Team automata : a formal approach to the modeling of collaboration between system components

Beek, M.H. ter

5. Team Automata

In the preceding two chapters we have prepared the basis for team automata.

In Chapter 3 we have defined automata underlying the *component automata* that team automata are built on. In Chapter 4 we consequently defined synchronized automata over sets of automata as a way to coordinate the interactions of those automata. Team automata are defined similar to synchronized automata, but they coordinate component automata rather than automata. The extra feature of component automata with respect to automata is a classification of their set of actions into *input*, *output*, and *internal* actions. *Subteams* of team automata are defined analogous to the subautomata of synchronized automata and we show how to iteratively build team automata over team automata similar to the iterative construction of synchronized automata.

The extra feature of component automata now allows us to characterize more types of synchronization and more *predicates of synchronization* by using the classification of their sets of actions. Consequently *maximal-syn team automata* are defined with respect to a given type of synchronization *syn*, similar to the way we did this in the context of synchronized automata. Finally, also this chapter is concluded with a study of the effect that synchronizations have on the inheritance of the automata-theoretic properties introduced in Section 3.2.

5.1 Definitions

Throughout this section we will occasionally illustrate our definitions using simple examples of coffee vending machines and their customers. This class of examples is very common in the literature on formal methods. Through these examples we thus hope to facilitate an interesting comparison of the team automata framework with models such as, e.g., (Theoretical) Communicating Sequential Processes (see, e.g., [Hoa78], [BHR84], and [Hoa85]) and Input/Output automata (see, e.g., [Tut87], [LT87], [LT89], and [Lyn96]). A survey can be found in [Shi97].

5.1.1 Component Automata

Team automata are built from component automata.

A component automaton is an automaton together with a classification of its actions. The actions are divided into two main categories. *Internal* actions have strictly local visibility and can thus not be used for collaboration with other components, whereas *external* actions are observable by other components. These external actions can be used for collaboration between components and are divided into two more categories: *input* actions and *output* actions. As formulated in [Ell97]: "input actions are not under the local system's control and are caused by another non-local component, the output actions are under the system's control and are externally observable by other components, and internal actions are under the local system's control but are not externally observable".

When describing a component automaton with the system to be modeled in mind, one of the design issues that thus has to be considered is the role of the actions within that component in relation to the other components within the system.

Definition 5.1.1. *A* component automaton *is a construct* $\mathcal{C} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$*, where*

$(Q, \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}, \delta, I)$ *is an automaton,*
Σ_{inp} *is the* input alphabet *of* $\mathcal{C}$*,*
Σ_{out} *is the* output alphabet *of* $\mathcal{C}$*, and*
Σ_{int} *is the* internal alphabet *of* $\mathcal{C}$ *such that* Σ_{inp}*,* Σ_{out}*, and* Σ_{int} *are mutually disjoint.* □

The automaton $(Q, \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}, \delta, I)$ of a component automaton $\mathcal{C} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ is called the *underlying automaton* of $\mathcal{C}$ and it is denoted by $\mathrm{und}(\mathcal{C})$. Moreover, the elements of the input, output, and internal alphabet of $\mathcal{C}$ are called the *input*, *output*, and *internal actions* of $\mathcal{C}$, respectively. We refer to $\mathcal{C}$ as the *trivial component automaton* if $\mathcal{C} = (\varnothing, (\varnothing, \varnothing, \varnothing), \varnothing, \varnothing)$. Finally, if both Q and $\Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$ are finite, then $\mathcal{C}$ is called a *finite component automaton*.

Definition 5.1.2. *Let* $\mathcal{C} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ *be a component automaton. Then*

(1) the (full) alphabet *of* $\mathcal{C}$ *is denoted by* Σ *and is defined as* $\Sigma = \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$*,*

(2) the external alphabet *of* $\mathcal{C}$ *is denoted by* Σ_{ext} *and is defined as* $\Sigma_{ext} = \Sigma_{inp} \cup \Sigma_{out}$*, and*

(3) the locally-controlled alphabet *of $\mathcal{C}$ is denoted by Σ_{loc} and is defined as* $\Sigma_{loc} = \Sigma_{out} \cup \Sigma_{int}$. □

The elements of the full alphabet of a component automaton $\mathcal{C}$ are called the *actions* of $\mathcal{C}$. The elements of the external and locally-controlled alphabets are called the *external* and *locally-controlled actions* of $\mathcal{C}$, respectively.

For a given component automaton $\mathcal{C}$, its set of (*finite* and *infinite*) *computations* and — given a set of actions Θ — its *Θ-records* and its *Θ-behavior* are carried over from Definitions 3.1.2 and 3.1.7 through its underlying automaton und($\mathcal{C}$). This means that we have, e.g., $\mathbf{C}_{\mathcal{C}} = \mathbf{C}_{\mathrm{und}(\mathcal{C})}$ and $\mathbf{B}^{\Theta}_{\mathcal{C}} = \mathbf{B}^{\Theta}_{\mathrm{und}(\mathcal{C})}$.

The different roles actions can play within a component automaton naturally give rise to various behavioral language definitions. Given a component automaton $\mathcal{C}$, we can distinguish specific *records* and *behavior* of $\mathcal{C}$ by selecting an appropriate subset of Σ.

If $\Theta = \Sigma_{inp}$, then we refer to the Θ-records of $\mathcal{C}$ as the *input records* and to $\mathbf{B}^{\Theta,\infty}_{\mathcal{C}}$ as the *input behavior* of $\mathcal{C}$. Analogously, by setting $\Theta = \Sigma_{out}$, we obtain the *output records* and the *output behavior* of $\mathcal{C}$; with $\Theta = \Sigma_{int}$ we deal with *internal records* and the *internal behavior* of $\mathcal{C}$; in case $\Theta = \Sigma_{ext}$ we have *external records* and the *external behavior* of $\mathcal{C}$; finally, when $\Theta = \Sigma_{loc}$ we have *locally-controlled records* and the *locally-controlled behavior* of $\mathcal{C}$. Needless to say, also *finitary* and *infinitary* (Θ-)*behavior* can be distinguished in this way.

Example 5.1.3. Let $\mathcal{C} = (\{e, f\}, (\{\$\}, \{c\}, \varnothing), \{(e, \$, f), (f, c, e)\}, \{e\})$ be a component automaton modeling a very simple **c**offee vending machine. It is depicted in Figure 5.1.

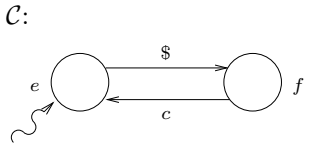

Fig. 5.1. Component automaton $\mathcal{C}$.

State e indicates that the coin slot of the vending machine is **e**mpty, while state f indicates that it is **f**illed. The result of inserting a dollar is modeled by the action \$ and fills the coin slot. The vending machine obviously is not in charge of determining the moment a dollar is inserted and \$ is thus defined to be an input action. The automaton does decide when to output coffee and this should moreover be observable by the environment. Hence the result of

outputting a **c**offee is modeled by the output action c. After the vending machine has produced the coffee it is ready for another request for coffee. Initially, the vending machine is waiting for the insertion of a dollar into its empty coin slot. Hence the vending machine's initial state is e.

The behavior of the vending machine is alternatingly accepting a dollar and producing a coffee. It can do so ad infinitum. □

Before we turn to the definition of a team automaton formed from a set of component automata we fix some notation.

Notation 4. *In the rest of this chapter we assume a fixed, but arbitrary and possibly infinite index set $\mathcal{I} \subseteq \mathbb{N}$, which we will now use to index the component automata involved. For each $i \in \mathcal{I}$, we let $\mathcal{C}_i = (Q_i, (\Sigma_{i,inp}, \Sigma_{i,out}, \Sigma_{i,int}), \delta_i, I_i)$ be a fixed component automaton and we use Σ_i to denote its set of actions $\Sigma_{i,inp} \cup \Sigma_{i,out} \cup \Sigma_{i,int}$. Moreover, we let $\mathcal{S} = \{\mathcal{C}_i \mid i \in \mathcal{I}\}$ be a fixed set of component automata. Recall that $\mathcal{I} \subseteq \mathbb{N}$ implies that $\mathcal{I}$ is ordered by the usual $\leq$ relation on $\mathbb{N}$, thus inducing an ordering on $\mathcal{S}$. Note that the $\mathcal{C}_i$ are not necessarily different.* □

5.1.2 Team Automata

When composing a team automaton over $\mathcal{S}$, we require that the internal actions of the component automata involved are private, i.e. uniquely associated to one component automaton. This is formally expressed as follows.

Definition 5.1.4. *$\mathcal{S}$ is a* composable system *if for all $i \in \mathcal{I}$,*

$$\Sigma_{i,int} \cap \bigcup_{j \in \mathcal{I} \setminus \{i\}} \Sigma_j = \varnothing.$$ □

Note that every subset of a composable system is again a composable system.

Example 5.1.5. (Example 5.1.3 continued) Let $\mathcal{A} = (\{s,t\}, (\{c\}, \{\$\}, \varnothing), \{(s,\$,t), (t,c,s)\}, \{s\})$ be a component automaton modeling a coffee **a**ddict. It is depicted in Figure 5.2.

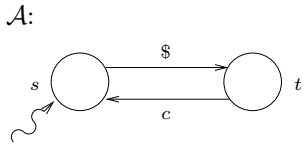

Fig. 5.2. Component automaton $\mathcal{A}$.

State s indicates that our coffee addict is (temporarily) **s**atisfied, while state t indicates that our coffee addict is **t**hirsty (again). The result of our coffee addict inserting a dollar (into a coffee vending machine) is modeled by the action \$ and shows our coffee addict's thirst. Our coffee addict obviously is in charge of determining when to show his or her thirst and thus when to insert a dollar. Since this should also be observable by the coffee vending machine we define \$ to be an output action. Our coffee addict however cannot decide when the coffee vending machine produces the much-awaited coffee. The result of our coffee addict trenching his or her thirst and becoming satisfied is thus modeled by the input action c. Initially our coffee addict is satisfied, modeled by our coffee addict's initial state s.

The behavior of our coffee addict is alternatingly inserting a dollar and trenching his or her thirst with a delicious cup of coffee. Like a true addict, our coffee addict can do so ad infinitum.

Since neither $\mathcal{C}$ nor $\mathcal{A}$ has any internal actions, $\mathcal{C}$ and $\mathcal{A}$ trivially form a composable system $\{\mathcal{C}, \mathcal{A}\}$. □

We are now ready to define a team automaton over a composable system $\mathcal{S}$ as a synchronized automaton over $\mathcal{S}$, except that in our definition of a team automaton we need to specify how to deal with the distinction of the alphabet into input, output, and internal actions.

The alphabet of actions of any team automaton $\mathcal{T}$ formed from $\mathcal{S}$ is uniquely determined by the alphabets of actions of the component automata constituting $\mathcal{S}$. The internal actions of the component automata will be the internal actions of $\mathcal{T}$. Each action which is output for one or more of the component automata is an output action of $\mathcal{T}$. Hence an action that is an output action of one component automaton and also an input action of another component automaton, is considered an output action of the team automaton. The input actions of the component automata that do not occur at all as an output action of any of the component automata, are the input actions of the team automaton. The reason for this construction of alphabets is again based on the intuitive idea of [Ell97] that when relating an input action a of a component automaton to an output action a of another component, then the input may be thought of as being caused by the output. On the other hand, output actions remain observable as output to other component automata.

Finally, the freedom of choosing a particular transition relation for a synchronized automaton over $\mathcal{S}$ is reduced slightly in the definition of a team automaton over $\mathcal{S}$, viz. for an internal action each component automaton always retains all its possibilities to execute that action and change state. Since $\mathcal{S}$ is a composable system, all internal actions are moreover uniquely

associated to one component automaton, which implies that synchronizations on internal actions thus never involve more than one component automaton.

Definition 5.1.6. *Let $\mathcal{S}$ be a composable system. Then a* team automaton over $\mathcal{S}$ *is a construct $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$, where*

$(Q, \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}, \delta, I)$ *is a synchronized automaton over $\mathcal{S}$ such that*

$$\delta_a = \Delta_a(\mathcal{S}), \text{ for all } a \in \Sigma_{int},$$

$\Sigma_{inp} = (\bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}) \setminus \bigcup_{i \in \mathcal{I}} \Sigma_{i,out}$,
$\Sigma_{out} = \bigcup_{i \in \mathcal{I}} \Sigma_{i,out}$, *and*
$\Sigma_{int} = \bigcup_{i \in \mathcal{I}} \Sigma_{i,int}$. □

The synchronized automaton $(Q, \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}, \delta, I)$ of a team automaton $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ is called the *underlying synchronized automaton* of $\mathcal{T}$ and it is denoted by $\mathrm{und}(\mathcal{T})$.

All team automata over a given composable system have the same set of states, the same alphabet of actions — including the distribution over input, output, and internal actions — and the same set of initial states. They only differ by the choice of the transition relation, and in fact only as far as external actions are concerned: for each external action a we have the freedom to choose a δ_a. This implies that $\mathcal{S}$, even if it is a composable system, does not uniquely define a team automaton.

Example 5.1.7. (Example 5.1.5 continued) We now show how our coffee addict can obtain a coffee from our vending machine by forming a team automaton $\mathcal{T}$ over the composable system $\{\mathcal{C}, \mathcal{A}\}$. This team automaton should model a form of collaboration between our coffee addict and the vending machine. This is implemented by synchronizations of certain actions. We require the output action \$ of our coffee addict to be synchronized with the input action \$ of our vending machine. The occurrence of this action in the team automaton then reflects the simultaneous execution of \$ by our coffee addict and our vending machine. Likewise action c is simultaneously executed by our coffee addict and our vending machine. This defines the transition relation of $\mathcal{T}$. Note that only the transition relation of $\mathcal{T}$ had to be chosen, the other elements of $\mathcal{T}$ follow directly from Definition 5.1.6. Note in particular that both \$ and c are output actions of $\mathcal{T}$. Hence $\mathcal{T}$ is formally defined as $\mathcal{T} = (\{(e,s),(e,t),(f,s),(f,t)\}, (\varnothing, \{\$, c\}, \varnothing), \delta, \{(e,s)\})$, where $\delta = \{((e,s), \$, (f,t)), ((f,t), c, (e,s))\}$. It is depicted in Figure 5.3. □

Consistency in the sense that in a team automaton every action appears exclusively as an input, output, or internal action, is guaranteed by Definition 5.1.6 (which ensures that input and output actions remain distinct) and

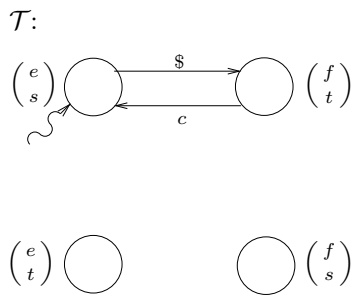

Fig. 5.3. Team automaton $\mathcal{T}$ over $\{\mathcal{C}, \mathcal{A}\}$.

the fact that a team automaton is constructed over a composable system. Together with Definition 5.1.4 this implies that every team automaton is again a component automaton, which in its turn could be used as a component automaton in a new team automaton.

Theorem 5.1.8. *Every team automaton is a component automaton.* □

As was the case for synchronized automata (cf. Section 4.1) we note that even though a team automaton over a composable system consisting of just one component automaton $\{\mathcal{C}_i\}$ is again a component automaton, such a team automaton is different from its only constituting component automaton.

All observations on (component) automata hold for team automata as well. The abbreviations for sets of alphabets carry over to team automata in the obvious way. Finally, note that whenever the distinction of the alphabet of actions into input, output, and internal actions is irrelevant, then a synchronized automaton can be seen as a team automaton. As a matter of fact, in examples in the remainder of this chapter we will often refer to synchronized automata defined in earlier chapters as team automata.

5.1.3 Subteams

Similar to the way we extracted subautomata from synchronized automata, by focusing on a subset of the composable system $\mathcal{S}$ of component automata constituting a team automaton $\mathcal{T}$ we now distinguish *subteams* within $\mathcal{T}$. As before, the transitions of a subteam are restrictions of the transitions of $\mathcal{T}$ to the component automata in the subteam, while its actions are the actions of the component automata involved. However, the actions of both component automata and team automata are distributed over three distinct alphabets. Since we want to be able to deal with a subteam as an independent team automaton over a subset of $\mathcal{S}$, we need to classify its actions without

the context provided by $\mathcal{T}$. Hence, whether an action is input, output, or internal for the subteam only depends on its role in the component automata forming the subteam rather than on how it is classified in $\mathcal{T}$. This means in particular that an action which is an output action of $\mathcal{T}$ is an input action for the subteam, whenever this action is an input action of at least one of the component automata of the subteam and no component automata of the subteam have this action as an output action.

Definition 5.1.9. *Let $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ be a team automaton over the composable system $\mathcal{S}$ and let $J \subseteq \mathcal{I}$. Then the* subteam of $\mathcal{T}$ determined by J *is denoted by $SUB_J(\mathcal{T})$ and is defined as $SUB_J(\mathcal{T}) = (Q_J, (\Sigma_{J,inp}, \Sigma_{J,out}, \Sigma_{J,int}), \delta_J, I_J)$, where*

$(Q_J, \Sigma_{J,inp} \cup \Sigma_{J,out} \cup \Sigma_{J,int}, \delta_J, I_J)$ *is the subautomaton $SUB_J(und(\mathcal{T}))$,*
$\Sigma_{J,inp} = (\bigcup_{j \in J} \Sigma_{j,inp}) \setminus \bigcup_{j \in J} \Sigma_{j,out}$,
$\Sigma_{J,out} = \bigcup_{j \in J} \Sigma_{j,out}$, *and*
$\Sigma_{J,int} = \bigcup_{j \in J} \Sigma_{j,int}$. □

As before, we write SUB_J instead of $SUB_J(\mathcal{T})$ whenever $\mathcal{T}$ is clear from the context. Note that the notation SUB_J is used both for the subautomaton of a synchronized automaton and for the subteam of a team automaton. In cases where this might lead to confusion, we will always state explicitly the type of automaton we deal with.

It is not hard to see that any subteam satisfies the requirements of a team automaton.

Theorem 5.1.10. *Let $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ be a team automaton over the composable system $\mathcal{S}$ and let $J \subseteq \mathcal{I}$. Then*

SUB_J is a team automaton over $\{\mathcal{C}_j \mid j \in J\}$.

