the system is state-sharing. Suppose that action $a$ ($b$) is an output (input) action for component $\mathcal{A}_1$, while it is an input (output) action for $\mathcal{A}_2$. Then one could say that in $\mathcal{T}$ receptiveness is violated in state $(p_1, p_2)$ in which action $a$ is output by $\mathcal{A}_1$, but not received in $\mathcal{A}_2$, and similarly for state $(p'_1, p_2)$ and action $b$. Since the role of actions is preserved, we have on the other hand four such receptiveness violations in $\mathcal{X}(\mathcal{S}')$, of which two do not correspond to real violations in $\mathcal{T}$.

For non-state-sharing team automata, the transformation does not lead to compatibility problems that did not exist before. In other words, if there is a compatibility violation (in the sense of no progress, non-receptiveness, or deadlock) in the synchronous product transformation, then there was one in the original TA. This observation leads to the following definition:

**Definition 6.** Let $\mathcal{T}$ be a non-state-sharing team over $\mathcal{S} = \{\mathcal{A}_i \mid 1 \leq i \leq n\}$, and $\mathcal{X}(\mathcal{S}')$ the corresponding synchronous product over the transformed composable system $\mathcal{S}'$. $\mathcal{T}$ has a receptive/non-communicating progress/deadlock-freeness violation at state $q$ if $\mathcal{X}(\mathcal{S}')$ does.

This definition lets the compatibility violations found in the transformed system be mapped back to the original team. This can only be done for non-state-sharing team automata, because of the one-to-one correspondence between the reachable states in $\mathcal{T}$ and in $\mathcal{X}(\mathcal{S}')$. Moreover, since the role of actions is preserved in the component RTSs, the actions of $\mathcal{S}'$ are communicating if and only if their original is communicating in $\mathcal{S}$. One should, however be aware that in general a team automaton may still violate certain compatibility properties which do not come to light in its synchronous product transformation. For $\{\mathcal{A}_1, \mathcal{A}_2\}$ of Fig. 9, define the team automaton $\mathcal{T}_0$ by choosing $\delta = \emptyset$. Then $\delta'_1 = \emptyset$ and $\delta'_2 = \emptyset$. Therefore $\chi^{\mathcal{S}'} = \emptyset$, and hence this synchronous product has a trivial compatibility relation even though in the composable system $\{\mathcal{A}_1, \mathcal{A}_2\}$ receptiveness is violated at the only reachable state in $\mathcal{T}_0$.

Finally, as remarked before, the conversion of a team automaton into a synchronous product as considered here does not change its set of potential states and if the original TA is non-state-sharing also the reachable states remain the same. However, in the case of a state-sharing team automaton, there may be an exponential blow-up of the number of reachable states. This further complicates finding practical algorithms for this type of TA.

## 5. Related work and discussion

In this paper we have proposed an extension of Input/Output compatibility to reactive systems consisting of multiple components. I/O compatibility was proposed originally in [12] motivated by the ultimate aim of implementing asynchronous logic circuits. It is inspired by the idea of *conformance* from Dill [18]. Conformance, however, is defined in terms of executed

sequences of actions, in contrast to the state-based definition of I/O compatibility for components specified as transition systems.

The criteria for I/O compatibility (receptiveness, internal progress, deadlock-free communication) derive from the wish to guarantee certain safety and liveness properties in the composed system. Moreover, as shown in [12], the additional requirement that each component should be livelock-free alleviates the complexity of the methods to check some specific liveness properties like deadlock-freeness.

Receptiveness can be seen as a weak version of the input-enabledness condition required in I/O automata [25]. Input-enabledness may be too strong for multi-component systems where assumptions about the environment are made. For instance, reactive systems are often used to model hardware components which require a particular protocol on their input signals. Other examples would come from object-oriented software components. An optimistic way of composing components concerned with receptiveness without requiring input-enabledness is considered in the Interface Automata approach of [2,3]. This approach allows compositions for which there exists at least one environment that makes it possible to escape from falling into error states. There is, however, no idea of progress in this concept of compatibility since the environment is supposed to steer the system away from illegal states. This approach has been followed in several papers, e.g., [17] in the context of Timed I/O Automata.

More recently, [23] has investigated modal transition systems (automata with *may* and *must* transitions) and the relation between alternating simulations as used in Interface Automata and observational modal transition systems refinement. Also this work is inspired by the conformance from [18] and has the concept of *error state* in which one component can output something that the other component might be unable to receive. To deal with these error states the composition is *pruned* and the components are said to be compatible when the resulting pruned composition is non-empty. In [7] various notions of modal refinement and compatibility are discussed starting from the Modal I/O automata of [23]. In particular, a concept of *weak modal compatibility* is presented by which input actions *may* be sent to a partner if the other *must* be able to receive it eventually (after performing some internal *must* steps).

