

# Operating Guidelines for Finite-State Services<sup>\*</sup>

Niels Lohmann<sup>1</sup>, Peter Massuthe<sup>1</sup>, and Karsten Wolf<sup>2</sup>

<sup>1</sup> Humboldt-Universität zu Berlin, Institut für Informatik  
{nlohmann, massuthe}@informatik.hu-berlin.de

<sup>2</sup> Universität Rostock, Institut für Informatik  
karsten.wolf@informatik.uni-rostock.de

**Abstract.** We introduce the concept of an operating guideline for an arbitrary finite-state service  $P$ , extending the work of [1, 2] which was restricted to acyclic services.

An operating guideline gives complete information about how to correctly (in this paper: deadlock-free) communicate with  $P$ . It can further be executed or used for service discovery.

An operating guideline for  $P$  is a particular service  $S$  that is enriched with annotations.  $S$  communicates deadlock-free with  $P$  and is able to simulate every other service that communicates deadlock-free with  $P$ . The attached annotations give complete information about whether or not a simulated service is deadlock-free, too.

## 1 Introduction

In real life, we routinely use complicated electronic devices such as digital cameras, alarm clocks, mobile phones, CD players, vending machines, etc. Using such a device involves complex interaction, where information from the user to the device flows via pushing buttons or spinning wheels while information is passed from the device to the user via displays or blinking LED.

In some cases, we do not even abstractly know what is going on inside the device. Nevertheless, we are typically able to participate in the interaction. Besides ergonomic design, help from experienced friends, or trial-and-error exploration, it is often the user instructions which help us to figure out what to do at which stage. The typical features of user instructions (at least good ones) are:

- they are shipped with, or pinned to, the device,
- they are operational, i. e. a user can execute them step by step,
- they are complete, i. e. they cover the full intended functionality of the device,
- they use only terms related to the interface (buttons, displays, etc.) without trying to explain the internal processes.

In the virtual world, services [3] replace the devices of the real world. Still, using a service may require involved interaction with the user (which can be another service, like in service-oriented computing [4]). With the concept of an

---

<sup>\*</sup> Partially funded by the BMBF project “Tools4BPEL”.

*operating guideline*, we are going to propose an artifact that, in the virtual world, plays the role of user instructions in the real world. In particular, we will show that it exhibits the characteristics listed above. Moreover, we show that the operating guideline for a service can be automatically computed and be used for automatically checking compliance between services.

In contrast, a *public view* of a service (a condensed version of the service itself) has been proposed as an artifact for explaining the interaction with the service [5, 6]. Public views, however, do neither match the second nor the fourth item of the list.

In this paper, we extend our prior work [1, 2] and introduce an operating guideline for an arbitrary finite-state service  $P$  as a distinguished service  $S$  that properly interacts with  $P$ , together with annotations at each state of  $S$ . For this paper, we assume that “proper interaction” between services  $P$  and  $R$  means deadlock freedom of the system composed of  $P$  and  $R$ . We are well aware that there are other possibilities for defining “correct interaction”. Nevertheless, deadlock freedom will certainly be part of any such definition, so this paper can be seen as a step towards a more sophisticated solution. The annotations are used for deciding whether services other than  $S$  communicate deadlock-free with  $P$ .

The rest of the paper is structured as follows. In Sect. 2, we introduce a formal notion of services that emphasizes control and interaction. We thereby abstract from issues such as interface compatibility, data, and semantic compliance for which we refer to dedicated research efforts. Sections 3–7 are devoted to the construction of an operating guideline and its use. We start by defining, in Sect. 3, a concept that we call *situations*, which describes the coupling between a given service  $P$  and a partner of  $P$ . This concept is fundamental to our approach. Based on the concept of situations, we are able to characterize, in Sect. 4, deadlock freedom in a way that is suitable for subsequent considerations. The characterization can be translated into Boolean formulas which are used as annotations in the operating guideline later on. Section 5 provides important calculation procedures. They are used for the calculation of the particular partner  $S$  mentioned above. The calculation and justification of  $S$  is subject of Sect. 6. In Sect. 7, finally, we formalize the concept of an operating guideline and show how it can be used for identifying partners that communicate deadlock-free with  $P$ . Section 8 discusses issues of an implementation and presents experimental results.

## 2 Service Automata

For this paper, we make the following assumptions. First, we assume that a service has finitely many internal (control) states. In particular, we assume that data either do not play an important role, or have been abstracted to a reasonably small finite domain. Today, there exists satisfactory technology that supports this assumption [7].