Proof. We already noted that every subset of a composable system is again a composable system. Since the alphabets of SUB_J as given in Definition 5.1.9 moreover satisfy the requirements of Definition 5.1.6 for team automata over $\{\mathcal{C}_j \mid j \in J\}$, it directly follows from Theorem 4.1.8 that SUB_J is a team automaton over $\{\mathcal{C}_j \mid j \in J\}$. □

Similar to our conclusion — in Subsection 4.1.2 — that a subautomaton of a synchronized automaton is again a synchronized automaton, and thus also an automaton, we now conclude from Theorem 5.1.10 that a subteam of a team automaton is again a team automaton and thus, by Theorem 5.1.8, also a component automaton. Based on the results from Section 4.3 we will consider the dual approach and use team automata as component automata in "larger" team automata in the next section.

5.2 Iterated Composition

This section continues our investigation of Section 4.3, the difference being that instead of synchronized automata we now consider team automata. This means that we have to take into account that team automata can only be formed over composable systems and, moreover, that we deal with three mutually disjoint alphabets constituting the alphabet of a team automaton.

Notation 5. *In the rest of this chapter we let $\mathcal{S}$ be a composable system.* □

We consider the issue of iteratively composing team automata, given a composable system of team automata. First we prove that composability is preserved in the process of iteration.

Theorem 5.2.1. *Let $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$, where $\mathcal{J} \subseteq \mathbb{N}$, form a partition of $\mathcal{I}$. Let, for each $j \in \mathcal{J}$, $\mathcal{T}_j$ be a team automaton over $\mathcal{S}_j = \{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$. Then*

$\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ is a composable system.

Proof. Denote for each $\mathcal{T}_j$, $j \in \mathcal{J}$, by Γ_j its set of actions and by $\Gamma_{j,int}$ its internal alphabet. By Definition 5.1.6 we have $\Gamma_{j,int} = \bigcup_{i \in \mathcal{I}_j} \Sigma_{i,int}$ and $\Gamma_j = \bigcup_{i \in \mathcal{I}_j} \Sigma_i$, for all $j \in \mathcal{J}$. By the composability of $\mathcal{S}$ we have $\Sigma_{i,int} \cap \bigcup_{\ell \in \mathcal{I} \setminus \{i\}} \Sigma_\ell = \varnothing$, for all $i \in \mathcal{I}$. Since the $\mathcal{I}_j$ are mutually disjoint it now follows immediately that for all $j \in \mathcal{J}$, $\Gamma_{j,int} \cap \bigcup_{\ell \in \mathcal{J} \setminus \{j\}} \Gamma_\ell = \varnothing$. Hence $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ is a composable system. □

Given a composable system one may thus form team automata over disjoint subsets of the composable system. These team automata together with the component automata not involved in any of these team automata form — by Theorem 5.2.1 — again a composable system, which can subsequently be used as the basis for the formation of still higher-level team automata. Completely analogous to Definition 4.3.8 we now define iterated team automata as a generalization of team automata.

Definition 5.2.2. *$\mathcal{T}$ is an* iterated team automaton over *$\mathcal{S}$ if either*

(1) $\mathcal{T}$ is a team automaton over $\mathcal{S}$, or

(2) $\mathcal{T}$ is a team automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, where each $\mathcal{T}_j$ is an iterated team automaton over $\{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$, for some $\mathcal{I}_j \subseteq \mathcal{I}$, and $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forms a partition of $\mathcal{I}$. □

As was the case for iterated synchronized automata, we see that an iterated team automaton is thus a generalization of a team automaton: every team automaton over a given composable system may also be viewed as an iterated team automaton over that composable system. Conversely, as before, team automata formed iteratively over a composable system are essentially team automata over that composable system. Once again, the only difference is the ordering and grouping of the elements from the composable system. Heavily based on the results from Section 4.3, we now formalize this statement.

By Lemma 4.3.9, the set of (initial) states of an iterated team automaton over $\mathcal{S}$ is — after reordering — the same as the set of (initial) states of any team automaton over $\mathcal{S}$. According to Lemma 4.3.10 also its actions are the same as the actions of any team automaton formed over $\mathcal{S}$. However, the basic difference between team automata and synchronized automata is the distinction of actions into three mutually disjoint alphabets. The following lemma shows that this property is not destroyed by iteration.

Lemma 5.2.3. *Let $\mathcal{T} = (P, (\Gamma_{inp}, \Gamma_{out}, \Gamma_{int}), \gamma, J)$ be an iterated team automaton over $\mathcal{S}$. Then*

(1) $\Gamma_{inp} = (\bigcup_{i\in\mathcal{I}} \Sigma_{i,inp}) \setminus \bigcup_{i\in\mathcal{I}} \Sigma_{i,out}$,

(2) $\Gamma_{out} = \bigcup_{i\in\mathcal{I}} \Sigma_{i,out}$, *and*

(3) $\Gamma_{int} = \bigcup_{i\in\mathcal{I}} \Sigma_{i,int}$.

Proof. If $\mathcal{T}$ is a team automaton over $\mathcal{S}$, then the statement follows immediately from Definition 5.1.6. Now assume that $\mathcal{T}$ is a team automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, where $\mathcal{J} \subseteq \mathbb{N}$, and each $\mathcal{T}_j = (P_j, (\Gamma_{j,inp}, \Gamma_{j,out}, \Gamma_{j,int}), \gamma_j, J_j)$ is an iterated team automaton over $\{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$, with $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forming a partition of $\mathcal{I}$. Assume furthermore inductively that for all $j \in \mathcal{J}$, $\Gamma_{j,inp} = (\bigcup_{i\in\mathcal{I}_j} \Sigma_{i,inp}) \setminus \bigcup_{i\in\mathcal{I}_j} \Sigma_{i,out}$, $\Gamma_{j,out} = \bigcup_{i\in\mathcal{I}_j} \Sigma_{i,out}$, and $\Gamma_{j,int} = \bigcup_{i\in\mathcal{I}_j} \Sigma_{i,int}$.

Then $\Gamma_{int} = \bigcup_{j\in\mathcal{J}} \Gamma_{j,int} = \bigcup_{j\in\mathcal{J}} \bigcup_{i\in\mathcal{I}_j} \Sigma_{i,int} = \bigcup_{i\in\mathcal{I}} \Sigma_{i,int}$, by Definition 5.1.6, and because $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forms a partition of $\mathcal{I}$.
Similarly, $\Gamma_{out} = \bigcup_{i\in\mathcal{I}} \Sigma_{i,out}$.
Finally, $\Gamma_{inp} = (\bigcup_{j\in\mathcal{J}} \Gamma_{j,inp}) \setminus \Gamma_{out}$ by Definition 5.1.6. Hence $\Gamma_{inp} = (\bigcup_{j\in\mathcal{J}}((\bigcup_{i\in\mathcal{I}_j} \Sigma_{i,inp}) \setminus \bigcup_{i\in\mathcal{I}_j} \Sigma_{i,out})) \setminus \Gamma_{out} = (\bigcup_{i\in\mathcal{I}} \Sigma_{i,inp}) \setminus \Gamma_{out}$ because $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forms a partition of $\mathcal{I}$. □

Hence the set of actions — including their distribution over input, output, and internal actions — of every iterated team automaton over $\mathcal{S}$ is the same as that of any team automaton over $\mathcal{S}$. Finally, from Lemma 4.3.10 we moreover know that the transitions of any team automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ are — after reordering — the transitions of a team automaton over $\mathcal{S}$. Iteration in

the construction of a team automaton thus does not lead to an increase of the possibilities for synchronization. In other words, we can conclude that every iterated team automaton over a composable system can be interpreted as a team automaton over that composable system by reordering its state space and its transition space.

Definition 5.2.4. *Let $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ be an iterated team automaton over $\mathcal{S}$. Then the* reordered version of $\mathcal{T}$ w.r.t. $\mathcal{S}$ *is denoted by $\langle\langle \mathcal{T} \rangle\rangle_{\mathcal{S}}$ and is defined as*

$$\langle\langle \mathcal{T} \rangle\rangle_{\mathcal{S}} = (\{\langle q \rangle_Q \mid q \in Q\}, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \\ \{(\langle q \rangle_Q, a, \langle q' \rangle_Q) \mid q, q' \in Q,\ (q, a, q') \in \delta\}, \{\langle q \rangle_I \mid q \in I\}). \ \square$$

Note that the notation $\langle\langle \mathcal{T} \rangle\rangle_{\mathcal{S}}$ is used both for the reordered version of a synchronized automaton and for the reordered version of a team automaton. In cases where this might lead to confusion, we will always state explicitly the type of automaton we deal with.

From Lemmata 4.3.9, 4.3.10, and 5.2.3 we conclude that $\langle\langle \mathcal{T} \rangle\rangle_{\mathcal{S}}$ indeed is a team automaton over $\mathcal{S}$ whenever $\mathcal{T}$ is an iterated team automaton over $\mathcal{S}$. In fact, $\langle\langle \mathcal{T} \rangle\rangle_{\mathcal{S}}$ is the interpretation of $\mathcal{T}$ as a team automaton over $\mathcal{S}$ by reordering. We thus obtain the following direct consequences of Theorems 4.3.12 and 4.3.13.

Theorem 5.2.5. *Let $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ be an iterated team automaton over $\mathcal{S}$ and let Θ be an alphabet disjoint from Q. Then*

(1) $\mathbf{C}^{\infty}_{\langle\langle \mathcal{T} \rangle\rangle_{\mathcal{S}}} = \{\langle q_0 \rangle_Q a_1 \langle q_1 \rangle_Q a_2 \langle q_2 \rangle_Q \cdots \mid q_0 a_1 q_1 a_2 q_2 \cdots \in \mathbf{C}^{\infty}_{\mathcal{T}}\}$ *and*

(2) $\mathbf{B}^{\Theta,\infty}_{\langle\langle \mathcal{T} \rangle\rangle_{\mathcal{S}}} = \mathbf{B}^{\Theta,\infty}_{\mathcal{T}}$. $\square$

Theorem 5.2.6. *Let $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ be a team automaton over $\mathcal{S}$ and let $\{\mathcal{I}_j \mid j \in \mathcal{J}\}$, where $\mathcal{J} \subseteq \mathbb{N}$, form a partition of $\mathcal{I}$. Let, for each $j \in \mathcal{J}$, $\mathcal{T}_j = (P_j, (\Gamma_{j,inp}, \Gamma_{j,out}, \Gamma_{j,int}), \gamma_j, J_j)$ be an iterated team over $\{\mathcal{C}_i \mid i \in \mathcal{I}_j\}$. Then*

(1) if $(\delta_{\mathcal{I}_j})_a \subseteq \{(\langle q \rangle_{P_j}, \langle q' \rangle_{P_j}) \mid (q, q') \in \gamma_{j,a}\}$, for all $a \in \Gamma_{j,inp} \cup \Gamma_{j,out} \cup \Gamma_{j,int}$ for all $j \in \mathcal{J}$, then there exists a team automaton $\widehat{\mathcal{T}}$ over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$ such that $\langle\langle \widehat{\mathcal{T}} \rangle\rangle_{\mathcal{S}} = \mathcal{T}$, and

(2) if $\widehat{\mathcal{T}}$ is a team automaton over $\{\mathcal{T}_j \mid j \in \mathcal{J}\}$, then $\langle\langle \widehat{\mathcal{T}} \rangle\rangle_{\mathcal{S}} = \mathcal{T}$ implies that $(\delta_{\mathcal{I}_j})_a \setminus \{(p, p) \mid (p, p) \in \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_j\})\} \subseteq \{(\langle q \rangle_{P_j}, \langle q' \rangle_{P_j}) \mid (q, q') \in \gamma_{j,a}\}$, for all $a \in \Gamma_{j,inp} \cup \Gamma_{j,out} \cup \Gamma_{j,int}$ for all $j \in \mathcal{J}$. $\square$

Similar to the conclusion we reached for synchronized automata in Section 4.3 we now see that not only every iterated team automaton over $\mathcal{S}$ can be considered as a team automaton directly constructed from $\mathcal{S}$ by Definition 5.2.4, but according to Theorem 5.2.6 also every team automaton can be iteratively constructed from its subteams. Consequently, both subteams and iterated team automata can be treated as team automata — including the considerations concerning their computations and their behavior — and it thus suffices to study only the relationship between subteams and team automata in the sequel, i.e. without considering iterated team automata explicitly.

5.3 Synchronizations

In Section 4.4 we introduced three natural types of synchronization. These types of synchronization can be studied in the context of team automata as well. However, they obviously ignore whether actions are input, output, or internal to certain component automata. For internal actions which belong to only one component automaton, distinguishing between their roles in different component automata is indeed not very relevant. External actions, however, may be input to some component automata, and output to other component automata. In this section we thus investigate types of synchronizations relating to the different roles that an action may have in different component automata.

Notation 6. *For the remainder of this chapter we let* $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ *be a fixed team automaton over* $\mathcal{S}$*. Note that* Σ_{inp}*,* Σ_{out}*, and* Σ_{int} *are the input, output, and internal alphabet, respectively, of any team automaton over* $\mathcal{S}$ *(i.e. not only of* $\mathcal{T}$*). Furthermore, we use* Σ *to denote the set of actions* $\Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$*, we use* Σ_{ext} *to denote the set of external actions* $\Sigma_{inp} \cup \Sigma_{out}$*, and we use* Σ_{loc} *to denote the set of locally-controlled actions* $\Sigma_{out} \cup \Sigma_{int}$ *of any team automaton over* $\mathcal{S}$ *(i.e. including* $\mathcal{T}$*).* □

First we separate the output role of external actions from their input role. Given an external action, we locate its input and output domain within $\mathcal{I}$, and then use these domains to define input subteams and output subteams. Finally, we define two specific types of synchronization relating such input subteams and output subteams of team automata.

Definition 5.3.1. *Let* $a \in \Sigma_{ext}$*. Then*

(1) $\mathcal{I}_{a,inp}(\mathcal{S}) = \{j \in \mathcal{I} \mid a \in \Sigma_{j,inp}\}$ *is the* input domain *of* a *in* $\mathcal{S}$ *and*

(2) $\mathcal{I}_{a,out}(\mathcal{S}) = \{j \in \mathcal{I} \mid a \in \Sigma_{j,out}\}$ *is the* output domain *of* a *in* $\mathcal{S}$*.* □

No external action of any team automaton $\mathcal{T}$ will ever be both an input and an output action for one component automaton. Thus, for each $j \in \mathcal{I}$, $\Sigma_{j,inp} \cap \Sigma_{j,out} = \varnothing$, and consequently $\mathcal{I}_{a,inp}(\mathcal{S}) \cap \mathcal{I}_{a,out}(\mathcal{S}) = \varnothing$, for all $a \in \Sigma_{ext}$.

Note that, by Definition 5.1.6, $a \in \Sigma_{out}$ if and only if $\mathcal{I}_{a,out}(\mathcal{S}) \neq \varnothing$, while $a \in \Sigma_{inp}$ if and only if $\mathcal{I}_{a,inp}(\mathcal{S}) \neq \varnothing$ and $\mathcal{I}_{a,out}(\mathcal{S}) = \varnothing$.

In the following example we show how to to determine the input and output domains of actions in a composable system.

Example 5.3.2. (Example 4.1.5 continued) We turn the automata W_i, with $i \in [4]$, into component automata by distributing their alphabet $\{a, b\}$ over input, output, and internal alphabets. We let a and b be output actions in both W_1 and W_2 and we let them be input actions in both W_3 and W_4. Since $\{W_1, W_2\}$ is now a composable system, the synchronized automaton $\mathcal{T}_{\{1,2\}}$ (over $\{W_1, W_2\}$) is now a team automaton. Likewise $\{\mathcal{T}_{\{1,2\}}, W_3, W_4\}$ is now a composable system and the synchronized automaton $\mathcal{T}$ (over $\{\mathcal{T}_{\{1,2\}}, W_3, W_4\}$) is now a team automaton. Both these team automata have an empty input alphabet, output alphabet $\{a, b\}$, and an empty internal alphabet.

Let $\mathcal{T}_1 = \mathcal{T}_{\{1,2\}}$, $\mathcal{T}_2 = W_3$, and $\mathcal{T}_3 = W_4$. Then $\mathcal{T}$ is a team automaton over $\mathcal{S} = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3\}$. Actions a and b are output actions in $\mathcal{T}_1$, whereas they are input actions in both $\mathcal{T}_2$ and $\mathcal{T}_3$. Hence $\mathcal{I}_{a,out}(\mathcal{S}) = \{1\}$ and $\mathcal{I}_{a,inp}(\mathcal{S}) = \{2, 3\}$. □

Note that the input domain and the output domain of an external action of a team automaton may be empty. For every external action, however, at least one of these domains is nonempty. In case the input (output) domain is empty, then the input (output) subteam is the trivial component automaton.

Example 5.3.3. In Figure 5.4 the structure of a team automaton $\mathcal{T}$ with respect to one of its external actions a is depicted. Indicated are its input subteam $SUB_{a,inp}$ and its output subteam $SUB_{a,out}$. The square boxes in this figure denote component automata. Clearly, $\mathcal{T}$ may also contain component automata that do not have a as an external action. □

Notation 7. *For the remainder of this chapter we make no more explicit references to the fixed composable system $\mathcal{S}$ when denoting the input and output domain of an action a in $\mathcal{S}$, i.e. we write $\mathcal{I}_{a,inp}$ and $\mathcal{I}_{a,out}$ rather than $\mathcal{I}_{a,inp}(\mathcal{S})$ and $\mathcal{I}_{a,out}(\mathcal{S})$, respectively. Furthermore, for all $a \in \Sigma_{ext}$, we use $SUB_{a,inp}(\mathcal{T})$ to denote $SUB_{\mathcal{I}_{a,inp}}(\mathcal{T})$, the* input subteam *of a in $\mathcal{T}$, and we use $SUB_{a,out}(\mathcal{T})$ to denote $SUB_{\mathcal{I}_{a,out}}(\mathcal{T})$, the* output subteam *of a in $\mathcal{T}$. If no confusion arises we even omit the $\mathcal{T}$ and simply write $SUB_{a,inp}$ and $SUB_{a,out}$, respectively.* □

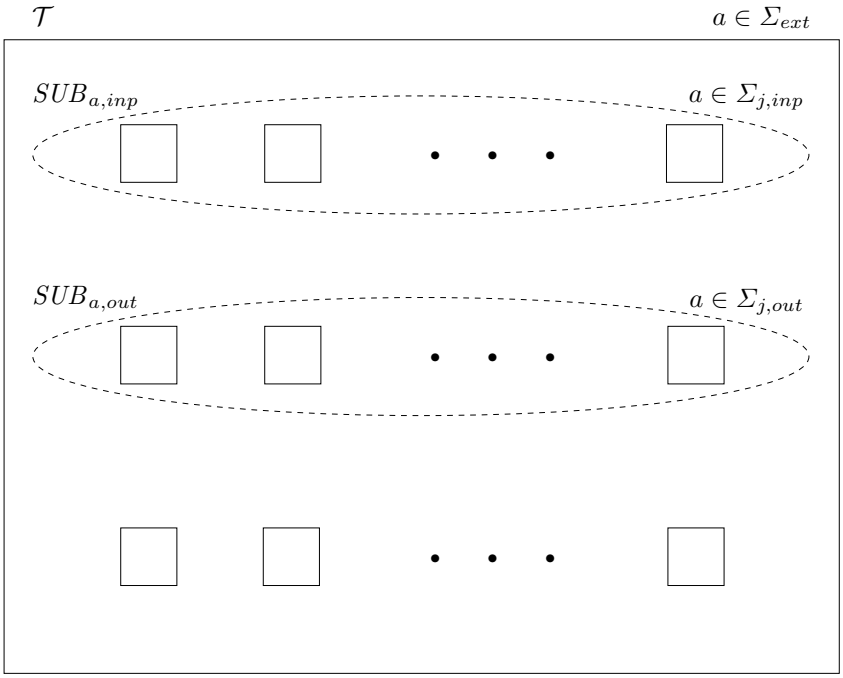

Fig. 5.4. A team automaton $\mathcal{T}$ with its subteams $SUB_{a,inp}$ and $SUB_{a,out}$.