The recent work in [22] on *communication-safe component assemblies* extends the theory of [7] by introducing the notion of *communication-safety*, which extends weak modal compatibility by allowing also must-communication transitions before receiving an event sent by another component. Additionally, the intuition behind deadlock-freedom in [22] is similar to the approach followed in this paper, and the incremental construction of communication-safety assemblies is presented by considering conditions on the interfaces that form the assembly, also with the same goals as the ones we have in Section 3.2. However, in [22] the synchronous product is the only composition operator considered which leads to different techniques.

In [6] it is investigated to what extent data-flow and control-flow aspects can be combined when components with data states interact through binary connectors. Specifically, the impact on compatibility notions is considered. The composition operator used is related to the synchronous product of symbolic transitions systems from [21]. Regarding compatibility, the essential requirement is that any output by one component meets the partner component in a state where it expects the corresponding input. These ideas are formalized in terms of a behavioral compatibility relation that extends the concept of strong compatibility in [7], and similar notions in [2,23]. Interestingly, an extension to $n$-ary connectors is suggested though – as the authors remark – the composition may be more involved and in particular the contract idea behind the compatibility has to be reconsidered. It might be that the approach presented in this paper could be of help with the concept of an $n$-ary compatibility relation and more general compositions than the synchronous product.

The motivation to retain in the models behavior that can represent potential errors in a multi-component system is also present in [1]. There, a new composition operator (*the consent* operator) is proposed to capture not only the traditional synchronous product, but also a set of behaviors for representing *bad activity* (our receptiveness), *no activity* (similar to our deadlock-freeness) and *divergence*. The focus in that work is to use this new operator (which allows only one type of synchronization) in order to check the atomicity of a dynamic update, while in the approach presented here we do not restrict the synchronization strategy, and instead put the effort into providing a translation of arbitrary synchronization into the synchronous product with the ultimate goal of finding errors via the compatibility notion introduced.

We have addressed the following questions in our attempt to generalize the work presented in [12]: How to define correct interaction (compatibility) in a multi-component environment and how to deal with synchronization strategies different from the synchronous product? In the general set-up studied in this paper, components not only communicate but they may also collaborate (synchronize on the execution of shared input or output actions). Consequently, an important novel issue is how to identify partners in communication to define receptiveness and deadlock-freeness. Here we have chosen to use domains for this purpose. When the synchronous product is used to compose the system, the components have to collaborate, but in other synchronization strategies it may be that not always are all components sharing an action required to take part in the execution of that action. For this situation we have obtained an initial result and we have formulated a proposal that makes it possible to identify potential problems for arbitrary synchronization strategies.

We believe that collaboration is becoming more common as a cooperating strategy in modern system design. Therefore compatibility checks that take this into account will help significantly in the development of correct-by-construction multi-component systems and the ideas presented in this paper should be considered as a first step in the development of techniques supporting the design, analysis and verification of multi-component systems. So, compatibility in the context of arbitrary synchronization strategies should be further investigated. Another topic of future research would be the synthesis of multi-component systems. In [14,12], synthesis rules are presented for asynchronous circuits modeled as Petri nets. Transformations are presented that preserve I/O compatibility of a component with its environment. Using the insights

gained in this paper we plan to investigate whether this approach can be extended to systems with more components and how to modify the concurrency of events in multi-component systems while preserving compatibility.

**Acknowledgments**

We are grateful to the reviewers for their careful reading and constructive criticism which helped us to improve the presentation of our ideas and results. This work was supported by the FORMALISM project (TIN2007-66523) and by the project TIN2011-22484.

## Appendix. Definition of I/O compatibility from [12]

**Definition 7.** Let $\mathcal{S} = \{\mathcal{A}_1, \mathcal{A}_2\}$ be a composable system with $\mathcal{A}_i = (Q_i, \Sigma_i, \delta_i, I_i)$, for $i = 1, 2$, such that $\Sigma_{1,\mathtt{inp}} = \Sigma_{2,\mathtt{out}}$ and $\Sigma_{1,\mathtt{out}} = \Sigma_{2,\mathtt{inp}}$. Then $\mathcal{R} \subseteq Q_1 \times Q_2$ is an *I/O-compatibility relation for* $\mathcal{S}$ if $I_1 \times I_2 \subseteq \mathcal{R}$ and for all $(p_1, p_2) \in \mathcal{R}$ the following conditions are satisfied.