Second, we assume that services communicate with each other via asynchronous message passing. We assume that messages cannot get lost but may

overtake each other. A service cannot see the internal state of another service. It cannot see the state of a message channel, except for the presence of incoming messages. As a technical restriction, we consider only such pairs of services where each channel contains, for some given and fixed  $k$ , at most  $k$  pending messages at any point of execution.

We model a service as an automaton where an edge label corresponds to a send or receive operation on an external message channel. Such an automaton could be modeled using I/O-automata [8]. That would, however, require to explicitly model the behavior of the message channels, as I/O automata make different assumptions about message passing. A service automaton can be easily retrieved from practical service specifications, for instance from the various formal operational semantics [9–15] of the emerging language BPEL [16].

Throughout the paper, we use the following notations for service automata. With  $P$  (from service provider), we denote an arbitrary service automaton for which we are going to calculate its operating guideline. With  $R$  (from service requester), we denote an arbitrary service in its role of a communication partner of  $P$ .  $S$  is used for the particular partner of  $P$  that forms the core of the operating guideline for  $P$ .

Furthermore, we fix a finite set  $C$ , the elements of which we call *channels*. We assume  $\tau \notin C$  (the symbol  $\tau$  is reserved for an internal move). With  $\text{bags}(C)$ , we denote the set of all multisets over  $C$ , that is, all mappings  $m : C \rightarrow \mathbb{N}$ . A multiset over  $C$  models a state of the channels, i.e. it represents, for each channel, the number of pending messages.  $[\ ]$  denotes the empty multiset ( $[\ ](x) = 0$  for all  $x$ ),  $[x]$  a singleton multiset ( $[x](x) = 1$ ,  $[x](y) = 0$  for  $y \neq x$ ), with  $m_1 + m_2$  the sum of two multisets ( $(m_1 + m_2)(x) = m_1(x) + m_2(x)$  for all  $x$ ), and with  $m_1 - m_2$  the difference ( $(m_1 - m_2)(x) = \max(m_1(x) - m_2(x), 0)$  for all  $x$ ).  $\text{bags}_k(C)$  denotes the set of all those multisets  $m$  over  $C$  where, for all  $x$ ,  $m(x) \leq k$ .  $\text{bags}_k(C)$  represents those states of the message channels that satisfy the requirement mentioned in the second paragraph of this section.

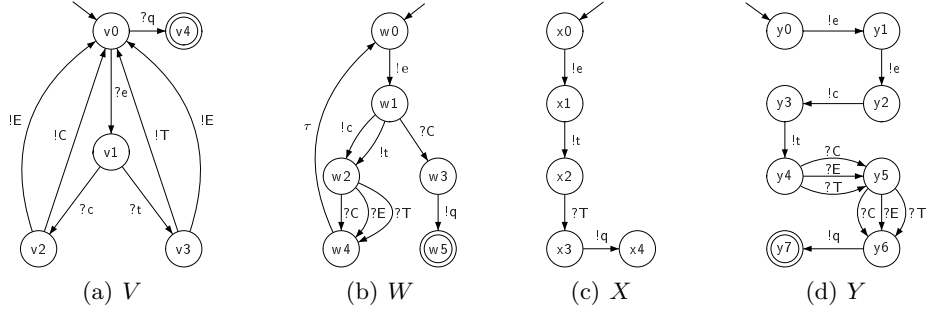
**Definition 1 (Service automaton).** A service automaton  $A = [Q, I, O, \delta, q_0, F]$  consists of a finite set  $Q$  of states, a set  $I \subseteq C$  of input channels, a set  $O \subseteq C$ ,  $I \cap O = \emptyset$  of output channels, a nondeterministic transition relation  $\delta \subseteq Q \times (I \cup O \cup \{\tau\}) \times Q$ , an initial state  $q_0 \in Q$ , and a set of final states  $F \subseteq Q$  such that  $q \in F$  and  $[q, x, q'] \in \delta$  implies  $x \in I$ .

Throughout this paper, in figures, we represent a channel  $x \in I$  with  $?x$  and a channel  $y \in O$  with  $!y$ . Figure 1 shows four examples of service automata.

We use indices to distinguish the constituents of different service automata.

**Definition 2 (Partner).** Two service automata  $P$  and  $R$  are partners if  $I_P = O_R$  and  $I_R = O_P$ .

Partners share channels in such a way that every channel represents a directed path of communication. In the sequel, we are mostly interested in partners.