5.3.1 Peer-to-Peer

Having determined for each external action a its input and its output subteam, we can now identify certain types of synchronization relating to a in its role as input or output. We begin by looking within these subteams, in which a by definition has only one role and all component automata are peers, in the sense that they are on an equal footing with respect to a. We say that an input (output) action a is *input (output) peer-to-peer* if every execution of a involving component automata of that subteam requires the participation of all.

This obligation to participate can be explained in a strong and in a weak sense. *Strong input (output) peer-to-peer* simply means that no synchronizations on a can take place unless all component automata in the input (output) domain of a take part. *Weak input (output) peer-to-peer* means that synchronizations on a involve all of the component automata in the input (output) domain of a which are ready to execute a — i.e. which are in a state in which a is enabled. Thus the notion of strong input (output) peer-to-peer requires

that a is *ai* in its input (output) subteam, while the notion of weak input (output) peer-to-peer requires that a is *si* in its input (output) subteam.

Definition 5.3.4. *(1) The set of* strong input peer-to-peer *(sipp for short) actions of $\mathcal{T}$ is denoted by $SIPP(\mathcal{T})$ and is defined as*

$$SIPP(\mathcal{T}) = \{a \in \Sigma_{ext} \mid a \in AI(SUB_{a,inp})\},$$

(2) the set of weak input peer-to-peer *(wipp for short) actions of $\mathcal{T}$ is denoted by $WIPP(\mathcal{T})$ and is defined as*

$$WIPP(\mathcal{T}) = \{a \in \Sigma_{ext} \mid a \in SI(SUB_{a,inp})\},$$

(3) the set of strong output peer-to-peer *(sopp for short) actions of $\mathcal{T}$ is denoted by $SOPP(\mathcal{T})$ and is defined as*

$$SOPP(\mathcal{T}) = \{a \in \Sigma_{ext} \mid a \in AI(SUB_{a,out})\}, \text{ and}$$

(4) the set of weak output peer-to-peer *(wopp for short) actions of $\mathcal{T}$ is denoted by $WOPP(\mathcal{T})$ and is defined as*

$$WOPP(\mathcal{T}) = \{a \in \Sigma_{ext} \mid a \in SI(SUB_{a,out})\}. \qquad \square$$

We should remark here that an external action a that does not occur as an input action in any of the component automata (implying that $\mathcal{I}_{a,inp} = \varnothing$ and that $SUB_{a,inp}$ is the trivial component automaton) can neither be *sipp* nor *wipp*. This is due to the fact that trivial component automata (as was the case for trivial automata) have no actions whatsoever, and thus neither *ai* nor *si* actions. Note that $a \in SIPP(\mathcal{T})$ or $a \in WIPP(\mathcal{T})$ does not imply that $a \in \Sigma_{inp}$. Similarly, if a is *sopp* or *wopp* in $\mathcal{T}$, then it must be the case that it occurs as an output action in at least one component automaton of $\mathcal{T}$ (implying that $a \in \Sigma_{out}$).

Note that an external action of a team automaton $\mathcal{T}$ over $\mathcal{S}$ can be both *sipp* and *sopp* in $\mathcal{T}$. In that case the external action is an input action of one component automaton of $\mathcal{S}$ and an output action of another component automaton of $\mathcal{S}$.

Example 5.3.5. (Example 5.3.3 continued) As depicted in Figures 5.5 and 5.6, strong and weak input (output) peer-to-peer synchronizations relate to synchronizations within the corresponding input (output) subteam. $\square$

Next we present a more concrete example of strong and weak input (output) synchronizations within team automata.

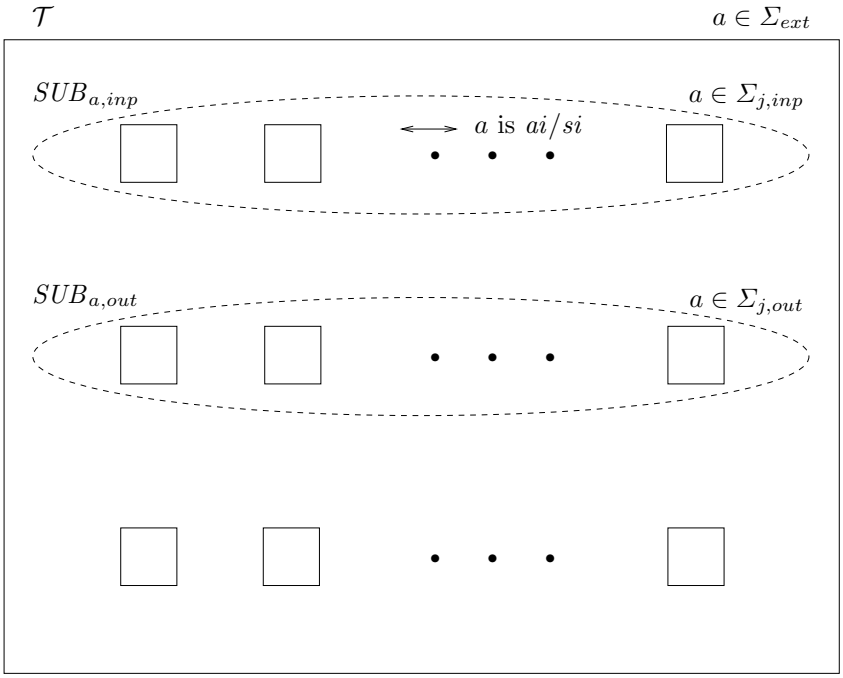

Fig. 5.5. A team automaton $\mathcal{T}$ with a *sipp*/*wipp* action a.

Example 5.3.6. (Example 5.3.2 continued) Actions a and b both are *sopp* as well as *wopp* in $\mathcal{T}$. This can be concluded from the fact that we already know from Example 4.4.4 that actions a and b both are ai in the output subteam $\mathcal{T}_1 = \mathcal{T}_{\{1,2\}}$ of $\mathcal{T}$. It is easy to verify that actions a and b both are also *sipp* as well as *wipp* in $\mathcal{T}$. □

5.3.2 Master-Slave

We now define synchronizations *between* the input and output subteams of an external action a. Here the idea is that input actions ("slaves") are driven by output actions ("masters"). This means that if a is an output action, then its input counterpart can never take place without being triggered (i.e. the slave never proceeds on its own). Consequently, the input subteam of an output action a cannot execute a unless a is also executed as an output action (by its output subteam). It is however possible that a is executed as an output action without its simultaneous execution as an input action. We say that a is *master-slave* if it is an output action and its output subteam participates in every a-transition of $\mathcal{T}$.

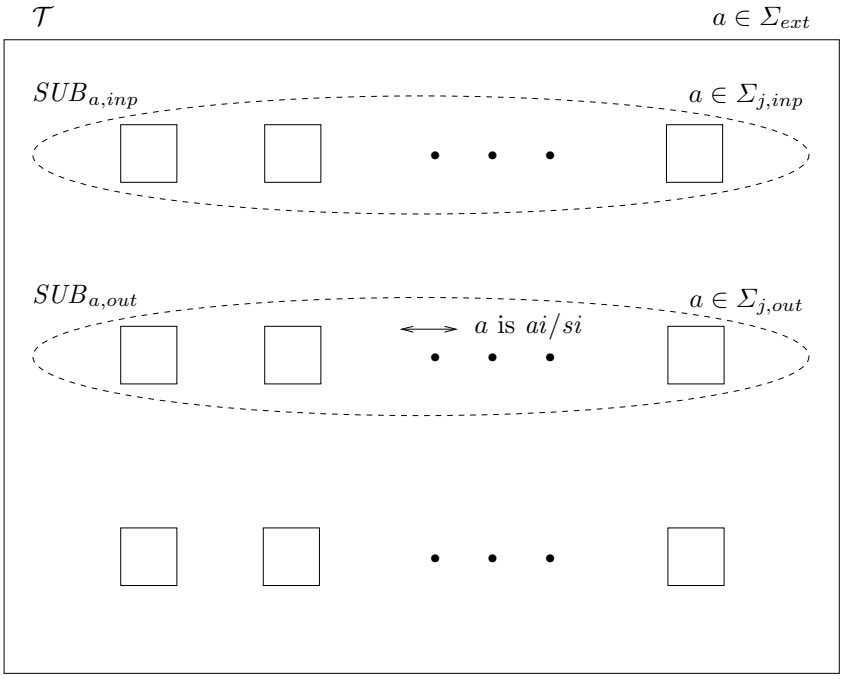

Fig. 5.6. A team automaton $\mathcal{T}$ with a *sopp*/*wopp* action a.

In addition one could require that a in its role of input action *has to* synchronize with a as an output action (i.e. the slave has to follow the master). Since the obligation of the slave to follow the master may again be formulated in two different ways, we obtain notions of *strong* and *weak master-slave* actions. When guided by the *ai* principle, we get a strong notion of master-slave synchronization, while the *si* principle leads to a weak notion of master-slave synchronization. We say that a is *strong master-slave* if it is master-slave and its input subteam moreover participates in every a-transition of $\mathcal{T}$. We say that a is *weak master-slave* if it is master-slave and its input subteam moreover participates in every a-transition of $\mathcal{T}$ *whenever it can*.

Definition 5.3.7. *Let $a \in \Sigma_{out}$, and let $J = \mathcal{I}_{a,out}$ and $K = \mathcal{I}_{a,inp}$. Then*

(1) the set of master-slave *(ms for short) actions of $\mathcal{T}$ is denoted by $MS(\mathcal{T})$ and is defined as*

$$MS(\mathcal{T}) = \{a \in \Sigma_{out} \mid proj_J{}^{[2]}(\delta_a) \subseteq (\delta_J)_a\},$$

(2) the set of strong master-slave *(sms for short) actions of $\mathcal{T}$ is denoted by $SMS(\mathcal{T})$ and is defined as*

$$SMS(\mathcal{T}) = \{a \in \Sigma_{out} \mid a \in MS(\mathcal{T}) \wedge ([K \neq \varnothing] \Rightarrow [proj_K{}^{[2]}(\delta_a) \subseteq (\delta_K)_a])\}, \text{ and}$$

(3) the set of weak master-slave *(wms for short) actions of $\mathcal{T}$ is denoted by $WMS(\mathcal{T})$ and is defined as*

$$WMS(\mathcal{T}) = \{a \in \Sigma_{out} \mid a \in MS(\mathcal{T}) \wedge ([K \neq \varnothing] \Rightarrow [((q,q') \in \delta_a \wedge a \ en_{SUB_K} \ proj_K(q)) \Rightarrow (proj_K{}^{[2]}(q,q') \in (\delta_K)_a)])\}. \ \square$$

For a to be *ms*, we require it to occur at least once as an output action ($\mathcal{I}_{a,out} \neq \varnothing$) — i.e. a can act as a master. Otherwise we could have slaves without a master. A master without slaves is allowed: $\mathcal{I}_{a,out} \neq \varnothing$ and $\mathcal{I}_{a,inp} = \varnothing$. In that case a trivially is *sms* and *wms*, since there are no slaves that do not follow the master.

Since the definition of a being *ms* in $\mathcal{T}$ guarantees that the output subteam of a is actively involved in every a-transition of $\mathcal{T}$, it follows immediately from Definition 4.1.6 that the a-transitions of the output subteam of a are precisely the projections of the a-transitions of $\mathcal{T}$ on the output domain of a. Similarly, in case a is *sms* we have in addition that the a-transitions of the input subteam of a are precisely the projections of the a-transitions of $\mathcal{T}$ on the input domain of a.

Theorem 5.3.8. *Let $J = \mathcal{I}_{a,out}$ and let $K = \mathcal{I}_{a,inp}$. Then*

(1) if $a \in MS(\mathcal{T})$, then $proj_J{}^{[2]}(\delta_a) = (\delta_J)_a$, and

(2) if $a \in SMS(\mathcal{T})$, then $proj_K{}^{[2]}(\delta_a) = (\delta_K)_a$.

Proof. (1) By Definition 4.1.6 we have $(\delta_J)_a = \mathrm{proj}_J{}^{[2]}(\delta_a) \cap \Delta_a(\{\mathcal{C}_j \mid j \in J\})$. Since $a \in MS(\mathcal{T})$ we have $\mathrm{proj}_J{}^{[2]}(\delta_a) \subseteq (\delta_J)_a$, for $J = \mathcal{I}_{a,out}$. Hence in this case $(\delta_J)_a = \mathrm{proj}_J{}^{[2]}(\delta_a)$.

(2) Analogous. Note that if $K = \varnothing$, then $\mathrm{proj}_K{}^{[2]}(\delta_a) = \varnothing = (\varnothing)_a$. $\square$

Note that if a is *wms*, then there may be a-transitions in $\mathcal{T}$ in which the input subteam — even when it is not trivial — is not actively involved. In those cases a is executed as an output action by $\mathcal{T}$ without the simultaneous execution of a as an input action.

Note that in Definition 5.3.7 input subteams and output subteams are treated as given entities (black boxes). Clearly, one can combine the master-slave types of synchronization with additional requirements on the synchronizations taking place within the subteams. One might, e.g., prescribe a master-slave type of synchronization on an action a that is in addition input peer-to-peer, in which case *all* component automata with a as an input action have to follow the output action. We will come back to this later.

Example 5.3.9. (Example 5.3.5 continued) If for an external action a of $\mathcal{T}$, $SUB_{a,out}$ is involved in all a-transitions of $\mathcal{T}$, then a is an *ms* action. If $SUB_{a,inp}$ moreover "has to" participate in every a-transition of $\mathcal{T}$, then a is an *sms* or *wms* action in $\mathcal{T}$. The idea of (strong or weak) types of master-slave synchronization between input and output subteams, is sketched in Figure 5.7. □

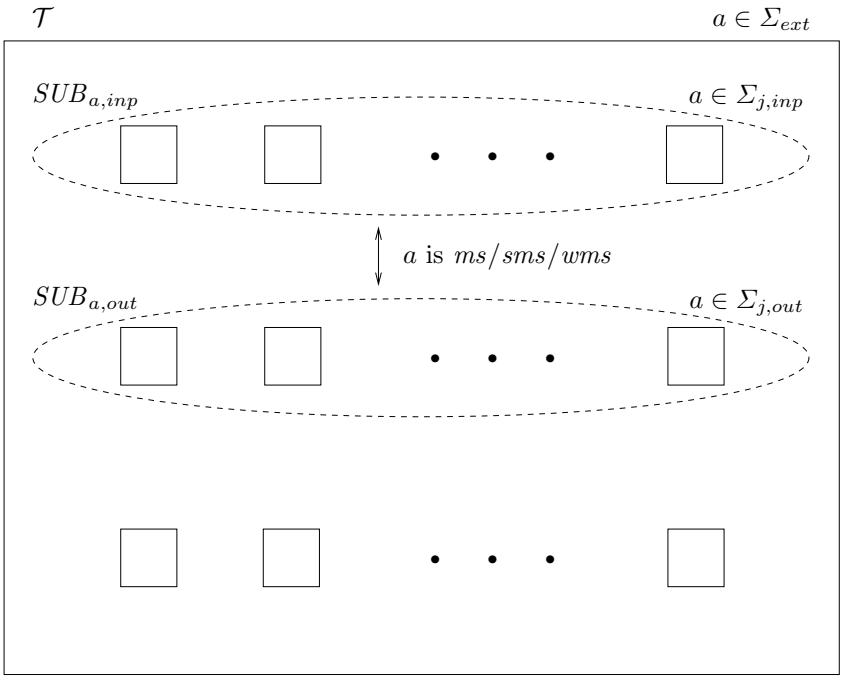

Fig. 5.7. A team automaton $\mathcal{T}$ with a $ms/sms/wms$ action a.

We thus note that whereas peer-to-peer types of synchronization are defined *within* subteams, master-slave types of synchronization are defined *between* input and output subteams.

Next we give a more elaborate example in which we apply the various types of synchronization introduced in this chapter so far to one of our running examples.

Example 5.3.10. (Example 5.3.6 continued) In this example we show that the car $\mathcal{T}$ is actually a two-wheel drive. Recall that we assume a maximal interpretation of the involvement of component automata in loops.

Actions a and b are both *sms* in $\mathcal{T}$. For a this can be concluded from the fact that $\text{proj}_{\{1\}}{}^{[2]}(\delta_a) = \{((s_1,s_2),(t_1,t_2)),((t_1,t_2),(t_1,t_2))\} = (\delta_{\{1\}})_a$ and $\text{proj}_{\{2,3\}}{}^{[2]}(\delta_a) = \{((s_3,s_4),(t_3,t_4)),((t_3,t_4),(t_3,t_4))\} = (\delta_{\{2,3\}})_a$, thus satisfying (1) and (2) of Definition 5.3.7. For b one can verify this in a similar fashion. We thus conclude that $\mathcal{T}$ models a two-wheel drive, in the sense that one axle (the input subteam of a and b) only turns and halts as a reaction to the other axle (the output subteam of a and b). Hence the former axle is the "slave" of the latter axle. □

5.3.3 A Case Study

In [Ell97] a simple example was presented to illustrate the concept of peer-to-peer and master-slave types of synchronization within team automata. In this subsection we give this example from [Ell97] a rigorous treatment in our formal team automata framework.