1. *Internal Progress*:
   (i) if $a \in \Sigma_{1,\mathtt{int}}$ and $(p_1, a, p_1') \in \delta_1$ for some $p_1'$, then $(p_1', p_2) \in \mathcal{R}$;
   (ii) if $a \in \Sigma_{2,\mathtt{int}}$ and $(p_2, a, p_2') \in \delta_2$ for some $p_2'$, then $(p_1, p_2') \in \mathcal{R}$.
2. *Receptiveness*:
   (i) if $a \in \Sigma_{1,\mathtt{out}}$ and $(p_1, a, p_1') \in \delta_1$ for some $p_1'$, then $a\ \mathtt{en}_{\mathcal{A}_2}\ p_2$ and $(p_1', p_2') \in \mathcal{R}$ for all $p_2'$ such that $(p_2, a, p_2') \in \delta_2$;
   (ii) if $a \in \Sigma_{2,\mathtt{out}}$ and $(p_2, a, p_2') \in \delta_2$ for some $p_2'$, then $a\ \mathtt{en}_{\mathcal{A}_1}\ p_1$ and $(p_1', p_2') \in \mathcal{R}$ for all $p_1'$ such that $(p_1, a, p_1') \in \delta_1$.
3. *Deadlock-freeness*:
   (i) if $a\ \mathtt{en}_{\mathcal{A}_1}\ p_1$ implies that $a \in \Sigma_{1,\mathtt{inp}}$, then there is an action $b \in \Sigma_2 - \Sigma_{2,\mathtt{inp}}$ such that $b\ \mathtt{en}_{\mathcal{A}_2}\ p_2$.
   (ii) if $a\ \mathtt{en}_{\mathcal{A}_2}\ p_2$ implies that $a \in \Sigma_{2,\mathtt{inp}}$, then there is an action $b \in \Sigma_1 - \Sigma_{1,\mathtt{inp}}$ such that $b\ \mathtt{en}_{\mathcal{A}_1}\ p_1$.

### A.1. Proof of Theorem 1

**Proof.** To start with, we observe that both I/O compatibility and compatibility require that $I_1 \times I_2 \subseteq \mathcal{R}$. Next, we compare the various conditions on $\mathcal{R}$ as formulated in Definitions 4 and 7.
Let $(p_1, p_2) \in \mathcal{R}$.

Considering the progress conditions, we observe that, since $\mathcal{S}$ is complete, the internal actions of $\mathcal{S}$ are its only non-communicating actions. Moreover, since each such action $a$ appears in exactly one of the components, it follows that $((p_1, p_2), a, (q_1, q_2)) \in \chi^{\mathcal{S}}$ if and only if either $(p_1, a, q_1) \in \delta_1$ and $q_2 = p_2$ or $(p_2, a, q_2) \in \delta_2$ with $q_1 = p_1$. It then follows immediately that the Non-communicating Progress condition of Definition 4 and the Internal Progress condition of Definition 7 are reformulations of one another.

With respect to receptiveness, we observe that, since $\mathcal{S}$ is complete, every communicating action $a$ appears as input in one component and as output in the other component. Moreover, $((p_1, p_2), a, (q_1, q_2)) \in \chi^{\mathcal{S}}$ if and only if both $(p_1, a, q_1) \in \delta_1$ and $(p_2, a, q_2) \in \delta_2$. Consequently, also the receptiveness conditions are simple reformulations of one another.

Finally we consider deadlock-freeness.

First, suppose that $\mathcal{R}$ satisfies the deadlock-freeness from Definition 4. We only prove condition (i) of the deadlock-freeness from Definition 7 as the proof for (ii) is completely symmetric.
Assume that no output or internal actions of $\mathcal{A}_1$ are enabled at $p_1$. Since $\mathcal{A}$ is not terminating, there exists an $a \in \Sigma_{1,\mathtt{inp}}$ such that $a\ \mathtt{en}_{\mathcal{A}_1}\ p_1$. The completeness of $\mathcal{S}$ implies that $\mathtt{inpdom}(\mathcal{S}, a) = \{1\}$. Then by the deadlock-freeness property of Definition 4, there exists $b \in \Sigma_1 \cup \Sigma_2$ and $(q_1, q_2) \in Q_1 \times Q_2$ such that $((p_1, p_2), b, (q_1, q_2)) \in \chi^{\mathcal{S}}$. If $b \in \Sigma_{1,\mathtt{inp}}$, then by the completeness of $\mathcal{S}$ we have $b \in \Sigma_{2,\mathtt{out}}$ and $b\ \mathtt{en}_{\mathcal{A}_2}\ p_2$, as required. If $b \in \Sigma_{2,\mathtt{inp}}$, then again by the completeness of $\mathcal{S}$ we have $b \in \Sigma_{1,\mathtt{out}}$ and moreover $b\ \mathtt{en}_{\mathcal{A}_1}\ p_1$ by the definition of the synchronous product, a contradiction with the assumption above. Consequently, $b$ is either an internal action or an output action of $\mathcal{A}_2$ and $b\ \mathtt{en}_{\mathcal{A}_2}\ p_2$ holds, as required.