**Fig. 1.** Examples of service automata. Service  $V$  models a vending machine where the user may insert a coin ( $e$ ) and choose coffee or tea ( $c, t$ ). The machine returns the corresponding beverage ( $C, T$ ) or rejects the coin ( $E$ ). In the initial state, the service may be switched off ( $q$ ). The services  $W, X,$  and  $Y$  model partners of  $V$ .

**Definition 3 (Composition of partners).** For partners  $P$  and  $R$ , their composition, denoted  $P \oplus R$ , is defined as an automaton with the following constituents:  $Q_{P \oplus R} = Q_P \times Q_R \times \text{bags}(C)$ ,  $I_{P \oplus R} = \emptyset$ ,  $O_{P \oplus R} = \emptyset$ ,  $q_{0P \oplus R} = [q_{0P}, q_{0R}, []]$ , and  $F_{P \oplus R} = F_P \times F_R \times \{[]\}$ . The transition relation  $\delta_{P \oplus R}$  contains the elements

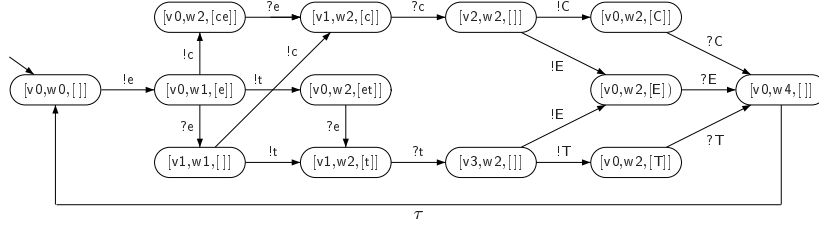
- (internal move in  $P$ )  $[[q_P, q_R, m], \tau, [q'_P, q_R, m]]$  iff  $[q_P, \tau, q'_P] \in \delta_P$ ,
- (internal move in  $R$ )  $[[q_P, q_R, m], \tau, [q_P, q'_R, m]]$  iff  $[q_R, \tau, q'_R] \in \delta_R$ ,
- (receive by  $P$ )  $[[q_P, q_R, m], \tau, [q'_P, q_R, m - [x]]]$  iff  $[q_P, x, q'_P] \in \delta_P$ ,  $x \in I_P$ , and  $m(x) > 0$ ,
- (receive by  $R$ )  $[[q_P, q_R, m], \tau, [q_P, q'_R, m - [x]]]$  iff  $[q_R, x, q'_R] \in \delta_R$ ,  $x \in I_R$ , and  $m(x) > 0$ ,
- (send by  $P$ )  $[[q_P, q_R, m], \tau, [q'_P, q_R, m + [x]]]$  iff  $[q_P, x, q'_P] \in \delta_P$  and  $x \in O_P$ ,
- (send by  $R$ )  $[[q_P, q_R, m], \tau, [q_P, q'_R, m + [x]]]$  iff  $[q_R, x, q'_R] \in \delta_R$  and  $x \in O_R$ , and no other elements.

The automaton  $P \oplus R$  has neither input nor output channels. It can thus be seen as a plain transition system. Note that the composition of two service automata can have infinitely many states for now.

Figure 2 depicts the composed service  $V \oplus W$  of the services  $V$  and  $W$  of Fig. 1. Note that  $V \oplus W$  has no (reachable) final states. Nevertheless,  $V \oplus W$  is deadlock-free, which is central in this paper.

**Definition 4 (Wait state, deadlock).** For an automaton  $A$ , a state  $q$  is called a wait state iff  $[q, x, q'] \in \delta$  implies  $x \in I$ , that is,  $q$  cannot be left without help from the environment. For a wait state  $q$ , let  $\text{wait}(q) = \{x \in C \mid \exists q' \in Q : [q, x, q'] \in \delta\}$ . A wait state  $q$  is called deadlock iff  $q \notin F$  and  $\text{wait}(q) = \emptyset$ .

A wait state cannot be left without an incoming message.  $\text{wait}(q)$  is the set of all incoming messages that would help to leave  $q$ . A deadlock cannot be



**Fig. 2.** Composition of services  $V$  and  $W$  of Fig. 1. Only the states reachable from the initial state are depicted.

left, independently from incoming messages. The definition of service automata requires final states to be wait states which is reasonable.

Examples for wait states in Fig. 1 are  $v0$  with  $wait(v0) = \{e, q\}$ ,  $w2$  with  $wait(w2) = \{C, E, T\}$ , or  $x4$  with  $wait(x4) = \emptyset$ . An example for a deadlock is the state  $[v0, x2, [E]]$  of the composition of the services  $V$  and  $X$  of Fig. 1 that can be reached from the initial state  $[v0, x0, []]$  of  $V \oplus X$  by executing first the transitions send  $e$  and send  $t$  of service  $X$ , followed by the transitions receive  $e$ , receive  $t$ , and send  $E$  of service  $V$ .