Example 5.3.11. Consider the three component automata depicted in Figure 5.8. They are formally defined by $\mathcal{C}_i = (Q_i, (\Sigma_{i,inp}, \Sigma_{i,out}, \Sigma_{i,int}), \delta_i, I_i)$, where for $i \in [3]$,

$Q_i = \{q_i, q_i'\}$,
$\Sigma_{1,inp} = \Sigma_{2,inp} = \Sigma_{3,out} = \varnothing$,
$\Sigma_{1,out} = \Sigma_{2,out} = \Sigma_{3,inp} = \{b\}$,
$\Sigma_{i,int} = \{a_i, a_i'\}$, with all a_i and a_i' distinct symbols different from b,
$\delta_{i,b} = \{(q_i, q_i')\}$,
$\delta_{j,a_j} = \{(q_j, q_j')\}$ and $\delta_{j,a_j'} = \{(q_j', q_j)\}$, for $j \in [2]$,
$\delta_{3,a_3} = \{(q_3, q_3)\}$ and $\delta_{3,a_3'} = \{(q_3', q_3')\}$, and
$I_i = \{q_i\}$.

Hence $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$ is a composable system.

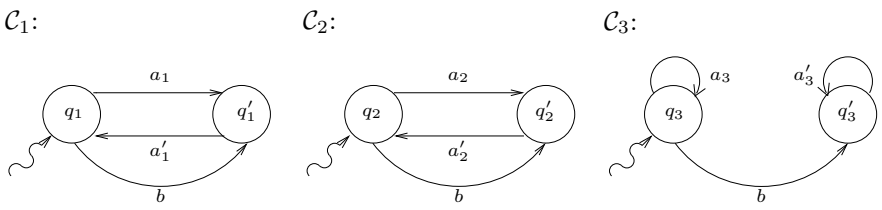

Fig. 5.8. Component automata $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$.

Two slightly different team automata $\mathcal{T}$ and $\mathcal{T}'$ over this composable system are defined next. All parameters of these team automata, except for

the set of labeled transitions, are predetermined by $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$. In fact, only the b-transitions can be varied as all the other actions are internal. The first team automaton ($\mathcal{T}$) is the one spelled out in [Ell97], whereas the second one ($\mathcal{T}'$) is the one discussed in the text in [Ell97].

Let $\mathcal{T} = (\prod_{i\in[3]} Q_i, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, \{(q_1, q_2, q_3)\})$ and let $\mathcal{T}' = (\prod_{i\in[3]} Q_i, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta', \{(q_1, q_2, q_3)\})$, where

$\Sigma_{inp} = \varnothing$,
$\Sigma_{out} = \{b\}$,
$\Sigma_{int} = \{a_1, a_1', a_2, a_2', a_3, a_3'\}$, and
δ and δ' are defined by
$\delta_a = \delta'_a = \Delta_a(\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\})$, for each $a \in \{a_1, a_1', a_2, a_2', a_3, a_3'\}$,
$\delta_b = \{((q_1, q_2, q_3), (q_1', q_2', q_3'))\}$, and
$\delta'_b = \{((q_1, q_2, q_3), (q_1', q_2', q_3')), ((q_1, q_2, q_3'), (q_1', q_2', q_3'))\}$.

Hence in $\mathcal{T}$ there is only one b-transition that can take place. It involves all three component automata and requires the j-th component to be in state q_j, for each $j \in [3]$. This transition is thus a simultaneous execution of b by all three component automata. In $\mathcal{T}'$, however, next to this b-transition just described, there is another b-transition that can take place and it involves only the first two component automata while the third component automaton is in state q_3' (in which b is not enabled). Hence this transition is a simultaneous execution of b by the first two component automata only. Both these team automata are depicted in Figure 5.9: $\mathcal{T}'$ contains all the depicted transitions, whereas $\mathcal{T}$ is obtained by ignoring the "dashed" transition $((q_1, q_2, q_3'), b, (q_1', q_2', q_3'))$.

It is easy to check that $\mathit{Free}(\mathcal{T}) = \mathit{Free}(\mathcal{T}') = AI(\mathcal{T}') = \Sigma_{int}$ and $AI(\mathcal{T}) = SI(\mathcal{T}) = SI(\mathcal{T}') = \Sigma$. Thus b is both *si* and *ai* in $\mathcal{T}$, while b is *si* but not *ai* in $\mathcal{T}'$. This is because $\mathcal{T}'$ has a b-transition in which $\mathcal{C}_3$ does not participate, even though $\mathcal{C}_3$ contains b in its (input) alphabet.

Note that in $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$ the input domain $\mathcal{I}_{b,inp}$ of b is $\{3\}$ and the output domain $\mathcal{I}_{b,out}$ of b is $\{1, 2\}$. The subteams of $\mathcal{T}$ and $\mathcal{T}'$ determined by $\{1, 2\}$ are the same: $SUB_{\{1,2\}}(\mathcal{T}) = SUB_{\{1,2\}}(\mathcal{T}')$. This is because $\mathrm{proj}_{\{1,2\}}{}^{[2]}(\delta_c) = \mathrm{proj}_{\{1,2\}}{}^{[2]}(\delta'_c)$, for each $c \in \{a_1, a_1', a_2, a_2', b\}$. Also $SUB_{\{3\}}(\mathcal{T}) = SUB_{\{3\}}(\mathcal{T}')$, since $\mathrm{proj}_{\{3\}}{}^{[2]}(\delta_c) = \mathrm{proj}_{\{3\}}{}^{[2]}(\delta'_c)$, for each $c \in \{a_3, a_3'\}$, and $\mathrm{proj}_{\{3\}}{}^{[2]}(\delta_b) \cap \Delta_b(\{\mathcal{C}_3\}) = \mathrm{proj}_{\{3\}}{}^{[2]}(\delta'_b) \cap \Delta_b(\{\mathcal{C}_3\}) = \{((q_3), (q_3'))\}$.

Since b is *ai* in $\mathcal{T}$, Lemma 4.7.1(2) implies that b is also *ai* in both $SUB_{\{1,2\}}(\mathcal{T}) = SUB_{\{1,2\}}(\mathcal{T}')$ and $SUB_{\{3\}}(\mathcal{T}) = SUB_{\{3\}}(\mathcal{T}')$. From this it follows that b is both *sopp* and *sipp* in $\mathcal{T}$ as well as in $\mathcal{T}'$.

Moreover, action b is *ms* in both $\mathcal{T}$ and $\mathcal{T}'$ since we have $\mathrm{proj}_{\{1,2\}}{}^{[2]}(\delta_b) = \mathrm{proj}_{\{1,2\}}{}^{[2]}(\delta'_b) = \{((q_1, q_2), (q_1', q_2'))\} \subseteq \{((q_1, q_2), (q_1', q_2'))\} = (\delta_{\{1,2\}})_b =$

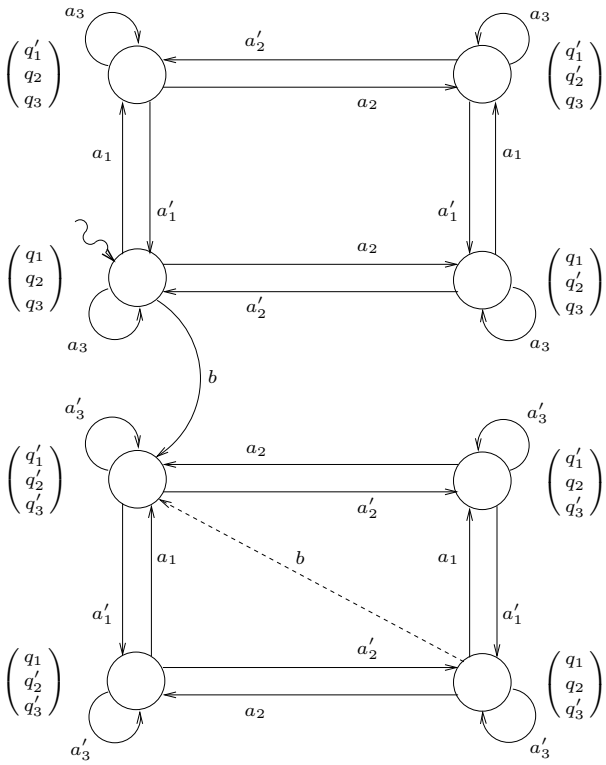

Fig. 5.9. Team automata $\mathcal{T}$ and $\mathcal{T}'$.

$(\delta'_{\{1,2\}})_b$, i.e. the output subteam of b participates in every b-transition of the team automata. In fact, b is even *sms* in $\mathcal{T}$ as b is *ms* in $\mathcal{T}$ and $\text{proj}_{\{3\}}{}^{[2]}(\delta_b) = \{(q_3, q_3')\} \subseteq \{(q_3, q_3')\} = (\delta_{\{3\}})_b$, i.e. also the input subteam of b participates in every b-transition of $\mathcal{T}$. It is clear that b is also *wms* in $\mathcal{T}$. However, $\text{proj}_{\{3\}}{}^{[2]}(\delta'_b) = \{((q_3), (q_3')), ((q_3'), (q_3'))\} \not\subseteq \{((q_3), (q_3'))\} = (\delta'_{\{3\}})_b$ and b is thus not *sms* in $\mathcal{T}'$. Since q_3 is the only state of $\mathcal{C}_3$ at which b is enabled in $\mathcal{C}_3$ we do have that b is *wms* in $\mathcal{T}'$.

The fact that $\mathcal{T}$ does not allow an output action b to take place without a "slave" input action b leads to b being *sms* in $\mathcal{T}$. In $\mathcal{T}'$, however, b is *wms* since the input action b follows the "master" output action b only when enabled.

To understand that despite the similarities this subtle difference — due to the distinction between *ai* and *si* — may lead to different externally observable behaviors of $\mathcal{T}$ and $\mathcal{T}'$, it is sufficient to show that $ba_1'a_2'b \in \mathbf{B}^{\Sigma}_{\mathcal{T}'}$ while no word with two b's is contained in $\mathbf{B}^{\Sigma}_{\mathcal{T}}$. The computation $(q_1, q_2, q_3)b(q_1', q_2', q_3')a_1'(q_1, q_2', q_3')a_2'(q_1, q_2, q_3')b(q_1', q_2', q_3') \in \mathbf{C}_{\mathcal{T}'}$ proves

that $ba'_1a'_2b \in \mathbf{B}^{\Sigma}_{\mathcal{T}'}$, whereas in δ the execution of b from the initial state (q_1, q_2, q_3) always leads to (q'_1, q'_2, q'_3), after which (q_1, q_2, q_3) — the only state from which b can be executed — has become unreachable. □

5.3.4 Peer-to-Peer and Master-Slave

We continue our comparison of the various types of synchronization started in Subsection 4.4.4 by extending our study to the types of synchronization introduced in this section.

First we revisit the synchronizations introduced in Section 4.4. This time, however, we deal with team automata rather than synchronized automata and we thus have a distribution of the alphabet of actions into input, output, and internal actions. We immediately note that if a is an internal action of one of the component automata of a team automaton $\mathcal{T}$, then it is not an action of any other component automaton of $\mathcal{T}$, in which case a thus trivially is *free*, *ai*, and *si* in $\mathcal{T}$.

Lemma 5.3.12. $\Sigma_{int} \subseteq Free(\mathcal{T}) \cap AI(\mathcal{T})$.

Proof. Let $a \in \Sigma_{int}$. From Definition 5.1.4 it follows that for all $(q, q') \in \delta_a$ there exists a unique $i \in \mathcal{I}$ such that $(\mathrm{proj}_i(q), a, \mathrm{proj}_i(q')) \in \delta_i$ and, moreover, $a \notin \bigcup_{j \in \mathcal{I} \setminus \{i\}} \Sigma_j$. Hence a trivially is *free*, *ai*, and *si*. □

We continue our investigation by involving also the synchronizations introduced in Section 5.3. We begin by comparing the various types of peer-to-peer (master-slave) synchronization among each other.

Definition 5.3.4 and Lemma 4.4.7 directly imply that actions that are *sipp* (*sopp*) are also *wipp* (*wopp*).

Lemma 5.3.13. *(1)* $SIPP(\mathcal{T}) \subseteq WIPP(\mathcal{T})$ *and*

(2) $SOPP(\mathcal{T}) \subseteq WOPP(\mathcal{T})$. □

From Example 4.4.8 we immediately conclude that the inclusions of this lemma in general do not hold the other way around.

From Definition 5.3.7 we immediately obtain that the fact that an action is *sms* implies that it is *wms*, which in its turn implies that it is *ms*.

Lemma 5.3.14. $SMS(\mathcal{T}) \subseteq WMS(\mathcal{T}) \subseteq MS(\mathcal{T})$. □

In Example 5.3.11 we have seen an example of a synchronization that is *wms* but not *sms*. This implies that also the inclusion of this lemma in general does not hold the other way around.

We now continue our investigation by comparing the various types of peer-to-peer (master-slave) synchronizations with the types of synchronization introduced in Section 4.4.

First we consider the types of peer-to-peer synchronization. Recall that $\Sigma_{out} = \bigcup_{i\in\mathcal{I}} \Sigma_{i,out}$, whereas Σ_{inp} need not equal $\bigcup_{i\in\mathcal{I}} \Sigma_{i,inp}$.

Theorem 5.3.15. *(1)* $(\bigcup_{i\in\mathcal{I}} \Sigma_{i,inp}) \cap AI(\mathcal{T}) \subseteq SIPP(\mathcal{T})$,

(2) $(\bigcup_{i\in\mathcal{I}} \Sigma_{i,inp}) \cap SI(\mathcal{T}) \subseteq WIPP(\mathcal{T})$,

(3) $\Sigma_{out} \cap AI(\mathcal{T}) \subseteq SOPP(\mathcal{T})$, *and*

(4) $\Sigma_{out} \cap SI(\mathcal{T}) \subseteq WOPP(\mathcal{T})$.

Proof. (1) Let $a \in (\bigcup_{i\in\mathcal{I}} \Sigma_{i,inp}) \cap AI(\mathcal{T})$. According to Definition 5.3.4(1) it remains to prove that $a \in AI(SUB_{a,inp})$. However, $a \in \bigcup_{i\in\mathcal{I}} \Sigma_{i,inp}$ implies that $\mathcal{I}_{a,inp} \neq \varnothing$ and since $a \in AI(\mathcal{T})$, it thus follows directly from Lemma 4.7.1(2) that $a \in AI(SUB_{a,inp})$.

(2-4) Analogous. □

In the following example we show that in general none of the inclusions of this theorem holds also the other way around.

Example 5.3.16. (Example 4.4.8 continued) We turn automata $\mathcal{A}_1$ and $\mathcal{A}_2$ into component automata $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively, each with input action a. This is done in the obvious way, viz. $\mathcal{C}_1 = (\{q, q'\}, (\{a\}, \varnothing, \varnothing), \{(q, a, q')\}, \{q\})$ and $\mathcal{C}_2 = (\{r, r'\}, (\{a\}, \varnothing, \varnothing), \{(r, a, r')\}, \{r\})$. Note that $\mathrm{und}(\mathcal{C}_1) = \mathcal{A}_1$ and $\mathrm{und}(\mathcal{C}_2) = \mathcal{A}_2$ are depicted in Figure 4.10.

Now consider the team automaton $\widehat{\mathcal{T}}^1 = (\{(q, r), (q, r'), (q', r), (q', r')\}, (\{a\}, \varnothing, \varnothing), \delta^1, \{(q, r)\})$, where we recall that $\delta^1 = \{((q, r), a, (q, r')), ((q, r), a, (q', r'))\}$. Then it is clear that input action a is not *si* and thus neither *ai*. However, in $SUB_{\{2\}}(\widehat{\mathcal{T}}^1)$ — which is essentially a copy of $\mathcal{C}_2$ — action a trivially is *sipp* and *wipp*.

In an analogous way we can show that in general neither of the inclusions stated in Theorem 5.3.15(3,4) holds the other way around as well. □

Next we consider the types of master-slave synchronization.

Theorem 5.3.17. $\Sigma_{out} \cap AI(\mathcal{T}) \subseteq MS(\mathcal{T})$.

Proof. Let $a \in \Sigma_{out} \cap AI(\mathcal{T})$ and let $(q, q') \in \delta_a$. Then for all $j \in \mathcal{I}_{a,out}$, we have that $\mathrm{proj}_j{}^{[2]}(q, q') \in \delta_{j,a}$. This implies that it must be the case that $\mathrm{proj}_{\mathcal{I}_{a,out}}{}^{[2]}(\delta_a) \subseteq (\delta_{\mathcal{I}_{a,out}})_a$ and thus $a \in MS(\mathcal{T})$. □

In the following example we show that in general the inclusion of this theorem does not hold also the other way around.

Example 5.3.18. Consider the composable system $\{\mathcal{C}_1, \mathcal{C}_2\}$ consisting of component automata $\mathcal{C}_i = (\{q_i, q_i'\}, (\varnothing, \{a\}, \varnothing), \{(q_i, a, q_i')\}, \{q_i\})$, with $i \in [2]$. It is depicted in Figure 5.10(a).

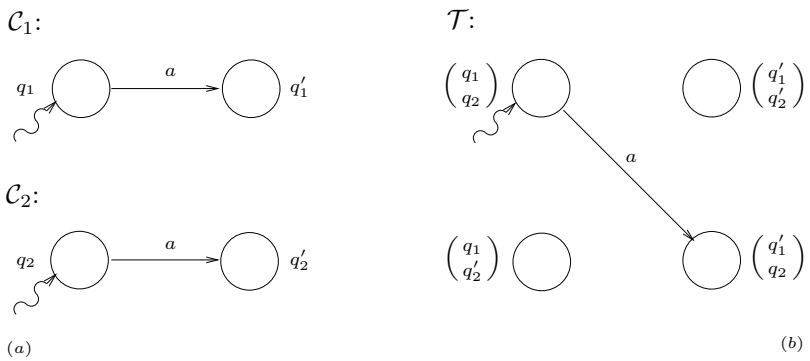

Fig. 5.10. Component automata $\mathcal{C}_1$ and $\mathcal{C}_2$, and team automaton $\mathcal{T}$.