Conversely, assume that $\mathcal{R}$ satisfies the conditions from Definition 7, in particular deadlock-freeness and receptiveness. Let $a$ be a communicating action of $\mathcal{S}$ such that $\mathcal{A}_1$ is the component in which it appears as an input action and such that $a\ \mathtt{en}_{\mathcal{A}_1}\ p_1$.
If in $\mathcal{A}_1$ only input actions are enabled at $p_1$, it follows from (i) of the deadlock-freeness of Definition 7 that there is an action $b \in \Sigma_2 - \Sigma_{2,\mathtt{inp}}$ such that $b\ \mathtt{en}_{\mathcal{A}_2}\ p_2$. If this $b$ is an internal action, $((p_1, p_2), b, (p_1, q_2)) \in \chi^{\mathcal{S}}$ for some $q_2 \in Q_2$ as required. If $b$ is an output action of $\mathcal{A}_2$, then by the receptiveness of $\mathcal{R}$, we have $((p_1, p_2), b, (q_1, q_2)) \in \chi^{\mathcal{S}}$ for some $(q_1, q_2) \in Q_1 \times Q_2$, again as required.
On the other hand if there is an action $b \in \Sigma_1 - \Sigma_{1,\mathtt{inp}}$ such that $b\ \mathtt{en}_{\mathcal{A}_1}\ p_1$, it follows, again using the receptiveness of $\mathcal{R}$, that $((p_1, p_2), b, (q_1, q_2)) \in \chi^{\mathcal{S}}$ for some $(q_1, q_2) \in Q_1 \times Q_2$. □

### A.2. Proof of Theorem 5

**Proof.** First of all we observe that, because internal actions do not communicate, $\mathcal{X}(\{\mathcal{A}_1, \mathcal{A}_2\})$ is livelock-free if and only if both $\mathcal{A}_1$ and $\mathcal{A}_2$ are livelock-free. Next, let $\mathcal{R}$ be a compatibility relation for $\mathcal{S}$. Then, as we now set out to prove, $\mathcal{R}' = \{((p_1, p_2), p_3) | (p_1, p_2, p_3) \in \mathcal{R}\}$ is a compatibility relation for $\mathcal{S}'$.

Clearly, $\mathcal{R}'$ includes $(I_1 \times I_2) \times I_3$.
Let $p = ((p_1, p_2), p_3) \in \mathcal{R}'$ with $(p_1, p_2, p_3) \in \mathcal{R}$.
*Non-communicating progress*: Let $a$ be a non-communicating action of $\mathcal{S}'$ and assume that $a$ is enabled at $(p_1, p_2)$ and $p_3$, respectively whenever $a$ is an action of $\mathcal{X}(\mathcal{A}_1, \mathcal{A}_2)$ and $\mathcal{A}_3$, respectively. Let $(p, a, p') \in \chi_a^{\mathcal{S}'}$ for some $p' = ((p'_1, p'_2), p'_3)$. We have to prove that $p' \in \mathcal{R}'$. Depending on the domain of action $a$, three cases arise.

If $a$ does not appear as an action in $\mathcal{X}(\{\mathcal{A}_1, \mathcal{A}_2\})$, then it is only an action of $\mathcal{A}_3$ and hence a non-communicating action of $\mathcal{S}$. Moreover, $p' = ((p_1, p_2), p'_3)$ and $((p_1, p_2, p_3), a, (p_1, p_2, p'_3)) \in \chi_a^{\mathcal{S}}$. Thus $(p_1, p_2, p'_3) \in \mathcal{R}$ by the non-communicating progress property of $\mathcal{S}$ and so $p' \in \mathcal{R}'$.

If action $a$ does not appear as an action in $\mathcal{A}_3$, we have $p'_3 = p_3$. Moreover, $a$ can be a non-communicating or a communicating action in $\mathcal{S}$. In the first case, it follows that $((p_1, p_2, p_3), a, (p'_1, p'_2, p_3)) \in \chi^{\mathcal{S}}$. and so $(p'_1, p'_2, p_3) \in \mathcal{R}$ by the non-communicating progress property of $\mathcal{R}$. Hence $p' \in \mathcal{R}'$. In the latter case, $a$ is also a communicating action in $\{\mathcal{A}_1, \mathcal{A}_2\}$. Hence, $((p_1, p_2, p_3), a, (p'_1, p'_2, p_3)) \in \chi^{\mathcal{S}}$ which now implies that $(p'_1, p'_2, p_3) \in \mathcal{R}$ by the receptiveness of $\mathcal{R}$. Hence $p' \in \mathcal{R}'$.