The initial requirement that communication channels must never contain more than  $k$  pending messages can be formalized as follows.

**Definition 5 ( $k$ -boundedness).** *If, for two services  $P$  and  $R$ ,  $Q_{P \oplus R} \subseteq Q_P \times Q_R \times bags_k(C)$ , then  $R$  is called a  $k$ -bounded partner of  $P$ .*

In Fig. 1,  $W$  and  $X$  are 1-bounded partners of  $V$ .  $Y$  is no 1-bounded partner since  $V \oplus Y$  contains, for instance, the state  $[v0, y2, [ee]]$ .  $Y$  is, however, a 2-bounded partner of  $V$ .

If  $P$  and  $R$  are  $k$ -bounded partners, then  $P \oplus R$  is finite-state. Then,  $P \oplus R$  is a well-defined service automaton itself.

### 3 Situations and Knowledge

In this section, we introduce concepts that help us understanding the coupling between two service  $P$  and  $R$ , from the point of view of  $R$ . To this end, consider the following mapping  $K$ .

**Definition 6 ( $K$ , situation).** *Let  $P$  and  $R$  be partners. Then, let  $K : Q_R \rightarrow 2^{Q_P \times bags(C)}$  be defined by  $K(q_R) = \{[q_P, m] \mid [q_P, q_R, m] \in Q_{P \oplus R}\}$ . The elements of  $2^{Q_P \times bags(C)}$  are called situations.*

A situation comprises all parts of a state of  $P \oplus R$  beyond the state of  $R$  itself. It can thus be dealt with independently of  $R$ .  $K(q_R)$  can be interpreted as the *knowledge* that  $R$  has about the possible situations that can occur in  $P \oplus R$  together with  $q_R$ .

We give some examples for values of  $K$ , referring to Fig. 1. We consider  $W$  as a partner of  $V$ . Then Fig. 2 tells us that  $K(w0) = \{[v0, []]\}$ ,  $K(w1) = \{[v0, [e]], [v1, []]\}$ ,  $K(w2) = \{[v0, [ce]], [v0, [et]], [v1, [c]], [v1, [t]], [v2, []], [v3, []], [v0, [C]], [v0, [E]], [v0, [T]]\}$ ,  $K(w3) = \emptyset$ , and  $K(w4) = \{[v0, []]\}$ .

Within a set  $M$  of situations, we distinguish transient and stable situations. A situation is transient in  $M$  if a move of  $P$  in that situation leads to another situation, also contained in  $M$ . Otherwise it is stable.

**Definition 7 (Transient, stable situation).** *Let  $M$  be a set of situations.*

*Within  $M$ ,  $[q_P, m]$  is transient iff there is an  $[q_P, x, q'_P] \in \delta_P$  such that:*

- $x = \tau$  and  $[q'_P, m] \in M$ , or
- $x \in I_P$ ,  $m(x) > 0$ , and  $[q'_P, m - [x]] \in M$ , or
- $x \in O_P$  and  $[q'_P, m + [x]] \in M$ .

*Otherwise,  $[q_P, m]$  is stable.*

For example, situation  $[v0, [e]]$  is transient in  $K(w1)$  (cf. Fig. 2) while  $[v1, []]$  is stable in  $K(w1)$ .

## 4 A Characterization of Deadlocks

With the vocabulary defined in the previous section, a deadlock in the composed system  $P \oplus R$ , seen from the point of view of  $R$ , reads as follows.

**Lemma 1.**  *$[q_P, q_R, m]$  is a deadlock of  $P \oplus R$  if and only if all of the following conditions hold:*

- $q_P \notin F_P$ , or  $q_R \notin F_R$ , or  $m \neq []$ ;
- $q_R$  is a wait state of  $R$ ;
- $[q_P, m]$  is stable in  $K(q_R)$  and, for all  $x \in \text{wait}(q_R)$ ,  $m(x) = 0$ .

*Proof.* Proofs can be found in the Appendix of this paper.

Consider again the example deadlock  $[v0, x2, [E]]$  in  $V \oplus X$  of the services in Fig. 1. Obviously,  $[E] \neq []$  and  $x2$  is a wait state of  $X$  with  $K(x2) = \{[v0, [et]], [v1, [t]], [v3, []], [v0, [E]], [v0, [T]]\}$ . The situation  $[v0, [E]]$  is stable in  $K(x2)$  and  $\text{wait}(x2) = \{T\}$ .