Now consider team automaton $\mathcal{T} = (\{(q_1, q_2), (q_1', q_2), (q_1, q_2'), (q_1', q_2')\}, (\varnothing, \{a\}, \varnothing), \{((q_1, q_2), a, (q_1', q_2))\}, \{(q_1, q_2)\})$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$, depicted in Figure 5.10(b).

Clearly $\mathcal{I}_{a,out}(\{\mathcal{C}_1, \mathcal{C}_2\}) = \{1, 2\}$. Hence a trivially is *ms* (*sms*, *wms*) in $\mathcal{T}$, but a is not *ai* in $\mathcal{T}$ since $\mathcal{C}_2$ does not participate in the a-transition of $\mathcal{T}$ even though it has a in its alphabet. □

The preceding two theorems immediately imply the following result.

Corollary 5.3.19. $\Sigma_{out} \cap AI(\mathcal{T}) \subseteq SOPP(\mathcal{T}) \cap MS(\mathcal{T})$. □

Finally we involve also *sms* and *wms* actions.

Theorem 5.3.20. *If* $\Sigma_{out} \subseteq AI(\mathcal{T})$, *then* $MS(\mathcal{T}) = SMS(\mathcal{T}) = WMS(\mathcal{T})$.

Proof. Let $\Sigma_{out} \subseteq AI(\mathcal{T})$. Now let $a \in MS(\mathcal{T})$. Then by Definition 5.3.7(1), $a \in \Sigma_{out}$ and thus also $a \in AI(\mathcal{T})$. We distinguish two cases.
If there does not exist a $j \in \mathcal{I}$ such that $a \in \Sigma_{j,inp}$, then $\mathcal{I}_{a,inp} = \varnothing$ and thus trivially $a \in SMS(\mathcal{T})$.
If there exist a $j \in \mathcal{I}$ such that $a \in \Sigma_{j,inp}$, then $\mathcal{I}_{a,inp} \neq \varnothing$ and, because a is *ai*, $\mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(\delta_a) \subseteq (\delta_{\mathcal{I}_{a,inp}})_a$. Hence $a \in SMS(\mathcal{T})$.

In both cases we thus obtain that $a \in SMS(\mathcal{T})$. Hence $MS(\mathcal{T}) \subseteq SMS(\mathcal{T})$ and since, by Lemma 5.3.14, $SMS(\mathcal{T}) \subseteq WMS(\mathcal{T}) \subseteq MS(\mathcal{T})$ the equality follows. □

5.4 Predicates of Synchronizations

In the preceding sections of this chapter we have presented our team automata framework. We have seen that team automata over composable systems are themselves component automata that can be used in further constructions of team automata. Team automata can thus be used as building blocks. We have analyzed the transition relations of team automata in order to determine whether or not they satisfy the conditions inherent to certain specific types of synchronization modeling collaboration between system components. However, we have seen that these conditions in general do not lead to uniquely defined team automata.

To make the model of team automata of any use, e.g. in the early phases of system design, it is necessary to be able to unambiguously construct a team automaton according to the specification of the required type of synchronization. Given a composable system and certain conditions to be satisfied by the synchronizations, we want to construct the unique team automaton over this composable system. This is done in very much the same way as we constructed the *maximal-free* (*maximal-ai*, *maximal-si*) synchronized automata of Section 4.5, viz. by defining predicates of synchronization. Since for an internal action the transition relation is by definition equal to its complete transition space in $\mathcal{S}$, we need to choose predicates only for all external actions. Once we do so, the team automaton over $\mathcal{S}$ defined by these predicates is unique.

Based on Definition 4.5.1, this is formalized as follows.

Definition 5.4.1. *Let $\mathcal{R}_a(\mathcal{S}) \subseteq \Delta_a(\mathcal{S})$, for all $a \in \Sigma_{ext}$, and let $\mathcal{R}_a(\mathcal{S}) = \Delta_a(\mathcal{S})$, for all $a \in \Sigma_{int}$. Let $\mathcal{R} = \{\mathcal{R}_a(\mathcal{S}) \mid a \in \Sigma\}$. Then $\mathcal{T}$ is* the $\mathcal{R}$-team automaton *over $\mathcal{S}$ if for all $a \in \Sigma$,*

$$\delta_a = \mathcal{R}_a(\mathcal{S}). \qquad \square$$

In Section 4.5 we have seen that each of the predicates $\mathcal{R}_a^{free}(\mathcal{S})$, $\mathcal{R}_a^{ai}(\mathcal{S})$, and $\mathcal{R}_a^{si}(\mathcal{S})$ defines the largest transition relation in $\Delta_a(\mathcal{S})$ in which an action a is *free*, *ai*, and *si*, respectively.

As an immediate corollary of Theorem 4.5.5 we obtain that in case of an internal action, each such a predicate equals the *no-constraints* predicate, i.e. its complete transition space in $\mathcal{S}$.

Theorem 5.4.2. *Let $a \in \Sigma_{int}$. Then*

$$\Delta_a(\mathcal{S}) = \mathcal{R}_a^{no}(\mathcal{S}) = \mathcal{R}_a^{syn}(\mathcal{S}), \text{ for all } syn \in \{free, ai, si\}. \qquad \square$$

The generic setup of Definition 5.4.1 now allows us to define three specific team automata as an extension of Definition 4.5.4.

Definition 5.4.3. *Let* $syn \in \{free, ai, si\}$. *Then*

the $\{\mathcal{R}_a^{syn}(\mathcal{S}) \mid a \in \Sigma\}$*-team automaton over* $\mathcal{S}$ *is called the* maximal-syn team automaton *(over* $\mathcal{S}$*).* □

We now consider the constraints relating to the types of synchronization defined in Section 5.3. This will allow us to define more types of team automata than those of Definition 5.4.3. We define the predicates of synchronization without any reference to a team automaton, its subteams, and its transition relation.

We begin by considering the peer-to-peer types of synchronization. In this case we have to distinguish between the input and output role an external action a may have in $\mathcal{S}$. The predicates thus have to refer to the input and output domains of a in $\mathcal{S}$. Moreover, we have to distinguish between strong (ai) and weak (si) types of synchronization. This leads to four predicates, each of which includes all and only those transitions from $\Delta_a(\mathcal{S})$ in which all component automata given by the input or output domain, respectively, are forced (in the strong or in the weak sense) to participate.

Recall that, for an external action a, $\mathcal{I}_{a,inp}(\mathcal{S}) = \{i \in \mathcal{I} \mid a \in \Sigma_{i,inp}\}$ is the input domain of a in $\mathcal{S}$ and $\mathcal{I}_{a,out}(\mathcal{S}) = \{i \in \mathcal{I} \mid a \in \Sigma_{i,out}\}$ is the output domain of a in $\mathcal{S}$. As before, we may simply write $\mathcal{I}_{a,inp}$ and $\mathcal{I}_{a,out}$, since $\mathcal{S}$ has been fixed.

First we focus on input actions.

Definition 5.4.4. *Let* $a \in \Sigma$ *and let* $\mathcal{S}_{a,inp} = \{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\}$. *Then*

(1) the predicate is-sipp *in* $\mathcal{S}$ *for* a *is denoted by* $\mathcal{R}_a^{sipp}(\mathcal{S})$ *and is defined as*

if $a \in \bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}$, *then*

$$\mathcal{R}_a^{sipp}(\mathcal{S}) = \{(q,q') \in \Delta_a(\mathcal{S}) \mid proj_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \Delta_a(\mathcal{S}_{a,inp}) \Rightarrow proj_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \mathcal{R}_a^{ai}(\mathcal{S}_{a,inp})\},$$

otherwise

$$\mathcal{R}_a^{sipp}(\mathcal{S}) = \Delta_a(\mathcal{S}), \text{ and}$$

(2) the predicate is-wipp *in* $\mathcal{S}$ *for* a *is denoted by* $\mathcal{R}_a^{wipp}(\mathcal{S})$ *and is defined as*

if $a \in \bigcup_{i\in\mathcal{I}} \Sigma_{i,inp}$, *then*

$$\mathcal{R}_a^{wipp}(\mathcal{S}) = \{(q,q') \in \Delta_a(\mathcal{S}) \mid proj_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \Delta_a(\mathcal{S}_{a,inp}) \Rightarrow proj_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \mathcal{R}_a^{si}(\mathcal{S}_{a,inp})\},$$

otherwise

$$\mathcal{R}_a^{wipp}(\mathcal{S}) = \Delta_a(\mathcal{S}).$$ □

Next we focus on output actions.

Definition 5.4.5. *Let* $a \in \Sigma$ *and let* $\mathcal{S}_{a,out} = \{\mathcal{C}_i \mid i \in \mathcal{I}_{a,out}\}$. *Then*

(1) the predicate is-sopp *in* $\mathcal{S}$ *for* a *is denoted by* $\mathcal{R}_a^{sopp}(\mathcal{S})$ *and is defined as*

if $a \in \bigcup_{i\in\mathcal{I}} \Sigma_{i,out}$, *then*

$$\mathcal{R}_a^{sopp}(\mathcal{S}) = \{(q,q') \in \Delta_a(\mathcal{S}) \mid proj_{\mathcal{I}_{a,out}}{}^{[2]}(q,q') \in \Delta_a(\mathcal{S}_{a,out}) \Rightarrow proj_{\mathcal{I}_{a,out}}{}^{[2]}(q,q') \in \mathcal{R}_a^{ai}(\mathcal{S}_{a,out})\},$$

otherwise

$$\mathcal{R}_a^{sopp}(\mathcal{S}) = \Delta_a(\mathcal{S}), \text{ and}$$

(2) the predicate is-wopp *in* $\mathcal{S}$ *for* a *is denoted by* $\mathcal{R}_a^{wopp}(\mathcal{S})$ *and is defined as*

if $a \in \bigcup_{i\in\mathcal{I}} \Sigma_{i,out}$, *then*

$$\mathcal{R}_a^{wopp}(\mathcal{S}) = \{(q,q') \in \Delta_a(\mathcal{S}) \mid proj_{\mathcal{I}_{a,out}}{}^{[2]}(q,q') \in \Delta_a(\mathcal{S}_{a,out}) \Rightarrow proj_{\mathcal{I}_{a,out}}{}^{[2]}(q,q') \in \mathcal{R}_a^{si}(\mathcal{S}_{a,out})\},$$

otherwise

$$\mathcal{R}_a^{wopp}(\mathcal{S}) = \Delta_a(\mathcal{S}).$$ □

One should recall at this point that we are not discussing the properties of a given team automaton over $\mathcal{S}$, with a fixed transition relation determining the transitions in the input and output subteams of an external action a. Thus, in Definitions 5.4.4 and 5.4.5, we relate to the complete transition spaces of a in the respective "subsystems" determined by the input and output domain of a. Each predicate includes all and only those transitions from $\Delta_a(\mathcal{S})$, for which

all component automata given by the input or output domain, respectively, are forced (in the weak or in the strong sense) to participate in the execution of a by any of these component automata.

As the next result shows, the predicates of Definitions 5.4.4 and 5.4.5 describe the maximal sets of a-transitions satisfying the given constraint. Recall that $\Sigma_{out} = \bigcup_{i \in \mathcal{I}} \Sigma_{i,out}$.

Theorem 5.4.6. *Let $a \in \bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}$. Then*

(1) $a \in SIPP(\mathcal{T})$ if and only if $\delta_a \subseteq \mathcal{R}_a^{sipp}(\mathcal{S})$, and

(2) $a \in WIPP(\mathcal{T})$ if and only if $\delta_a \subseteq \mathcal{R}_a^{wipp}(\mathcal{S})$.

Let $a \in \Sigma_{out}$. Then

(3) $a \in SOPP(\mathcal{T})$ if and only if $\delta_a \subseteq \mathcal{R}_a^{sopp}(\mathcal{S})$, and

(4) $a \in WOPP(\mathcal{T})$ if and only if $\delta_a \subseteq \mathcal{R}_a^{wopp}(\mathcal{S})$.

Proof. (1) (Only if) Let $a \in SIPP(\mathcal{T})$. Hence according to Definition 5.3.4(1) we have $a \in AI(SUB_{a,inp})$, i.e. a is ai in the subteam of $\mathcal{T}$ determined by the input domain of a. According to Definition 4.1.6 the a-transitions of this subteam are $(\delta_{\mathcal{I}_{a,inp}})_a = \mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(\delta_a) \cap \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$. Now, by Theorem 4.5.3(2), $a \in AI(SUB_{a,inp})$ implies that $(\delta_{\mathcal{I}_{a,inp}})_a \subseteq \mathcal{R}_a^{ai}(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$. Hence for all $(q,q') \in \delta_a$, whenever $\mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$, then $\mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \mathcal{R}_a^{ai}(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$. Consequently, according to Definition 5.4.4(1), $\delta_a \subseteq \mathcal{R}_a^{sipp}(\mathcal{S})$.

(If) Let $\delta_a \subseteq \mathcal{R}_a^{sipp}(\mathcal{S})$. By Definition 5.3.4(1) we now have to prove that $a \in AI(SUB_{a,inp})$. Since $a \in \bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}$, we know that $\mathcal{I}_{a,inp} \neq \varnothing$. Hence consider an arbitrary pair $(p,p') \in (\delta_{\mathcal{I}_{a,inp}})_a$. Since $(p,p') \in (\delta_{\mathcal{I}_{a,inp}})_a = \mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(\delta_a) \cap \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$ there is a $(q,q') \in \delta_a \subseteq \Delta_a(\mathcal{S})$ for which $\mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') = (p,p')$. From $\delta_a \subseteq \mathcal{R}_a^{sipp}(\mathcal{S})$ we infer that $(p,p') \in \mathcal{R}_a^{ai}(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$. Hence $(\delta_{\mathcal{I}_{a,inp}})_a \subseteq \mathcal{R}_a^{ai}(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$ and thus, by Theorem 4.5.3(2), $a \in AI(SUB_{a,inp})$.

(2-4) Analogous. □

Now we turn to the master-slave types of synchronization. As in the case of the peer-to-peer predicates, we have to distinguish between the input and the output role of actions. This time, however, the predicates describe synchronizations *between* the component automata from the input domain and those from the output domain.

The *is-ms* predicate for an external action a includes all and only those a-transitions in which a appears at least once in its output role. For the

predicates *is-sms* and *is-wms* in $\mathcal{S}$, there is the additional requirement that a should also be executed by the component automata from its input domain. In the strong case, this obligation is strict in the sense that if the input domain of a is not empty, then always at least one component automaton from the input domain of a participates in every a-transition included in the predicate. In the weak case, this obligation has to be met only when at least one component automaton from the input domain of a is ready to execute a.

Definition 5.4.7. *Let $a \in \Sigma$, let $\mathcal{S}_{a,inp} = \{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\}$, and let $\mathcal{S}_{a,out} = \{\mathcal{C}_i \mid i \in \mathcal{I}_{a,out}\}$. Then*

(1) the predicate is-ms *in $\mathcal{S}$ for a is denoted by $\mathcal{R}_a^{ms}(\mathcal{S})$ and is defined as*

if $a \in \Sigma_{out}$, then

$$\mathcal{R}_a^{ms}(\mathcal{S}) = \{(q,q') \in \Delta_a(\mathcal{S}) \mid proj_{\mathcal{I}_{a,out}}{}^{[2]}(q,q') \in \Delta_a(\mathcal{S}_{a,out})\},$$

otherwise

$$\mathcal{R}_a^{ms}(\mathcal{S}) = \Delta_a(\mathcal{S}),$$

(2) the predicate is-sms *in $\mathcal{S}$ for a is denoted by $\mathcal{R}_a^{sms}(\mathcal{S})$ and is defined as*

if $a \in \Sigma_{out}$, then

$$\mathcal{R}_a^{sms}(\mathcal{S}) = \mathcal{R}_a^{ms}(\mathcal{S}) \cap \{(q,q') \in \Delta_a(\mathcal{S}) \mid \mathcal{I}_{a,inp} \neq \varnothing \Rightarrow proj_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \Delta_a(\mathcal{S}_{a,inp})\},$$

otherwise

$$\mathcal{R}_a^{sms}(\mathcal{S}) = \Delta_a(\mathcal{S}), \text{ and}$$

(3) the predicate is-wms *in $\mathcal{S}$ for a is denoted by $\mathcal{R}_a^{wms}(\mathcal{S})$ and is defined as*

if $a \in \Sigma_{out}$, then

$$\mathcal{R}_a^{wms}(\mathcal{S}) = \mathcal{R}_a^{ms}(\mathcal{S}) \cap \{(q,q') \in \Delta_a(\mathcal{S}) \mid \mathcal{I}_{a,inp} \neq \varnothing \Rightarrow [(\exists i \in \mathcal{I}_{a,inp} : a \ en_{\mathcal{C}_i} \ proj_i(q)) \Rightarrow proj_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \Delta_a(\mathcal{S}_{a,inp})]\},$$

otherwise

$$\mathcal{R}_a^{wms}(\mathcal{S}) = \Delta_a(\mathcal{S}).$$ □

The *is-ms* (*is-sms*, *is-wms*) predicate guarantees that the output action a is indeed *ms* (*sms*, *wms*) in every team automaton over $\mathcal{S}$ with that predicate for its a-transitions. The predicates *is-ms* and *is-sms*, moreover, are the largest set of a-transitions satisfying the specified constraint.

It is, however, not necessarily the case that every set of a-transitions by which a is *is-wms* is contained in the predicate *is-wms*. This difference stems from the fact that the predicate refers to component automata from the input domain of a rather than an input subteam. There is no way out and in fact the maximality principle is not applicable, because to define a subteam with transitions, a team automaton including the transition relation should have been defined already. Since a subteam only contains a selection of all possible a-transitions, it may happen that a is enabled in a component automaton of the input subteam, but not in the subteam. Thus a can be *wms* in team automaton $\mathcal{T}$ even when δ_a contains transitions in which the input subteam of a does not participate, although a is currently enabled in a component automaton of this subteam.

Theorem 5.4.8. *Let $a \in \Sigma_{out}$. Then*

(1) $a \in MS(\mathcal{T})$ if and only if $\delta_a \subseteq \mathcal{R}_a^{ms}(\mathcal{S})$,

(2) $a \in SMS(\mathcal{T})$ if and only if $\delta_a \subseteq \mathcal{R}_a^{sms}(\mathcal{S})$, and

(3) if $\delta_a \subseteq \mathcal{R}_a^{wms}(\mathcal{S})$, then $a \in WMS(\mathcal{T})$.