Finally, if action $a$ appears both in $\mathcal{X}(\{\mathcal{A}_1, \mathcal{A}_2\})$ and $\mathcal{A}_3$, then $a$ can be a non-communicating action in $\mathcal{S}$ (input in $\mathcal{A}_3$ in combination with input in $\mathcal{A}_1$ or $\mathcal{A}_2$ or both; output in $\mathcal{A}_3$ in combination with output in $\mathcal{A}_1$ or $\mathcal{A}_2$ or both) or a communicating action in $\mathcal{S}$ (output in $\mathcal{A}_1$ and input in $\mathcal{A}_2$ or vice versa and output in $\mathcal{A}_3$).
In the first case, $((p_1, p_2, p_3), a, (p'_1, p'_2, p'_3)) \in \chi_a^{\mathcal{S}}$. Thus $(p'_1, p'_2, p'_3) \in \mathcal{R}$ by the non-communicating progress property of $\mathcal{S}$ and so $p' \in \mathcal{R}'$. In the latter case, $a$ is also a communicating action in $\{\mathcal{A}_1, \mathcal{A}_2\}$. Hence, $((p_1, p_2, p_3), a, (p'_1, p'_2, p'_3)) \in \chi^{\mathcal{S}}$ which now implies that $(p'_1, p'_2, p'_3) \in \mathcal{R}$ by the receptiveness of $\mathcal{R}$. Hence $p' \in \mathcal{R}'$.
*Receptiveness*: Let $a \in \Sigma'_{\mathtt{com}}$ and assume that if $a$ is an output action of $\mathcal{X}(\mathcal{A}_1, \mathcal{A}_2)$ or $\mathcal{A}_3$, then $a$ is enabled in $\mathcal{S}$ at $(p_1, p_2)$ or $p_3$, respectively, We have to prove that $a$ is also enabled for the other component of $\mathcal{S}'$. Moreover, if $(p, a, p') \in \chi_a^{\mathcal{S}'}$ for some $p' = ((p'_1, p'_2), p'_3)$, then $p' \in \mathcal{R}'$.

First of all, it is immediate that $a$ is a communicating action in $\mathcal{S}$. Moreover, $a$ is enabled at $p_i$ in every $\mathcal{A}_i$ for which it is an output action. From the receptiveness of $\mathcal{R}$ it then follows that $a$ is also enabled at $p_i$ in every $\mathcal{A}_i$ for which it is an input action. If $a$ is an input action of $\mathcal{A}_3$ in $\mathcal{S}'$, then it is an input action of $\mathcal{A}_3$ in $\mathcal{S}'$. Also, if $a$ is an input action of $\mathcal{X}(\{\mathcal{A}_1, \mathcal{A}_2\})$, then it is an input action of $\mathcal{A}_1$ or $\mathcal{A}_2$ or both. Consequently, $a$ is enabled in $\mathcal{S}$ for the component for which it is an input action. Now, let $(p, a, p') \in \chi_a^{\mathcal{S}'}$ for some $p' = ((p'_1, p'_2), p'_3)$, then $((p_1, p_2, p_3), a, (p'_1, p'_2, p'_3)) \in \chi^{\mathcal{S}}$ with $(p'_1, p'_2, p'_3) \in \mathcal{R}$ by the receptiveness of $\mathcal{R}$. Hence $p' \in \mathcal{R}'$.
*Deadlock-freeness*: Let $a \in \Sigma'_{\mathtt{com}}$ be such that $a$ is an input action of $\mathcal{X}(\mathcal{A}_1, \mathcal{A}_2)$ or $\mathcal{A}_3$ (not both) and $a$ is enabled in $\mathcal{S}$ at $(p_1, p_2)$ or $p_3$, respectively. We have to prove that there exists an action $b$ in $\mathcal{S}$ such that $(p, b, p') \in \chi^{\mathcal{S}'}$ for some $p' = ((p'_1, p'_2), p'_3)$.