The three requirements of Lemma 1 can be easily compiled into a Boolean formula  $\phi(q_R)$  that expresses the absence of deadlocks of the shape  $[\cdot, q_R, \cdot]$  in  $P \oplus R$ . This formula uses the set of propositions  $C \cup \{\tau, \text{final}\}$  (with  $\text{final} \notin C$ ). Propositions in  $C \cup \{\tau\}$  represent labels of transitions that leave  $q_R$ , whereas proposition  $\text{final}$  represents the fact whether  $q_R \in F_R$ .

**Definition 8 (Annotation, R-assignment).** *Let  $P$  and  $R$  be partners. Then, for each  $q_R \in Q_R$ , define the annotation of  $q_R$ ,  $\phi(q_R)$  as the Boolean formula over the propositions  $C \cup \{\tau, \text{final}\}$  as follows.*

$$\phi(q_R) = \bigwedge_{[q_P, m] \text{ stable in } K(q_R)} (\phi_1(q_P, q_R, m) \vee \phi_2 \vee \phi_3(q_P, q_R, m))$$

where

- $\phi_1(q_P, q_R, m) = \begin{cases} \text{final}, & \text{if } q_P \in F_P \text{ and } m = [], \\ \text{false}, & \text{otherwise,} \end{cases}$
- $\phi_2 = \tau \vee \bigvee_{x \in I_P} x,$
- $\phi_3(q_P, q_R, m) = \bigvee_{x \in O_P, m(x) > 0} x.$

The  $R$ -assignment  $ass_R(q_R)$  assigns *true* to proposition  $x \in C \cup \{\tau\}$  if and only if there is a  $q'_R$  such that  $[q_R, x, q'_R] \in \delta_R$  and *true* to *final* if and only if  $q_R \in F_R$ .

Since the formulas  $\phi(q_R)$  exactly reflect the conditions of Lemma 1, we obtain:

**Corollary 1.**  $P \oplus R$  is deadlock-free if and only if, for all  $q_R \in Q_R$ , the value of  $\phi(q_R)$  with the  $R$ -assignment  $ass_R(q_R)$  is *true*.

In Fig. 1, the annotation of state  $w1$  would be  $c \vee e \vee t \vee \tau$ , due to the single stable situation  $[v1, []] \in K(w1)$ . This formula is satisfied by the  $R$ -assignment of  $w1$  that assigns *true* to both  $c$  and  $t$ , and *false* to *final* and  $\tau$ . The annotation of  $w2$  is  $(C \wedge E \wedge T) \vee c \vee e \vee t \vee \tau$  since  $K(w2)$  contains the three stable situations  $[v0, [C]]$ ,  $[v0, [E]]$ , and  $[v0, [T]]$ . Since the  $R$ -assignment of  $w2$  assigns *true* to all of  $C$ ,  $E$ , and  $T$ , it satisfies the annotation. For state  $x2$ , the annotation is  $(T \wedge E) \vee c \vee e \vee t \vee \tau$ . Since the only transition leaving  $x2$  is  $T$ , the  $R$ -assignment of  $x2$  assigns *false* to all propositions except  $T$ , and the annotation yields *false*. This corresponds to the deadlock  $[v0, x2, [E]]$  in  $V \oplus X$ .

## 5 Operations on Sets of Situations

For  $k$ -bounded partners, all reachable situations are actually in  $2^{Q_P \times bags_k(C)}$  which is a finite domain. For sets of situations, define the following operations.

**Definition 9 (Closure).** For a set  $M$  of situations, let the closure of  $M$ , denoted  $cl(M)$ , be inductively defined as follows.

- Base:  $M \subseteq cl(M)$ , Step: If  $[q_P, m] \in cl(M)$  and  $[q_P, x, q'_P] \in \delta_P$ , then
  - $[q'_P, m] \in cl(M)$ , if  $x = \tau$ ,
  - $[q'_P, m + [x]] \in cl(M)$ , if  $x \in O_P$ ,
  - $[q'_P, m - [x]] \in cl(M)$ , if  $x \in I_P$  and  $m(x) > 0$ .

It can be easily seen that  $cl(M)$  comprises those situations that can be reached from a situation in  $M$  without interference from a partner. In Fig. 1 for example, we obtain  $cl(\{[v0, [ce]]\}) = \{[v0, [ce]], [v1, [c]], [v2, []], [v0, [C]], [v0, [E]]\}$ .