Proof. (1) (Only if) Let $a \in MS(\mathcal{T})$. Hence by Lemma 5.3.8(1) we have $\mathrm{proj}_{\mathcal{I}_{a,out}}{}^{[2]}(\delta_a) = (\delta_{\mathcal{I}_{a,out}})_a$. By Definition 4.1.6 consequently $(\delta_{\mathcal{I}_{a,out}})_a = \mathrm{proj}_{\mathcal{I}_{a,out}}{}^{[2]}(\delta_a) \cap \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,out}\})$ and thus $\mathrm{proj}_{\mathcal{I}_{a,out}}{}^{[2]}(\delta_a) \subseteq \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,out}\})$. Hence by Definition 5.4.7(1), $\delta_a \subseteq \mathcal{R}_a^{ms}(\mathcal{S})$.

(If) Let $\delta_a \subseteq \mathcal{R}_a^{ms}(\mathcal{S})$. Then by Definition 5.3.7(1) we have to prove that $\mathrm{proj}_{\mathcal{I}_{a,out}}{}^{[2]}(\delta_a) \subseteq (\delta_{\mathcal{I}_{a,out}})_a$. By Definition 4.1.6 we thus have to prove $\mathrm{proj}_{\mathcal{I}_{a,out}}{}^{[2]}(\delta_a) \subseteq \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,out}\})$. This follows immediately from Definition 5.4.7(1).

(2) Let $a \in SMS(\mathcal{T})$. If $\mathcal{I}_{a,inp} = \varnothing$, then there is nothing to prove. Hence assume that $\mathcal{I}_{a,inp} \neq \varnothing$. As in the proof of (1), for $\mathcal{I}_{a,out}$ it is easy to prove that $\mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(\delta_a) \subseteq (\delta_{\mathcal{I}_{a,inp}})_a$ if and only if $\delta_a \subseteq \{(q,q') \in \Delta_a(\mathcal{S}) \mid \mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})\}$. By using Definition 5.3.7(2) we thus infer that $a \in SMS(\mathcal{T})$ if and only if $\delta_a \subseteq \mathcal{R}_a^{ms}(\mathcal{S})$ and $\delta_a \subseteq \{(q,q') \in \Delta_a(\mathcal{S}) \mid \mathrm{proj}_{\mathcal{I}_{a,inp}}{}^{[2]}(q,q') \in \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})\}$. Hence according to Definition 5.4.7(2) we are ready.

(3) Again there is nothing to prove whenever $\mathcal{I}_{a,inp} = \varnothing$. Hence assume that $\mathcal{I}_{a,inp} \neq \varnothing$. Let $\delta_a \subseteq \mathcal{R}_a^{wms}(\mathcal{S})$. Then by Definition 5.3.7(3) we have

to prove that whenever $(q, q') \in \delta_a$ and a en $_{SUB_{a,inp}}$ proj$_{\mathcal{I}_{a,inp}}(q)$, then proj$_{\mathcal{I}_{a,inp}}{}^{[2]}(q, q') \in (\delta_{\mathcal{I}_{a,inp}})_a$. Definition 5.4.7(3) implies that for all $(q, q') \in \delta_a$, if there is an $i \in \mathcal{I}_{a,inp}$ for which a en $_{\mathcal{C}_i}$ proj$_i(q)$, then proj$_{\mathcal{I}_{a,inp}}{}^{[2]}(q, q') \in \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$. Since a en $_{SUB_{a,inp}}$ proj$_{\mathcal{I}_{a,inp}}(q)$ implies that then there is an $i \in \mathcal{I}_{a,inp}$ for which a en $_{\mathcal{C}_i}$ proj$_i(q)$, we now have that if $(q, q') \in \delta_a$ and a en $_{SUB_{a,inp}}$ proj$_{\mathcal{I}_{a,inp}}(q)$, then proj$_{\mathcal{I}_{a,inp}}{}^{[2]}(q, q') \in \Delta_a(\{\mathcal{C}_i \mid i \in \mathcal{I}_{a,inp}\})$. Now Definition 4.1.6 implies that $(\delta_{\mathcal{I}_{a,inp}})_a =$ proj$_{\mathcal{I}_{a,inp}}{}^{[2]}(q, q')$ and thus we are ready. □

In the following example we show that, as announced before, the converse of Theorem 5.4.8(3) in general indeed does not hold.

Example 5.4.9. Let $\mathcal{C}_1 = (\{q_1, q_2\}, (\{a\}, \varnothing, \varnothing), \{(q_1, a, q_1')\}, \{q_1\})$ and $\mathcal{C}_2 = (\{q_2, q_2'\}, (\varnothing, \{a\}, \varnothing), \{(q_2, a, q_2')\}, \{q_2\})$ be the two component automata depicted in Figure 5.11(a).

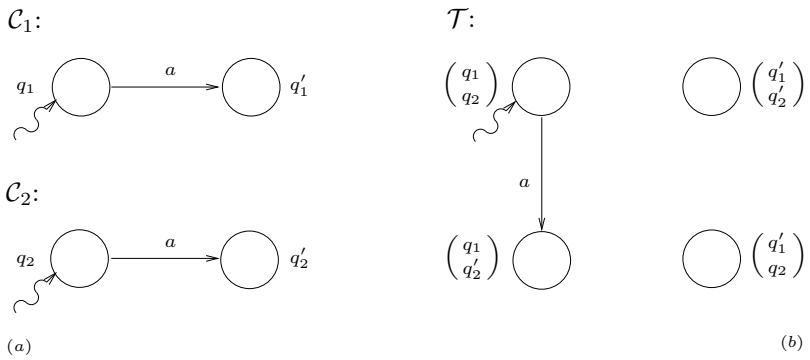

Fig. 5.11. Component automata $\mathcal{C}_1$ and $\mathcal{C}_2$, and team automaton $\mathcal{T}$.

Clearly $\mathcal{S} = \{\mathcal{C}_1, \mathcal{C}_2\}$ is a composable system. Consider team automaton $\mathcal{T} = (\{(q_1, q_2), (q_1, q_2'), (q_1', q_2), (q_1', q_2')\}, (\varnothing, \{a\}, \varnothing), \{((q_1, q_2), a, (q_1, q_2'))\}, \{(q_1, q_2)\})$ over $\mathcal{S}$. It is depicted in Figure 5.11(b). Since a is not enabled in state (q_1) of the input subteam of $\mathcal{T}$ it is trivial to see that $a \in WMS(\mathcal{T})$. Note however that a is enabled in state q_1 of component automaton $\mathcal{C}_1$ of the input subteam. Since this component automaton does not participate in the a-transition $((q_1, q_2), (q_1, q_2'))$ of $\mathcal{T}$, however, we have found that $((q_1, q_2), (q_1, q_2')) \in \delta_a \setminus \mathcal{R}_a^{wms}(\mathcal{S})$. □

Summarizing we thus conclude that except for *wms*, each of the types of synchronization introduced in Section 5.3 — as did each of the types introduced

in Section 4.4 — gives rise to a predicate that is the unique maximal representative among all transition relations satisfying the constraints implied by the type of synchronization. Consequently, we can now distinguish more specific types of team automata.

Definition 5.4.10. *Let* $syn \in \{sipp, wipp, sopp, wopp, ms, sms\}$. *Then*

(1) the $\{\mathcal{R}_a^{syn}(\mathcal{S}) \mid a \in \Sigma\}$*-team automaton over* $\mathcal{S}$ *is called the* maximal-syn team automaton *(over* $\mathcal{S}$*) and*

(2) an action $a \in \Sigma$ *is called* maximal-syn *in* $\mathcal{T}$ *if* $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$. □

5.4.1 Homogeneous Versus Heterogeneous

The team automata from Definitions 5.4.3 and 5.4.10(1) differ by the type of predicate that needs to be satisfied. However, it is one and the same predicate that needs to be satisfied by *all* external actions. Such team automata are called *homogeneous*, as opposed to team automata for which different subsets of external actions satisfy (potentially) different predicates, which are called *heterogeneous*.

When defining heterogeneous team automata we need to specify exactly which (combinations of) predicates must hold for which subsets of external actions. Consider, e.g., that we want to construct a team automaton over $\mathcal{S}$ such that *all* of its input actions are *ai*, while *all* of its locally-controlled actions are *ms*. Then we construct the $\{\mathcal{R}_a^{ai}(\mathcal{S}) \mid a \in \Sigma_{inp}\} \cup \{\mathcal{R}_a^{ms}(\mathcal{S}) \mid a \in \Sigma_{loc}\}$-team automaton over $\mathcal{S}$, which is thus an example of a heterogeneous team automaton.

Example 5.4.11. (Example 4.2.8 continued) We turn the automata $\mathcal{A}_1$ and $\mathcal{A}_2$, depicted in Figure 4.7(a), into component automata $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively, by distributing their respective alphabets over input, output, and internal alphabets. We let a and b be input actions in $\mathcal{C}_1$ and we let a be an output action in $\mathcal{C}_2$. Consequently, $\mathcal{S} = \{\mathcal{C}_1, \mathcal{C}_2\}$ is a composable system. Note that any team automaton over $\mathcal{S}$ will have input alphabet $\{b\}$, output alphabet $\{a\}$, and an empty internal alphabet.

We now construct a homogeneous team automata over $\mathcal{S}$. The $\{\mathcal{R}_c^{sms}(\mathcal{S}) \mid c \in \Sigma\}$-team automaton $\mathcal{T}^1$ (i.e. the *maximal-sms* team automaton) over $\mathcal{S}$ is depicted in Figure 5.12(a).

It is easy to construct other homogeneous team automata over $\mathcal{S}$. The $\{\mathcal{R}_c^{ms}(\mathcal{S}) \mid c \in \Sigma\}$-team automaton over $\mathcal{S}$, e.g., is obtained by adding the transition $((q_1', q_2), a, (q_1', q_2'))$ to the transition relation of $\mathcal{T}^1$. The resulting *maximal-ms* team automaton $\mathcal{T}^2$ is depicted in Figure 5.12(b).

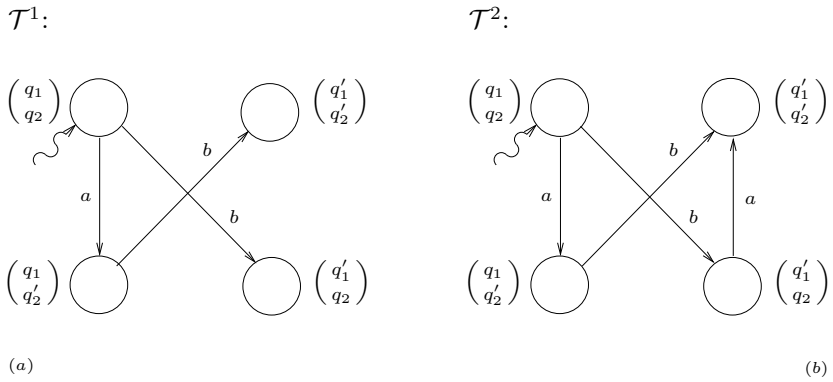

Fig. 5.12. Team automata $\mathcal{T}^1$ and $\mathcal{T}^2$.

It is also not difficult to construct heterogeneous team automata over $\mathcal{S}$. The $\{\mathcal{R}_c^{free}(\mathcal{S}) \mid c \in \Sigma_{inp}\} \cup \{\mathcal{R}_c^{ai}(\mathcal{S}) \mid c \in \Sigma_{out}\} \cup \{\Delta_c(\mathcal{S}) \mid c \in \Sigma_{int}\}$-team automaton over $\mathcal{S}$, e.g., is the team automaton $\mathcal{T}^1$ depicted in Figure 5.12(a). This is thus an example of a team automaton that is both homogeneous and heterogenous. □

As this example has shown, the dividing line between homogeneous and heterogeneous team automata is very thin.

We have paved the way for even more specific team automata that lie inbetween homogeneous and heterogeneous team automata, since we can also construct, e.g., the $\{\mathcal{R}_a^{sopp}(\mathcal{S}) \cap \mathcal{R}_a^{ms}(\mathcal{S}) \mid a \in \Sigma_{ext}\} \cup \{\Delta_a(\mathcal{S}) \mid a \in \Sigma_{int}\}$-team automaton over $\mathcal{S}$ or the $\{\mathcal{R}_a^{ai}(\mathcal{S}) \mid a \in \Sigma_{inp}\} \cup \{\mathcal{R}_a^{sopp}(\mathcal{S}) \cap \mathcal{R}_a^{ms}(\mathcal{S}) \mid a \in \Sigma_{out}\} \cup \{\Delta_a(\mathcal{S}) \mid a \in \Sigma_{int}\}$-team automaton over $\mathcal{S}$.

To conclude this section we make the observation that, given a composable system $\mathcal{S}$, there exist team automata over $\mathcal{S}$ that cannot be obtained as the homogeneous team automaton of any of the types introduced above. Shortly we will give an example of one such a team automaton. We moreover conjecture that it does not help to consider heterogeneous team automata. In other words, there exist team automata over $\mathcal{S}$ whose transition relations cannot be obtained as the result of any combination of the predicates introduced in Definitions 4.5.2, 5.4.4, 5.4.5, and 5.4.7.

Example 5.4.12. (Example 5.4.11 continued) Let $\mathcal{T}^3$ be obtained by removing the transition $((q_1, q_2), b, ((q_1', q_2))$ from the transition relation of $\mathcal{T}^2$. Now $\mathcal{T}^3$ is clearly a team automaton over $\mathcal{S}$. However, it is straightforward to verify that $\mathcal{T}^3$ cannot be obtained as the homogeneous team automaton defined by any of the predicates introduced in Definitions 4.5.2, 5.4.4, 5.4.5, and 5.4.7.

Furthermore, it seems unlikely that — given the current predicates — $\mathcal{T}^3$ can be obtained as a heterogeneous team automaton over $\mathcal{S}$. Intuitively, the reason for this resides in the fact that in $\mathcal{T}^3$, b is its only input action, its output domain is empty, and as far as its input domain is concerned, transitions $((q_1, q_2), b, ((q_1', q_2))$ and $((q_1, q_2'), b, (q_1', q_2'))$ cannot be distinguished. It thus appears to be the case that any team automaton over $\mathcal{S}$ that is constructed according to any (combination) of the predicates introduced in Definitions 4.5.2, 5.4.4, 5.4.5, and 5.4.7 will either contain none of the two b-transitions above, or both. □

Summarizing, in this section we have shown that there exists a large variety of combinations of types of synchronizations that can be used to model many intricate interactions among system components. Given that those components are modeled by component automata and that the interactions the system should exhibit are known, a designer can choose how to construct the unique team automaton over the component automata as a model of the system he or she set out to design.

5.5 Effect of Synchronizations

The (maximal) types of synchronization introduced earlier in this chapter, together with the (maximal) types of synchronization introduced in Sections 4.4 and 4.5, form a whole range of possible synchronizations within team automata. In Section 4.6 we studied the effect that the basic synchronizations *free*, *ai*, *si*, and their maximal variants have on the inheritance of the automata-theoretic properties of Section 3.2 from synchronized automata to their (sub)automata, and vice versa. In this section we extend this study to team automata, i.e. we now take into account that we deal with alphabets with a distinction into three distinct types of actions. We apply some restrictions, though.

First we do not extend this study to incorporate also the more complex types of synchronization introduced earlier on in this chapter. As already mentioned in the Introduction, such a full study is beyond the scope of this thesis. What we do provide is a systematic study of the role *free*, *ai*, and *si* actions play in our approach of modeling collaboration between system components through synchronizations of actions shared by these components.

Secondly, we do not take into account the properties action reducedness, transition reducedness, and state reducedness. Again, such a full study is beyond the scope of this thesis. Instead we focus on the inheritance of enabling and determinism from team automata to their constituents, and vice versa.

To this aim, the results of Section 4.6 are carried over to team automata, after which we study the specific role of the distinction of the set of actions of a team automaton into input, ouput, and internal actions. It turns out that we need to be particularly careful concerning the possibility of an action being input to a component automaton from $\mathcal{S}$ and output to the team automata over $\mathcal{S}$.

We start this section with a study of the top-down inheritance — from team automata to their subteams and component automata — of enabling and determinism. Subsequently we investigate also the bottom-up preservation — from subteams and component automata to team automata.

Notation 8. *For the remainder of this chapter we let $\Sigma_{i,ext}$ denote the set of external actions $\Sigma_{i,inp} \cup \Sigma_{i,out}$ of our fixed component automaton $\mathcal{C}_i$, where $i \in \mathcal{I}$, and we let $\Sigma_{i,loc}$ denote its set of locally-controlled actions $\Sigma_{i,out} \cup \Sigma_{i,int}$. Recall that Σ_i denotes its set of actions $\Sigma_{i,inp} \cup \Sigma_{i,out} \cup \Sigma_{i,int}$. Furthermore, we fix an arbitrary $j \in \mathcal{I}$ and an arbitrary subset $J \subseteq \mathcal{I}$. We let $\Sigma_{J,ext}$ denote the set of external actions $\Sigma_{J,inp} \cup \Sigma_{J,out}$ of the subteam SUB_J of $\mathcal{T}$ and we let $\Sigma_{J,loc}$ denote its set of locally-controlled actions $\Sigma_{J,out} \cup \Sigma_{J,int}$. Recall that Σ_J denotes its set of actions $\Sigma_{J,inp} \cup \Sigma_{J,out} \cup \Sigma_{J,int}$. Finally, recall that Σ denotes the set of actions $\Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$, Σ_{ext} denotes the set of external actions $\Sigma_{inp} \cup \Sigma_{out}$, and Σ_{loc} denotes the set of locally-controlled actions $\Sigma_{out} \cup \Sigma_{int}$ of any team automaton over our fixed composable system $\mathcal{S}$.* □

5.5.1 Top-Down Inheritance of Properties

In this subsection we search for sufficient conditions under which enabling and determinism are inherited from team automata to their subteams and component automata.

It is clear that Definitions 3.2.42 and 3.2.57 extend in a natural way to component automata. Given an alphabet Θ disjoint from the set of states, we can thus speak of a *Θ-enabling component automaton* and of a *Θ-deterministic component automaton*. Moreover, if Θ equals its set of actions, then we simply speak of enabling and deterministic component automata, respectively.

Finally, recall from Theorem 5.4.2 that for all $a \in \Sigma_{int}$, we know that $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$, for all $syn \in \{no, free, ai, si\}$.