If $a$ is an input action of $\mathcal{A}_3$ in $\mathcal{S}$, then it is an input action of $\mathcal{A}_3$ in $\mathcal{S}'$ and enabled at $p_3$. If $a$ is an input action of $\mathcal{A}_1$ or $\mathcal{A}_2$, then it is enabled at $p_1$ or $p_2$, respectively, by the enabledness of $a$ at $(p_1, p_2)$ in $\mathcal{X}(\mathcal{A}_1, \mathcal{A}_2)$. By the deadlock-freeness of $\mathcal{R}$, we know that there exists an action $b$ in $\mathcal{S}$ such that $((p_1, p_2, p_3), b, (p'_1, p'_2, p'_3)) \in \chi^{\mathcal{S}}$ for some $(p'_1, p'_2, p'_3)$. So, $(p, b, p') \in \chi^{\mathcal{S}'}$ with $p' = ((p'_1, p'_2), p'_3)$. □

### A.3. Proof of Lemma 6

**Proof.** Since internal actions do not communicate, $\mathcal{X}(\{\mathcal{A}_1, \mathcal{A}_2\})$ is livelock-free if and only if both $\mathcal{A}_1$ and $\mathcal{A}_2$ are livelock-free. Let $\mathcal{R}$ be a compatibility relation for $\mathcal{S}$. We will prove that $\mathcal{R}' = \{(p_1, p_2, p_3) | ((p_1, p_2), p_3) \in \mathcal{R}$ and $(p_1, p_2)$ reachable in $\mathcal{X}(\mathcal{A}_1, \mathcal{A}_2)\}$ is a compatibility relation for $\mathcal{S}'$.

Clearly, $\mathcal{R}'$ includes $I_1 \times I_2 \times I_3$.
Let $p = (p_1, p_2, p_3) \in \mathcal{R}'$ with $((p_1, p_2), p_3) \in \mathcal{R}$ and $(p_1, p_2)$ reachable in $\mathcal{X}(\mathcal{A}_1, \mathcal{A}_2)$.
*Non-communicating progress*: Let $a$ be a non-communicating action of $\mathcal{S}'$ and assume that $a$ is enabled at $p_i$ whenever $a$ is an action of $\mathcal{A}_i$ for $i = 1, 2, 3$. Let $(p, a, p') \in \chi_a^{\mathcal{S}'}$ for some $p' = (p'_1, p'_2, p'_3)$. We have to prove that $p' \in \mathcal{R}'$ and $(p'_1, p'_2)$ reachable in $\mathcal{X}(\mathcal{A}_1, \mathcal{A}_2)$. Observe first that $a$ is also a non-communicating action of $\mathcal{S}$. Moreover, in case $a$ is an input (output) action of $\mathcal{A}_1$ or $\mathcal{A}_2$ or both, it is enabled at $(p_1, p_2)$ in $\mathcal{X}(\mathcal{A}_1, \mathcal{A}_2)$. Consequently, $a$ is enabled at $((p_1, p_2), p_3)$ and $((p_1, p_2), p_3), a, ((p'_1, p'_2), p'_3)) \in \chi_a^{\mathcal{S}}$ independent of whether it is an input or output or internal action of $\mathcal{S}'$. Then by the non-communicating progress property of $\mathcal{R}$ it follows that $((p'_1, p'_2), p'_3)) \in \mathcal{R}$ and hence $p' \in \mathcal{R}'$.
*Receptiveness*: Let $a$ be a communicating action of $\mathcal{S}'$ and assume that $a$ is enabled at $p_i$ whenever $a$ is an output action of $\mathcal{A}_i$ for $i = 1, 2, 3$. We have to prove that $a$ is also enabled for those $\mathcal{A}_i$ for which it is an input action and, moreover, if $(p, a, p') \in \chi_a^{\mathcal{S}'}$ for some $p' = (p'_1, p'_2, p'_3)$, then $p' \in \mathcal{R}'$.

If $a$ is not a communicating action of $\mathcal{S}$, then it must be the case that $a$ is a communicating action of $\{\mathcal{A}_1, \mathcal{A}_2\}$ and $a$ is not an input action of $\mathcal{A}_3$. The receptiveness of $\{\mathcal{A}_1, \mathcal{A}_2\}$ implies that $a$ is enabled at $(p_1, p_2)$. Thus, $a$ is enabled at the $\mathcal{A}_i$ for which it is an input action. Further, $((p_1, p_2), p_3), a, ((p'_1, p'_2), p'_3) \in \chi_a^{\mathcal{S}}$ and the non-communication progress property for $\mathcal{R}$ yields $((p'_1, p'_2), p'_3) \in \mathcal{R}$. Hence $p' \in \mathcal{R}'$.