**Definition 10 (Send-event, receive-event, internal-event).** Let  $M \subseteq Q_P \times bags(C)$ . If  $x \in O_P$ , then the send-event  $x$ ,  $send(M, x)$ , is defined as  $send(M, x) = \{[q, m + [x]] \mid [q, m] \in M\}$ . If  $x \in I_P$ , then the receive-event  $x$ ,  $receive(M, x)$ , is defined as  $receive(M, x) = \{[q, m - [x]] \mid [q, m] \in M, m(x) > 0\}$ . The internal-event  $\tau$ ,  $internal(M, \tau)$ , is defined as  $internal(M, \tau) = M$ . As the shape of an event is clear from  $I_P$  and  $O_P$ , we define the event  $x$ ,  $event(M, x)$ , as  $receive(M, x)$  if  $x \in I_P$ ,  $send(M, x)$  if  $x \in O_P$ , and  $internal(M, x)$  if  $x = \tau$ .

A send-event models the effect that a message sent by  $R$  has on a set of situations  $M$ . A receive-event models the effect that a message received by  $R$  has on a set of situations  $M$ .

Considering the service  $V$  of Fig. 1, we get, for example,

$$\begin{aligned} \text{send}(\{\{v0, [e], [v1, []]\}, c) &= \{\{v0, [ce], [v1, c]\}\} \text{ and} \\ \text{receive}(\{\{v0, [ce], [v1, [c]], [v2, []], [v0, [C]], [v0, [E]]\}, C) &= \{\{v0, []\}\}. \end{aligned}$$

## 6 A Canonical Partner

Using the concepts defined so far, we are now ready to construct a partner  $S$  for a given service  $P$ . The construction is based on the following considerations. A state of  $S$  is a set of situations. States and transitions are organized such that, for all states  $q$  of  $S$ ,  $K(q) = q$ . That is, every state is defined by the set of situations it can occur with. Transitions can be determined using the operations *event* and *cl*. Starting with a service  $S_0$  with *all* such states and transitions,  $S_0 \oplus P$  may contain deadlocks. However, these deadlocks can be identified in  $S_0$  using the annotations of Def. 8. Removing all states  $q$  where the annotation  $\phi(q)$  evaluates to *false*, yields a new service  $S_1$ . This procedure is iterated until either the remaining set of states is empty, or all annotations evaluate to *true*. In the latter case, the remaining service is, by construction of the annotations, a partner that has a deadlock-free composition with  $P$ .

**Definition 11 (Canonical partner  $S$ ).** *Let  $P$  be a service automaton. Define inductively a sequence  $S_i = [Q_i, I_i, O_i, \delta_i, q_{0i}, F_i]$  of service automata as follows.*

*Let  $Q_0 = 2^{Q_P \times \text{bags}_k(C)}$ . Let, for all  $i$ ,  $I_i = O_P$ ,  $O_i = I_P$ ,  $[q, x, q'] \in \delta_i$  iff  $q, q' \in Q_i$  and  $q' = \text{cl}(\text{event}(q, x))$ ,  $q_{0i} = \text{cl}(\{[q_{0P}, []]\})$ , and  $F_i = \{q \in Q_i \mid q \text{ is wait state of } S_i\}$ . Let, for all  $i$ ,  $Q_{i+1} = \{q \mid q \in Q_i, \phi(q) \text{ evaluates to true with assignment } \text{ass}_{S_i}(q)\}$ .*

*Let  $S$  be equal to  $S_i$  for the smallest  $i$  satisfying  $S_i = S_{i+1}$ .*

As the sequence  $\{S_i\}_{i=0,1,\dots}$  is monotonously decreasing, all objects of this definition are well-defined. The resulting  $S$  is a service automaton if and only if  $q_{0S} \in Q_S$ . In that case,  $S$  is in fact a partner of  $P$ .

Figure 3 shows the service  $S_0$  for the service  $V$  of Fig. 1. In this figure, only states reachable from the initial state are depicted.  $S_0$  is constructed for  $k = 1$ . Figure 6 shows a service that is isomorphic to the resulting service  $S$ .

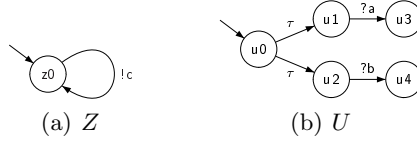
With the next few results, we further justify the construction.