Enabling

In case the distribution of the alphabet plays no role, then the results concerning the inheritance of enabling from team automata to their subteams and component automata can obviously be lifted from Theorem 4.6.19.

Theorem 5.5.1. *Let $\mathcal{T}$ be Θ-enabling. Then*

(1) if $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_J$, then SUB_J is Θ-enabling, and

(2) if $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_j$, then $\mathcal{C}_j$ is Θ-enabling. □

Since $\Sigma_{alph} \cap \Sigma_J \subseteq \Sigma_{J,alph}$ and $\Sigma_{alph} \cap \Sigma_j \subseteq \Sigma_{j,alph}$, for $alph \in \{inp, int, ext\}$, the following result follows immediately.

Corollary 5.5.2. *Let $alph \in \{inp, int, ext\}$ and let $\mathcal{T}$ be Σ_{alph}-enabling. Then*

(1) if $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Sigma_{J,alph}$, then SUB_J is Σ_{alph}-enabling, and

(2) if $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Sigma_{j,alph}$, then $\mathcal{C}_j$ is Σ_{alph}-enabling. □

Note that this corollary does not cover the cases in which $alph \in \{out, loc\}$. In the following example we show that the fact that a team automaton $\mathcal{T}$ over $\mathcal{S}$ is Σ_{out}-enabling in general does not imply that each of its subteams (component automata from $\mathcal{S}$) is Σ_{out}-enabling, not even if all its output actions are *ai* in $\mathcal{T}$.

Example 5.5.3. (Example 4.2.1 continued) We turn automata $\mathcal{A}_2$ and $\mathcal{A}_3$ into component automata $\mathcal{C}_2$ and $\mathcal{C}_3$, respectively, by making a an output action of $\mathcal{C}_2$ and an input action of $\mathcal{C}_3$. The other elements of $\mathcal{C}_2$ and $\mathcal{C}_3$ are as in their underlying automata depicted in Figure 4.6(a). Then $\{\mathcal{C}_2, \mathcal{C}_3\}$ is a composable system and any team automaton $\mathcal{T}$ over $\{\mathcal{C}_2, \mathcal{C}_3\}$ has output alphabet $\{a\}$, while its input as well as its internal alphabet is empty.

Consequently, let $\mathcal{T}$ be the team automaton whose underlying synchronized automaton is depicted in Figure 4.6(b) once states (p, q, r) and (p, q, r') have been replaced by states (q, r) and (q, r'), respectively. Clearly $\mathcal{T}$ is $\{a\}$-enabling. It is however easy to see that $\mathcal{C}_3$ is not, even though all its output actions trivially (since there are none) are *ai* in $\mathcal{T}$. Moreover, the subteam $SUB_{\{3\}}$ of $\mathcal{T}$ is essentially a copy of $\mathcal{C}_3$ and is thus neither $\{a\}$-enabling. □

An additional condition is needed to extend Corollary 5.5.2 to the cases in which $alph \in \{out, loc\}$.

Corollary 5.5.4. *Let $alph \in \{out, loc\}$ and let $\mathcal{T}$ be Σ_{alph}-enabling. Then*

(1) if $\Sigma_{alph} \cap \Sigma_J \subseteq \Sigma_{J,alph}$ and $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Sigma_{J,alph}$, then SUB_J is Σ_{alph}-enabling, and

(2) if $\Sigma_{alph} \cap \Sigma_j \subseteq \Sigma_{j,alph}$ and $\delta_a \subseteq \mathcal{R}_a^{ai}(\mathcal{S})$, for all $a \in \Sigma_{j,alph}$, then $\mathcal{C}_j$ is Σ_{alph}-enabling. □

Determinism

In case the distribution of the alphabet plays no role, then the results concerning the inheritance of determinism from team automata to their subteams and component automata can obviously be lifted from Theorem 4.6.22.

Theorem 5.5.5. *Let $\mathcal{T}$ be Θ-deterministic and let $syn \in \{no, free, ai, si\}$. Then*

(1) if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma_J$, then SUB_J is Θ-deterministic, and

(2) if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$ and each a-transition of $\mathcal{C}_j$ is present in $\mathcal{T}$, for all $a \in \Theta \cap \Sigma_j$, then $\mathcal{C}_j$ is Θ-deterministic. □

Since $\Sigma_{alph} \cap \Sigma_J \subseteq \Sigma_{J,alph}$ and $\Sigma_{alph} \cap \Sigma_j \subseteq \Sigma_{j,alph}$, for $alph \in \{inp, int, ext\}$, the following result follows immediately.

Corollary 5.5.6. *Let $alph \in \{inp, int, ext\}$ and let $\mathcal{T}$ be Σ_{alph}-deterministic. Let $syn \in \{no, free, ai, si\}$. Then*

(1) if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Sigma_{J,alph}$, then SUB_J is Σ_{alph}-deterministic, and

(2) if $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$ and each a-transition of $\mathcal{C}_j$ is present in $\mathcal{T}$, for all $a \in \Sigma_{j,alph}$, then $\mathcal{C}_j$ is Σ_{alph}-deterministic. □

Note that this corollary does not cover the cases in which $alph \in \{out, loc\}$. In the following example we show that the fact that a team automaton $\mathcal{T}$ over $\mathcal{S}$ is Σ_{out}-deterministic in general does not imply that each of its constituting component automata is Σ_{out}-deterministic, not even if all its output actions are *maximal-free*, *maximal-ai*, or *maximal-si* in $\mathcal{T}$ and all component automaton transitions of output actions are present in $\mathcal{T}$. It is not difficult to provide a similar example for the case of subteams.

Example 5.5.7. (Example 4.6.5 continued) We turn automata $\mathcal{A}_1$ and $\mathcal{A}_2$ into component automata $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively, by making a an output action of $\mathcal{C}_1$ and an input action of $\mathcal{C}_2$. The other elements of $\mathcal{C}_1$ and $\mathcal{C}_2$ are as in their underlying automata depicted in Figure 4.11. Then $\{\mathcal{C}_1, \mathcal{C}_2\}$ is a composable system and any team automaton $\mathcal{T}$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$ has output alphabet $\{a\}$, while its input as well as its internal alphabet is empty.

Now let $\mathcal{T}$ be the team automaton with empty transition relation. Hence $\mathcal{T}$ is trivially $\{a\}$-deterministic. It is however clear that $\mathcal{C}_2$ is not, even though all its output actions trivially (since there are none) are *maximal-free*, *maximal-ai*, and *maximal-si* in $\mathcal{T}$ and all its transitions of output actions trivially (again, there are none) are present in $\mathcal{T}$. □

An additional condition is needed to extend Corollary 5.5.6 to the cases in which $alph \in \{out, loc\}$.

Corollary 5.5.8. *Let $alph \in \{out, loc\}$ and let $\mathcal{T}$ be Σ_{alph}-deterministic. Let $syn \in \{no, free, ai, si\}$. Then*

(1) if $\Sigma_{alph} \cap \Sigma_J \subseteq \Sigma_{J,alph}$ and $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Sigma_{J,alph}$, then SUB_J is Σ_{alph}-deterministic, and

(2) if $\Sigma_{alph} \cap \Sigma_J \subseteq \Sigma_{J,alph}$, $\delta_a = \mathcal{R}_a^{syn}(\mathcal{S})$, and each a-transition of $\mathcal{C}_j$ is present in $\mathcal{T}$, for all $a \in \Sigma_{j,alph}$, then $\mathcal{C}_j$ is Σ_{alph}-deterministic. □

5.5.2 Bottom-Up Inheritance of Properties

Dual to the above investigations we now change focus and study the sufficient conditions under which enabling and determinism are preserved from component automata from $\mathcal{S}$ to team automata over $\mathcal{S}$.

We recall from Section 5.2 that $\mathcal{T}$ is a team automaton over $\mathcal{S}'$ — upto a reordering — whenever $\mathcal{S}' = \{SUB_{\mathcal{I}_j} \mid \{\mathcal{I}_j \mid j \in \mathcal{J}\}$ forms a partition of $\mathcal{I}\}$. Hence it suffices to investigate the conditions under which the enabling and determinism of (component automata from) a composable system is preserved by a team automaton over that composable system.

Enabling

In case the distribution of the alphabet plays no role, then the results concerning the preservation of enabling from component automata from $\mathcal{S}$ to a team automaton over $\mathcal{S}$ can obviously be lifted from Theorem 4.6.33.

Theorem 5.5.9. *Let $\mathcal{C}_j$ be Θ-enabling. Then*

if each a-transition of $\mathcal{C}_j$, for all $a \in \Theta$, is omnipresent in $\mathcal{T}$, then $\mathcal{T}$ is $\Theta \cap \Sigma_j$-enabling. □

As the set of input (output, internal) actions of any team automaton $\mathcal{T}$ over $\mathcal{S}$ is included in the union of the sets of input (output, internal) actions of the component automata from $\mathcal{S}$, we immediately obtain the following result.

Corollary 5.5.10. *Let $alph \in \{inp, out, int, ext, loc\}$ and let $\mathcal{C}_i$ be $\Sigma_{i,alph}$-enabling, for all $i \in \mathcal{I}$. Then*

if all a-transitions of $\mathcal{C}_i$, for all $a \in \Sigma_{i,alph}$ and for all $i \in \mathcal{I}$, are omnipresent in $\mathcal{T}$, then $\mathcal{T}$ is Σ_{alph}-enabling. □

Note how, contrary to the results in the previous subsection, the possibility of an action being input to a component automaton from $\mathcal{S}$ and output to the team automata over $\mathcal{S}$ plays no role here. The reason is the fact that every input (output) action of a team automaton $\mathcal{T}$ over $\mathcal{S}$ needs to be an input (output) action of at least one component automaton from $\mathcal{S}$. Hence no additional condition is needed to cover the case in which $alph \in \{out, loc\}$ in this corollary. Even though an input action a of a non-$\{a\}$-enabling component automaton from $\mathcal{S}$ may be an output action of $\mathcal{T}$, it cannot prevent $\mathcal{T}$ from being Σ_{out}-enabling if the conditions of this corollary are satisfied. The reason is that according to these conditions, the component automaton from $\mathcal{S}$ in which a appears as an output action must not only be $\{a\}$-enabling, but all its a-transitions must moreover be omnipresent in $\mathcal{T}$.

Determinism

In case the distribution of the alphabet plays no role, then the results concerning the preservation of determinism from component automata from $\mathcal{S}$ to a team automaton over $\mathcal{S}$ can obviously be lifted from Theorem 4.6.35.

Theorem 5.5.11. *Let $\mathcal{S}$ be Θ-deterministic and let $syn \in \{ai, si\}$. Then*

if $\delta_a \subseteq \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Theta \cap \Sigma$, then $\mathcal{T}$ is Θ-deterministic. □

Since $\Sigma_{alph} \cap \Sigma_j \subseteq \Sigma_{j,alph}$, for $alph \in \{inp, int, ext\}$, the following result follows immediately.

Lemma 5.5.12. *Let $alph \in \{inp, int, ext\}$. Then*

if $\mathcal{C}_j$ is $\Sigma_{j,alph}$-deterministic, then $\mathcal{C}_j$ is Σ_{alph}-deterministic. □

Since an action may be input in a component automaton from $\mathcal{S}$ but output in a team automaton over $\mathcal{S}$, Lemma 5.5.12 cannot be extended to the cases in which $alph \in \{out, loc\}$. To see this, consider an external action a that is input to a component automaton (e.g. $\mathcal{C}_2$) which is $\Sigma_{2,out}$-deterministic but not $\{a\}$-deterministic, and output to another component automaton (e.g. $\mathcal{C}_1$). Then a will clearly be an output action of any team automaton over $\{\mathcal{C}_1, \mathcal{C}_2\}$, but $\mathcal{C}_2$ nevertheless is not $\{a\}$-deterministic.

Lemma 5.5.12 allows us to extend Theorem 5.5.11 to the cases in which $alph \in \{inp, int, ext\}$.

Corollary 5.5.13. *Let $alph \in \{inp, int, ext\}$ and let $\mathcal{C}_i$ be $\Sigma_{i,alph}$-deterministic, for all $i \in \mathcal{I}$. Let $syn \in \{ai, si\}$. Then*

if $\delta_a \subseteq \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Sigma_{alph}$, then $\mathcal{T}$ is Σ_{alph}-deterministic. □

Note that — contrary to Corollary 5.5.10 — Corollary 5.5.13 cannot be extended to the cases in which $alph \in \{out, loc\}$, not even when we consider team automata whose every action is *maximal-free*, *maximal-ai*, or *maximal-si*. This is because the transitions that cause a component automaton not to be deterministic are not a priori excluded from being present in such team automata, but when they are present they thus also cause those team automata not to be deterministic. In the following example we demonstrate this by showing that even if $\mathcal{C}_i$ is $\Sigma_{i,out}$-deterministic, for all $i \in \mathcal{I}$, and $\delta_a \subseteq \mathcal{R}_a^{syn}(\mathcal{S})$, for all $a \in \Sigma_{out}$ and $syn \in \{free, ai, si\}$, then this in general does not imply that $\mathcal{T}$ is Σ_{out}-deterministic.

Example 5.5.14. Let component automata $\mathcal{C}_1 = (\{q_1, q_1'\}, (\{a\}, \varnothing, \varnothing), \{(q_1, a, q_1), (q_1, a, q_1')\}, \{q_1\})$ and $\mathcal{C}_2 = (\{q_2, q_2'\}, (\varnothing, \{a\}, \varnothing), \{(q_2, a, q_2')\}, \{q_2\})$ be as depicted in Figure 5.13.

Fig. 5.13. Component automata $\mathcal{C}_1$ and $\mathcal{C}_2$.

Note that both $\mathcal{C}_i$, with $i \in [2]$, are $\Sigma_{i,out}$-deterministic. Furthermore, $\{\mathcal{C}_i \mid i \in [2]\}$ is a composable system. Now consider the team automaton $\mathcal{T} = (Q, (\varnothing, \{a\}, \varnothing), \delta, \{(q_1, q_2)\})$, where $Q = \{(q_1, q_2), (q_1, q_2'), (q_1', q_2), (q_1', q_2')\}$ and $\delta = \{((q_1, q_2), a, (q_1, q_2')), ((q_1, q_2), a, (q_1', q_2'))\}$, over this composable system. It is depicted in Figure 5.14(a).

Clearly $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S}) \subseteq \mathcal{R}_a^{si}(\mathcal{S})$, but $\mathcal{T}$ obviously is not Σ_{out}-deterministic.

Next consider the team automaton $\mathcal{T}' = (Q, (\varnothing, \{a\}, \varnothing), \delta', \{(q_1, q_2)\})$, where $\delta' = \{((q_1, q_2), a, (q_1, q_2)), ((q_1, q_2), a, (q_1', q_2))\}$, over this composable system. It is depicted in Figure 5.14(b). Clearly $\delta_a' \subseteq \mathcal{R}_a^{free}(\mathcal{S})$. However, also $\mathcal{T}'$ obviously is not Σ_{out}-deterministic. □

5.6 Inheritance of Synchronizations

In this section we start an initial exploration into the conditions under which the types of synchronization introduced in Sections 5.3 and 5.4 are inherited top-down — from team automata to subteams — and preserved bottom-up —

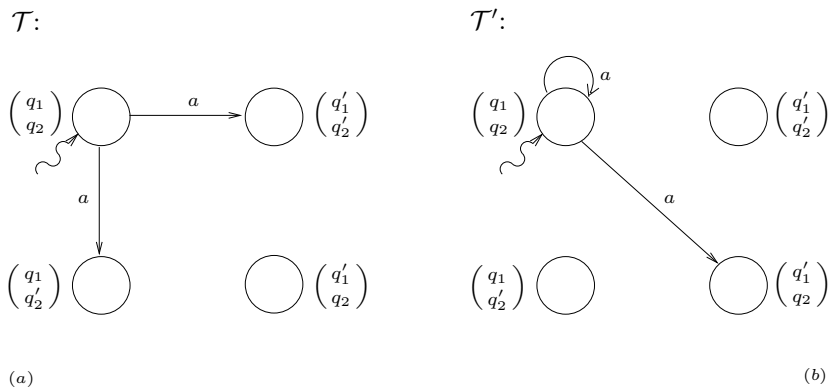

Fig. 5.14. Team automata $\mathcal{T}$ and $\mathcal{T}'$.

from subteams to team automata — as an addition to the results presented in Section 4.7 on the inheritance and preservation of the basic synchronizations *free*, *ai*, and *si*.

Since we deal with synchronizations *between* component automata constituting a team automaton, there is no need to study whether synchronizations are inherited by component automata from team automata — and vice versa: in any component automaton — and in any team automaton over a single component automaton — all its input (output) actions trivially are *sipp* and *wipp* (*sopp* and *wopp*) while all its output actions trivially are *ms*, *sms*, and *wms*.

We begin by considering the inheritance of the peer-to-peer types of synchronization. In the following example we show that if an action is *sipp* (*wipp*, *sopp*, *wopp*) in a team automaton, then this in general does not imply that it is also *sipp* (*wipp*, *sopp*, *wopp*) in each of its subteams.

Example 5.6.1. Consider the composable system $\{\mathcal{C}_1, \mathcal{C}_2\}$, which consists of component automata $\mathcal{C}_1 = (\{q_1, q_1'\}, (\{a\}, \varnothing, \varnothing), \{(q_1, a, q_1')\}, \{q_1\})$ and $\mathcal{C}_2 = (\{q_2, q_2'\}, (\varnothing, \{a\}, \varnothing), \{(q_2, a, q_2')\}, \{q_2\})$. It is depicted in Figure 5.15(a).

Now consider team automaton $\mathcal{T} = (\{(q_1, q_2), (q_1', q_2), (q_1, q_2'), (q_1', q_2')\}, (\varnothing, \{a\}, \varnothing), \{((q_1, q_2), a, (q_1', q_2'))\}, \{(q_1, q_2)\})$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$, depicted in Figure 5.15(b).

Clearly $\mathcal{I}_{a,inp}(\{\mathcal{C}_1, \mathcal{C}_2\}) = \{1\}$ and $\mathcal{I}_{a,out}(\{\mathcal{C}_1, \mathcal{C}_2\}) = \{2\}$. Thus a trivially is *ai* in both $SUB_{a,inp} = SUB_{\{1\}}$ and $SUB_{a,out} = SUB_{\{2\}}$. Hence a is *sipp* and *sopp* (and thus *wipp* and *wopp*) in $\mathcal{T}$.