If $a$ is a communicating action of $\mathcal{S}$, then either $a$ is a communicating action of $\{\mathcal{A}_1, \mathcal{A}_2\}$ and an input action of $\mathcal{A}_3$; or $a$ is an output (input) action of either $\mathcal{A}_1$ or $\mathcal{A}_2$ or both, and an input (output) action of $\mathcal{A}_3$. In the first case we use again the receptiveness of $\{\mathcal{A}_1, \mathcal{A}_2\}$ together with the receptiveness of $\mathcal{R}$, to conclude that $a$ is enabled at those $\mathcal{A}_i$ for which it is an input action and $((p_1, p_2), p_3), a, ((p'_1, p'_2), p'_3) \in \chi_a^{\mathcal{S}}$ with $((p'_1, p'_2), p'_3) \in \mathcal{R}$. Hence in this case $p' \in \mathcal{R}'$. For the latter case, the receptiveness of $\mathcal{R}$ suffices to finish the proof.

*Deadlock-freeness*: Let $a$ be a communicating action of $\mathcal{S}'$ and assume that $a$ is enabled at $p_i$ whenever $a$ is an input action of $\mathcal{A}_i$ for $i = 1, 2, 3$. We have to prove that there exists an action $b$ in $\mathcal{S}$ such that $(p, b, p') \in \chi^{\mathcal{S}'}$ for some $p' = (p'_1, p'_2, p'_3)$. If $a$ is an input action of $\mathcal{X}(\mathcal{A}_1, \mathcal{A}_2)$ or an input action of $\mathcal{A}_3$, then the deadlock-freeness of $\mathcal{S}$ implies that there exist $b$ and $((p'_1, p'_2), p'_3)$ such that $((p_1, p_2), p_3), b, ((p'_1, p'_2), p'_3) \in \chi^{\mathcal{S}}$ and we are done. The remaining case is that $a$ is communicating in $\mathcal{X}(\mathcal{A}_1, \mathcal{A}_2)$ and not an input action of $\mathcal{A}_3$. Now we use the deadlock-freeness of $\{\mathcal{A}_1, \mathcal{A}_2\}$ to infer that there exists an action $c$ and $(p'_1, p'_2)$ such that $((p_1, p_2), c, (p'_1, p'_2)) \in \chi^{\{\mathcal{A}_1, \mathcal{A}_2\}}$. Again we have different cases to distinguish:

If $c$ is not an action of $\mathcal{A}_3$, we are done: $(p, c, (p'_1, p'_2, p_3)) \in \chi^{\mathcal{S}'}$.

The remaining cases are that (i) either $c$ is an input action of $\mathcal{X}(\mathcal{A}_1, \mathcal{A}_2)$ and an output action of $\mathcal{A}_3$ or (ii) $c$ is an output action of $\mathcal{X}(\mathcal{A}_1, \mathcal{A}_2)$ and an input action of $\mathcal{A}_3$ or (iii) $c$ is an output (input) action of $\mathcal{X}(\mathcal{A}_1, \mathcal{A}_2)$ and an output (input, respectively) action of $\mathcal{A}_3$.

In case (i) – as before – the deadlock-freeness of $\mathcal{S}$ implies that there exist $((p'_1, p'_2), p'_3)$ such that $((p_1, p_2), p_3), c, ((p'_1, p'_2), p'_3) \in \chi^{\mathcal{S}}$ and we are done.

In case (ii) – as before – the receptiveness of $\mathcal{R}$ implies that there exist $b$ and $((p'_1, p'_2), p'_3)$ such that $((p_1, p_2), p_3), b, ((p'_1, p'_2), p'_3) \in \chi^{\mathcal{S}}$ and we are done.

In case (iii) due to the fact that $\mathcal{A}_3$ collaborates with $\mathcal{X}(\mathcal{A}_1, \mathcal{A}_2)$, there exists $((p'_1, p'_2), p'_3)$ such that $((p_1, p_2), p_3), c, ((p'_1, p'_2), p'_3) \in \chi^{\mathcal{S}}$ and we are done. □