**Lemma 2.** *If  $\text{cl}([q_{0P}, []]) \notin Q_P \times \text{bags}_k(C)$ , then  $P$  does not have  $k$ -bounded partners.*

This lemma states that  $S_0$  is well-defined for all interesting  $P$ . Thereby,  $P$  is interesting if it has at least one partner  $R$  such that  $P \oplus R$  is deadlock-free. To obtain a well-defined service, the initial state must be contained in the set of states.

The next lemma shows that we actually achieved one of the major goals of the construction.



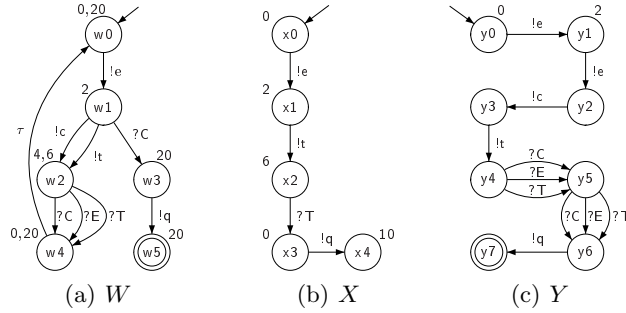


**Fig. 4.** Services that do not have  $k$ -bounded partners ( $Z$ ) or cannot communicate deadlock-free with any partner ( $U$ ).

For further studying the constructed partner  $S$ , we establish a relation between states of an arbitrary partner  $R$  of  $P$ , and states of  $S_0$ , the starting point for the construction of  $S$ .

**Definition 12 (Matching).** Let  $R_1$  and  $R_2$  be service automata and define the relation  $L_{R_1, R_2} \subseteq Q_{R_1} \times Q_{R_2}$ , the matching between  $R_1$  and  $R_2$ , inductively as follows. Let  $[q_0, q_0] \in L$ . If  $[q_1, q_2] \in L_{R_1, R_2}$ ,  $[q_1, x, q'_1] \in \delta_{R_1}$  and  $[q_2, x, q'_2] \in \delta_{R_2}$ , then let  $[q'_1, q'_2] \in L_{R_1, R_2}$ .

Examples for matchings are shown in Fig. 5.



**Fig. 5.** Matching of the three services  $W$ ,  $X$ , and  $Y$  of Fig. 1 with the service  $S_0$  depicted in Fig. 3. A number  $n$  attached to a state  $x$  represents a pair  $[x, n] \in L$ .

**Lemma 4.** Let  $S_0$  be the starting point of the construction in Def. 11 and  $R$  be an arbitrary partner of  $P$ . For all  $q_R \in Q_R$ ,  $K(q_R) = \bigcup_{[q_R, q_S] \in L_{R, S_0}} K(q_S)$ .

For example, state  $w_2$  of service  $W$  in Fig. 1 is matched with states 4 and 6 of the service  $S_0$  in Fig. 3. As demonstrated earlier,  $K(w_2)$  contains exactly the situations that occur in states 4 or 6 of the service in Fig. 3.

**Corollary 3.** For each state  $q_R$  of  $R$ , its annotation  $\phi(q_R)$  can be described as  $\phi(q_R) \equiv \bigwedge_{[q_S: [q_R, q_S] \in L_{R, S_0}} \phi(q_S)$ .

For example, the annotation of state w2 of the service in Fig. 1 is  $(C \wedge E \wedge T) \vee c \vee e \vee t \vee \tau$  which is equivalent to the conjunction of the annotations  $(C \wedge E) \vee \tau$  and  $(E \wedge T) \vee c \vee e \vee t \vee \tau$  to the states 4 and 6 of the service in Fig. 3.

The next result is the actual justification of the removal process described in Def. 11.

**Lemma 5.** *If  $R$  is a  $k$ -bounded partner of  $P$  such that  $P \oplus R$  is deadlock-free then, for all  $[q_R, q_S] \in L_{R, S_0}$ ,  $q_S \in S$ .*

**Corollary 4.**  *$P$  has a partner  $R$  such that  $P \oplus R$  is deadlock-free if and only if  $q_{0S} \in Q_S$  (i.e. the service-automaton  $S$  is well-defined).*

If  $P$  does not have partners  $R$  such that  $P \oplus R$  is deadlock-free, then  $P$  is fundamentally ill-designed. Otherwise, the particular partner  $S$  studied above is well-defined. It is the basis for the concept of an operating guideline for  $P$  which is introduced in the next section.