We observe that $SUB_{\{2\}_{a,inp}(\{\mathcal{C}_2\})}(SUB_{\{2\}}) = (\varnothing, (\varnothing, \varnothing, \varnothing), \varnothing, \varnothing) = SUB_{\{1\}_{a,out}(\{\mathcal{C}_1\})}(SUB_{\{1\}})$. This implies $a \notin SI(SUB_{\{2\}_{a,inp}(\{\mathcal{C}_2\})}(SUB_{\{2\}}))$

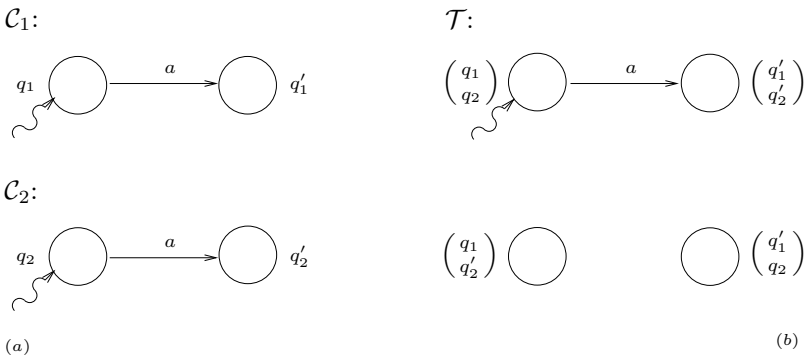

Fig. 5.15. Component automata $\mathcal{C}_1$ and $\mathcal{C}_2$, and team automaton $\mathcal{T}$.

and $a \notin SI(SUB_{\{1\}_{a,out}(\{\mathcal{C}_1\})}(SUB_{\{1\}}))$. Hence a is neither *wipp* nor *wopp* (and thus neither *sipp* nor *sopp*) in $SUB_{\{2\}}$ and $SUB_{\{1\}}$, respectively. □

From Lemma 4.7.1(2,3) we obtain that *sipp* and *wipp* (*sopp* and *wopp*) actions *are* inherited from a team automaton to a subteam as long as the subteam is chosen from the input (output) domain of the team automaton. Recall that $\Sigma_{out} = \bigcup_{i \in \mathcal{I}} \Sigma_{i,out}$.

Lemma 5.6.2. *Let $a \in \bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}$ and let $\varnothing \neq K \subseteq \mathcal{I}_{a,inp}(\mathcal{S})$. Then*

(1) if $a \in SIPP(\mathcal{T})$, then $a \in SIPP(SUB_K(\mathcal{T}))$, and

(2) if $a \in WIPP(\mathcal{T})$, then $a \in WIPP(SUB_K(\mathcal{T}))$.

Let $a \in \Sigma_{out}$ and let $\varnothing \neq L \subseteq \mathcal{I}_{a,out}(\mathcal{S})$. Then

(3) if $a \in SOPP(\mathcal{T})$, then $a \in SOPP(SUB_L(\mathcal{T}))$, and

(4) if $a \in WOPP(\mathcal{T})$, then $a \in WOPP(SUB_L(\mathcal{T}))$.

Proof. (1) From $a \in \Sigma_{inp}$ and $\varnothing \neq K \subseteq \mathcal{I}_{a,inp}(\mathcal{S})$ we know that the input domain of a in $\{\mathcal{C}_k \mid k \in K\}$ is K itself. Hence $K_{a,inp}(\{\mathcal{C}_k \mid k \in K\}) = K \neq \varnothing$. Now let a be *sipp* in $\mathcal{T}$. Then by Definition 5.3.4(1), a is *ai* in $SUB_{\mathcal{I}_{a,inp}(\mathcal{S})}(\mathcal{T})$. Since $K \subseteq \mathcal{I}_{a,inp}(\mathcal{S})$, Lemma 4.7.1(2) directly implies that a is *ai* in $SUB_K(SUB_{\mathcal{I}_{a,inp}(\mathcal{S})}(\mathcal{T})) = SUB_{K_{a,inp}(\{\mathcal{C}_k|k\in K\})}(SUB_K(\mathcal{T}))$. Definition 5.3.4(1) now implies that a is *sipp* in $SUB_K(\mathcal{T})$.

(2-4) Analogous. □

Next we wonder whether *sipp* and *wipp* (*sopp* and *wopp*) actions are preserved from subteams to team automata. In the following example we show that in general they are not.

Example 5.6.3. (Example 5.3.18 continued) Note that $a \notin WOPP(\mathcal{T}) \cup SOPP(\mathcal{T})$ since $a \notin SI(SUB_{a,out})$. However, it is easy to see that $a \in SOPP(SUB_{\{1\}}(\mathcal{T})) \subseteq WOPP(SUB_{\{1\}}(\mathcal{T}))$. It is not difficult to adjust this example in order to show that also *sipp* and *wipp* actions in general are not preserved from subteams to team automata. □

It turns out that *sipp* and *wipp* (*sopp* and *wopp*) actions *are* preserved from the input (output) subteam of a team automaton to the team automaton as a whole, which together with the previous lemma provides us with the following result.

Theorem 5.6.4. *Let* $a \in \bigcup_{i \in \mathcal{I}} \Sigma_{i,inp}$*, let* $K = \mathcal{I}_{a,inp}(\mathcal{S})$*, and let* $SUB_K(\mathcal{T}) = (Q_K, \Sigma_K, \delta_K, I_K)$*. Then*

(1) $a \in \Sigma_K \cap SIPP(\mathcal{T})$ *if and only if* $a \in SIPP(SUB_K(\mathcal{T}))$ *and*

(2) $a \in \Sigma_K \cap WIPP(\mathcal{T})$ *if and only if* $a \in WIPP(SUB_K(\mathcal{T}))$*.*

Let $a \in \Sigma_{out}$*, let* $L = \mathcal{I}_{a,out}(\mathcal{S})$*, and let* $SUB_L(\mathcal{T}) = (Q_L, \Sigma_L, \delta_L, I_L)$*. Then*

(3) $a \in \Sigma_L \cap SOPP(\mathcal{T})$ *if and only if* $a \in SOPP(SUB_L(\mathcal{T}))$ *and*

(4) $a \in \Sigma_L \cap WOPP(\mathcal{T})$ *if and only if* $a \in WOPP(SUB_L(\mathcal{T}))$*.*

Proof. (1) (Only if) Directly from Lemma 5.6.2(1).

(If) Let $a \in SIPP(SUB_K(\mathcal{T}))$. Then Definition 5.3.4(1) implies that $a \in \Sigma_K \cap AI(SUB_K(\mathcal{T})))$. Since $K = \mathcal{I}_{a,inp}(\mathcal{S})$ and $a \in \Sigma_K \cap AI(SUB_K(\mathcal{T}))$, Definition 5.3.4(1) implies that $a \in \Sigma_K \cap SIPP(\mathcal{T})$.

(2-4) Analogous. □

Finally, we turn to the master-slave types of synchronization. In the following example we show that if an action is *ms* (*sms*, *wms*) in a team automaton, then this in general does not imply that it is also *ms* (*sms*, *wms*) in each of its subteams.

Example 5.6.5. (Example 5.6.1 continued) Clearly a is *sms* (and thus also *ms* and *wms*) in $\mathcal{T}$. However, a is not an output action of $SUB_{\{1\}}$ and it thus cannot be *ms* (and hence neither *sms* nor *wms*) in $SUB_{\{1\}}$. □

We do have that every output action a of $\mathcal{T}$ is *ms* in any subteam of $\mathcal{T}$ determined by a subset of the output domain of a in $\mathcal{S}$.

Theorem 5.6.6. *If* $a \in \Sigma_{out}$ *and* $\varnothing \neq K \subseteq \mathcal{I}_{a,out}(\mathcal{S})$*, then* $a \in MS(SUB_K(\mathcal{T}))$*.*

Proof. Let $a \in \Sigma_{out}$ and let $\varnothing \neq K \subseteq \mathcal{I}_{a,out}(\mathcal{S})$. Clearly $a \in \Sigma_{K,out}$. In fact, the output domain $J = K_{a,out}(\{\mathcal{C}_k \mid k \in K\})$ of a in $\{\mathcal{C}_k \mid k \in K\}$ is K itself. Now let $(p,p') \in (\delta_K)_a = \text{proj}_K{}^{[2]}(\delta_a) \cap \Delta_a(\{\mathcal{C}_k \mid k \in K\})$. Then $\text{proj}_K{}^{[2]}(p,p') = (p,p')$ and it thus follows from the above that $\text{proj}_J{}^{[2]}((\delta_K)_a) = (\delta_K)_a = (\delta_J)_a$. Hence $a \in MS(SUB_K(\mathcal{T}))$. □

We also get that an *ms* action a from a team automaton over $\mathcal{S}$ is also *ms* in all subteams determined by a set that contains the output domain of a in $\mathcal{S}$.

Theorem 5.6.7. *If $a \in MS(\mathcal{T})$ and $K \supseteq \mathcal{I}_{a,out}(\mathcal{S})$, then $a \in MS(SUB_K(\mathcal{T}))$.*

Proof. Let $a \in MS(\mathcal{T})$ and let $K \supseteq \mathcal{I}_{a,out}(\mathcal{S})$. Clearly $a \in \Sigma_{out}$ and hence $\mathcal{I}_{a,out}(\mathcal{S}) \neq \varnothing$. Now let $(p,p') \in (\delta_K)_a$. Then there must exist $q,q' \in Q$ such that $(q,q') \in \delta_a$ and $\text{proj}_K{}^{[2]}(q,q') = (p,p') \in \Delta_a(\{\mathcal{C}_k \mid k \in K\})$. Since $a \in MS(\mathcal{T})$, there exists a $k \in \mathcal{I}_{a,out}(\mathcal{S}) \subseteq K$ such that $\text{proj}_k{}^{[2]}(q,q') = \text{proj}_k{}^{[2]}(p,p') \in \delta_{k,a}$. Because the output domain $J = K_{a,out}(\{\mathcal{C}_k \mid k \in K\})$ of a in $\{\mathcal{C}_k \mid k \in K\}$ is $\mathcal{I}_{a,out}(\mathcal{S})$ it follows that $\text{proj}_J{}^{[2]}(q,q') \in \Delta_a(\{\mathcal{C}_\ell \mid \ell \in J\})$ and thus $\text{proj}_J{}^{[2]}((\delta_K)_a) = (\delta_J)_a$. Hence $a \in MS(SUB_K(\mathcal{T}))$. □

Furthermore, as we show next, an *ms* action a is preserved from a subteam to the team automaton over $\mathcal{S}$ as a whole, provided that the subteam is determined by a set that contains the input domain of a in $\mathcal{S}$.

Theorem 5.6.8. *Let $a \in \Sigma_{out}$ and let $K \supseteq \mathcal{I}_{a,inp}(\mathcal{S})$. Then*

if $a \in MS(SUB_K(\mathcal{T}))$, then $a \in MS(\mathcal{T})$.

Proof. Let $J = \mathcal{I}_{a,out}(\mathcal{S})$. Note that $J \neq \varnothing$. Now let $(q,q') \in \delta_a$ and assume that $\text{proj}_J{}^{[2]}(q,q') \notin (\delta_J)_a$, which means that $\text{proj}_\ell{}^{[2]}(q,q') \notin \delta_{\ell,a}$, for all $\ell \in J$, i.e. only the input domain of a in $\mathcal{S}$ is involved in this transition. Consequently, $\text{proj}_K{}^{[2]}(q,q') \in \Delta_a(\{\mathcal{C}_k \mid k \in K\})$. Now suppose that $a \in MS(SUB_K(\mathcal{T}))$. Then $a \in \Sigma_{K,out}$ and thus $K \cap J \neq \varnothing$. Moreover, from a being *ms* in $SUB_K(\mathcal{T})$ it follows that there exists a $k \in K \cap J$ such that $\text{proj}_k{}^{[2]}(q,q') \in \delta_{k,a}$, a contradiction with the fact that $\text{proj}_J{}^{[2]}(q,q') \notin (\delta_J)_a$. Hence we have proven that $a \notin MS(\mathcal{T})$ implies $a \notin MS(SUB_K(\mathcal{T}))$. □

Finally, we note that whenever an output action a is *sms* (*wms*) in $\mathcal{T}$ and $J \subseteq \mathcal{I}_{a,out}(\mathcal{S})$, then a trivially is *sms* (*wms*) in $SUB_J(\mathcal{T})$ because the input domain of a in $\{\mathcal{C}_j \mid j \in J\}$ is empty.

This completes our initial exploration into the conditions under which the complex types of synchronization introduced in Section 5.3 are inherited from team automata to subteams, and vice versa.

We conclude this section with a result on the inheritance of the maximal types of synchronization introduced in Section 5.4. Using our knowledge from

earlier results of this section we extend the results presented in Theorem 4.7.5 to the case of peer-to-peer and master-slave types of synchronization.

Theorem 5.6.9. *Let* $a \in \bigcup_{i\in\mathcal{I}} \Sigma_{i,inp}$ *and let* $K \subseteq \mathcal{I}_{a,inp}(\mathcal{S})$. *Then*

(1) if $\delta_a = \mathcal{R}_a^{sipp}(\mathcal{S})$, *then* $(\delta_K)_a = \mathcal{R}_a^{sipp}(\{\mathcal{C}_k \mid k \in K\})$, *and*

(2) if $\delta_a = \mathcal{R}_a^{wipp}(\mathcal{S})$, *then* $(\delta_K)_a = \mathcal{R}_a^{wipp}(\{\mathcal{C}_k \mid k \in K\})$.

Let $a \in \Sigma_{out}$ *and let* $L \subseteq \mathcal{I}_{a,out}(\mathcal{S})$. *Then*

(3) if $\delta_a = \mathcal{R}_a^{sopp}(\mathcal{S})$, *then* $(\delta_L)_a = \mathcal{R}_a^{sopp}(\{\mathcal{C}_\ell \mid \ell \in L\})$,

(4) if $\delta_a = \mathcal{R}_a^{wopp}(\mathcal{S})$, *then* $(\delta_L)_a = \mathcal{R}_a^{wopp}(\{\mathcal{C}_\ell \mid \ell \in L\})$, *and*

(5) if $\delta_a = \mathcal{R}_a^{ms}(\mathcal{S})$, *then* $(\delta_L)_a = \mathcal{R}_a^{ms}(\{\mathcal{C}_\ell \mid \ell \in L\})$.

Proof. (1) By Lemma 5.6.2(1) we only need to prove that $\delta_a = \mathcal{R}_a^{sipp}(\mathcal{S})$ implies $\mathcal{R}_a^{sipp}(\{\mathcal{C}_k \mid k \in K\}) \subseteq (\delta_K)_a$. Hence let $(p,p') \in \mathcal{R}_a^{sipp}(\{\mathcal{C}_k \mid k \in K\})$. Then by Definition 5.4.4(1) there exists a $(q,q') \in \mathcal{R}_a^{sipp}(\mathcal{S})$ such that ${\rm proj}_K{}^{[2]}(q,q') = (p,p')$ and thus, since $\delta_a = \mathcal{R}_a^{sipp}(\mathcal{S})$, $(p,p') = {\rm proj}_K{}^{[2]}(q,q') \in (\delta_K)_a$.

(2-4) Analogous.

(5) Analogous, but now using Theorem 5.6.6 and Definition 5.4.7(1). □

5.7 Conclusion

Team automata can be classified on basis of the properties of their transition relations or by imposing conditions on their transition relations, which may lead to team automata that are maximal with respect to the given conditions. Furthermore, we can consider properties at the team level, or at the level of subteams.

Team automata allow exact descriptions of certain groupware notions which may otherwise have an ambiguous interpretation. Consider, e.g., the distinction between cooperation and collaboration within the team automaton model as described in [Ell97]:

"A Team Automaton is defined to be *cooperating* if it is structured so that one of its components is the active master, and all the others are passive slaves."

and

"A Team Automaton is defined to be *collaborating* if it is structured so that all of the automata are active peers."

To this it is added that the master-slave mechanism is referred to as *passive cooperation*, since the master is never blocked waiting for a slave. This contrasts with the peer-to-peer mechanism, in which blocking may occur when not all of the participants are ready to execute the action, and which is called *active collaboration*.

The framework of team automata clearly allows for more and finer distinctions. This is mainly due to the uniform approach towards the formalization of the notion of obligation for component automata to participate in the execution of a certain action, which is independent of the role of that action (input or output, peer, master, or slave).

We have thus provided two global interpretations of collaboration through the notions of *ai* (comparable to the adjective "active" above, as blocking may occur) and *si*. Here the input role an action may have is not yet separated from its output role. When this distinction is made we arrive at the four notions of strong (weak) input (output) peer-to-peer.

Cooperation, on the other hand, is formalized through the notions of (weak and strong) *ms* synchronizations. When an action is *ms*, then it cannot be executed as an input action without being simultaneously executed as an output action. In the strong case, all slaves (the component automata having the action as an input action) should participate in the action, whereas in the weak case all component automata that are ready for that action should participate in the synchronization (which corresponds to the "passive" cooperation mentioned above). Note that the master in an *ms* synchronization may be a subteam rather than a single component automaton. As argued in Section 5.2, there is no essential difference between a subteam of a team automaton and a component automaton which itself may have been obtained as a team automaton. Similarly, the slaves may be one or more component automata or one or more subteams.

The above viewpoint also easily allows combinations of cooperation and collaboration, called *hybrids* in [Ell97]. One may, e.g., have an *ms* synchronization in which within the master (subteam) the synchronizations are *sopp*, while the subteam of the slaves exhibits *wipp* synchronization (all slaves that can, participate) or *sipp* synchronization (all slaves have to take part).

Finally, observe that these considerations on cooperation and collaboration all relate to the synchronizations of a single external action. These notions can also be lifted to the level of the team automaton as a whole, either in a homogeneous way or in an heterogeneous way. In the first case there is one type of cooperation or collaboration (the same for all actions) including the identity of the master, the slaves, the input domain, the out-

put domain, etc. In the second case, each external action can have its own cooperation or collaboration specification.

Given requirements for each external action, one may follow the approach outlined in Section 5.4 to construct a unique team automaton with the appropriate combinations of cooperating and collaborating synchronizations.

The theory presented so far has thus led to a flexible framework that allows one to precisely classify, describe and construct many different incarnations of cooperation and collaboration. Which of these may be of use in applications, is for practice to decide.