## References


[1] J. Adámek, F. Plasil, Component composition errors and update atomicity: static analysis, Journal of Software Maintenance and Evolution: Research and Practice 17 (5) (2005) 363–377.
[2] L. de Alfaro, T.A. Henzinger, Interface automata, in: ESEC/SIGSOFT FSE (2001) 109–120.
[3] L. de Alfaro, T.A. Henzinger, Interface-based design, in: Engineering Theories of Software-Intensive Systems, Marktoberdorf Summer School, 2004.
[4] J. Armstrong, Erlang, Communications of ACM 53 (9) (2010) 68–75.
[5] A. Arnold, Finite Transition Systems, Prentice Hall, 1994.
[6] S.S. Bauer, R. Hennicker, S. Janisch, Behaviour protocols for interacting stateful components, Electronic Notes in Theoretical Computer Science 263 (2010) 47–66.
[7] S.S. Bauer, P. Mayer, A. Schroeder, R. Hennicker, On weak modal compatibility, refinement, and the MIO workbench, in: J. Esparza, R. Majumdar (Eds.), TACAS, in: Lecture Notes in Computer Science, vol. 6015, Springer, 2010, pp. 175–189.
[8] M.H. ter Beek, Team automata—a formal approach to the modeling of collaboration between system components, Ph.D. Thesis, Leiden Institute of Advanced Computer Science, Leiden University, 2003.
[9] M.H. ter Beek, C. Ellis, J. Kleijn, G. Rozenberg, Synchronizations in team automata for groupware systems, Computer Supported Cooperative Work—The Journal of Collaborative Computing 12 (1) (2003) 21–69.
[10] G. Berry, The esterel v5 language primer. Technical Report, Centre de Mathématiques Appliquées, Ecole des Mines and INRIA, 2004.
[11] J. Carmona, Structural methods for the synthesis of well-formed concurrent specifications, Ph.D. Thesis, Universitat Politècnica de Catalunya (UPC), Mar. 2004.
[12] J. Carmona, J. Cortadella, Input/output compatibility of reactive systems, in: Fourth International Conference on Formal Methods in Computer-Aided Design, FMCAD, Portland, Oregon, USA, Springer-Verlag, 2002, pp. 360–377.
[13] J. Carmona, J. Cortadella, M. Kishinevsky, A. Taubin, Elastic circuits, IEEE Transactions on CAD of Integrated Circuits and Systems 28 (10) (2009) 1437–1455.
[14] J. Carmona, J. Cortadella, E. Pastor, Synthesis of reactive systems: application to asynchronous circuit design, in: J. Cortadella, A. Yakovlev, G. Rozenberg (Eds.), Advances in Concurrency and Hardware Design, ACHD, in: Lecture Notes in Computer Science, vol. 2549, Springer-Verlag, 2002, pp. 108–151.
[15] J. Carmona, J. Kleijn, Interactive behaviour of multi-component systems, in: Workshop on Token-Based Computing, ToBaCo'04, 2004, pp. 27–31.
[16] D.M. Chapiro, Globally-Asynchronous Locally-Synchronous Systems, Ph.D. Thesis, Stanford University, Oct. 1984.
[17] A. David, K.G. Larsen, A. Legay, U. Nyman, A. Wasowski, Timed I/O automata: a complete specification theory for real-time systems, in: K.H. Johansson, W. Yi (Eds.), HSCC, ACM, 2010, pp. 91–100.
[18] D.L. Dill, Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits, in: ACM Distinguished Dissertations, MIT Press, 1989.
[19] C.A. Ellis, Team automata for groupware systems, in: Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge, Phoenix, Arizona, ACM Press, New York, 1997, pp. 415–424.
[20] G. Engels, L. Groenewegen, Towards team-automata-driven object-oriented collaborative work, in: W. Brauer, H. Ehrig, J. Karhumäki, A. Salomaa (Eds.), Formal and Natural Computing, in: Lecture Notes in Computer Science, vol. 2300, Springer, 2002, pp. 257–276.
[21] F. Fernandes, J.-C. Royer, The STSLib project: towards a formal component model based on sts, Electronic Notes in Theoretical Computer Science 215 (2008) 131–149.
[22] R. Hennicker, A. Knapp, Modal interface theories for communication-safe component assemblies, in: A. Cerone, P. Pihlajasaari (Eds.), ICTAC, in: Lecture Notes in Computer Science, vol. 6916, Springer, 2011, pp. 135–153.
[23] K.G. Larsen, U. Nyman, A. Wasowski, Modal I/O automata for interface and product line theories, in: R.D. Nicola (Ed.), ESOP, in: Lecture Notes in Computer Science, vol. 4421, Springer, 2007, pp. 64–79.
[24] N.A. Lynch, M.R. Tuttle, Hierarchical correctness proofs for distributed algorithms, in: Sixth Annual ACM Symposium on Principles of Distributed Computing, Sept. 1987, pp. 137–151.
[25] N.A. Lynch, M.R. Tuttle, An Introduction to Input/Output Automata, in: CWI-Quarterly, volume 2, 1989, pp. 219–246. Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.
[26] W. Reisig, G. Rozenberg (Eds.), Lectures on petri nets I: basic models, advances in petri nets, the volumes are based on the advanced course on petri nets, held in Dagstuhl, September 1996, in: Lecture Notes in Computer Science, vol. 1491, Springer, 1998.