## 7 Operating Guideline

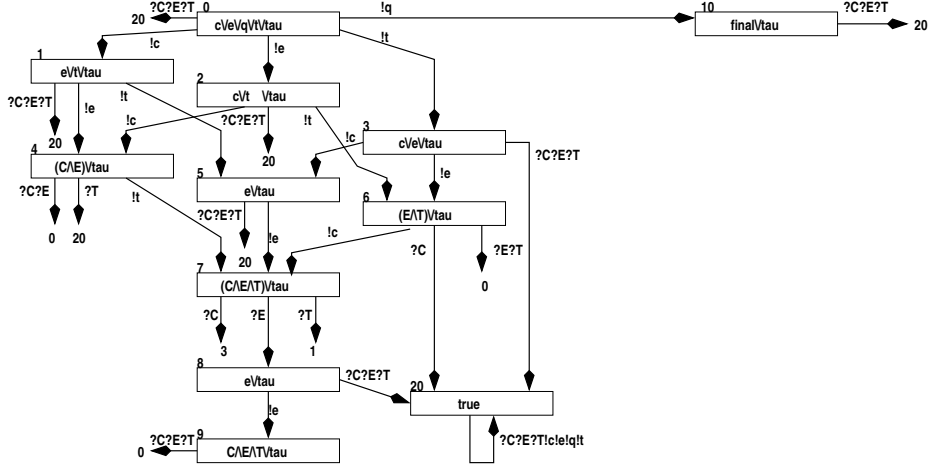
If the matching of a service  $R$  with  $S_0$  involves states of  $Q_{S_0} \setminus Q_S$ , Lemma 5 asserts that  $P \oplus R$  has deadlocks. In the case that the matching involves only states of  $Q_S$ ,  $P \oplus R$  may or may not have deadlocks. However, by Cor. 1, the existence of deadlocks in  $P \oplus R$  can be decided by evaluating the annotations  $\phi(q_R)$  for the states  $q_R \in Q_R$ . By Cor. 3, these formulas can be retrieved from the annotations to the states of  $S$ . Attaching these formulas explicitly to the states of  $S$ , the whole process of matching and constructing the  $\phi(q_R)$  can be executed without knowing the actual contents of the states of  $S$ , i.e. without knowing the situations — the topology of  $S$  is sufficient. This observation leads us to the concept of an operating guideline for  $P$ .

**Definition 13 (Operating guideline).** *Let  $P$  be a service automaton which has at least one partner  $R$  such that  $P \oplus R$  is deadlock-free. Then any automaton  $S^*$  that is isomorphic to  $S$  (which is the well-defined service automaton  $S$  of Def. 11), together with a mapping  $\Phi$  with  $\Phi(q_{S^*}) = \phi(q_S)$  for  $q_S^*$  being isomorphic to  $q_S$ , is called operating guideline for  $P$ .*

With the step from  $S$  to an isomorphic  $S^*$ , we just want to emphasize that only the topology of  $S$  is relevant in the operating guideline while the internal structure of states of  $S$  is irrelevant.

Figure 6 shows the operating guideline for the service  $V$  of Fig. 1.

Ignoring  $\Phi$ , the operating guideline is a partner  $S$  for  $P$  that can be directly executed thus satisfying the second requirement stated in the introduction. The annotations  $\Phi(q)$  give additional instructions about whether or not transitions leaving  $q$  may be skipped. The operating guideline can be used to decide for an arbitrary service  $R$  whether or not  $P \oplus R$  is deadlock-free, as the next result shows.



**Fig. 6.** Operating guideline for the service  $V$  in Fig. 1. Annotations have been simplified by removing those output channels that are not present in outgoing transitions. They get value *false* for every partner that satisfies requirement (topology).

**Theorem 1 (Main theorem of this article).**  $R$  is a partner of  $P$  such that  $P \oplus R$  is deadlock-free if and only if the following requirements hold for every  $[q_R, q_{S^*}] \in L_{R, S^*}$  :

**(topology)** For every  $x \in C \cup \{\tau\}$ , if there is an  $x$ -transition leaving  $q_R$  in  $R$ , then there is an  $x$ -transition leaving  $q_{S^*}$  in  $S^*$ .

**(annotation)** The assignment  $ass_R(q_R)$  satisfies  $\Phi(q_{S^*})$ .

Note that this theorem matches  $R$  with  $S^*$  (isomorphic to  $S$ ) while the results in the previous section match  $R$  with  $S_0$ . Requirement (topology) actually states that  $L_{R, S^*}$  is a simulation relation.

Consider the service  $V$  of Fig. 1 and its partners. From Fig. 5, we can see that  $W$  and  $X$  satisfy the requirement (topology) while  $Y$  does not ( $Y$  is not a 1-bounded partner of  $V$ ).  $X$  violates in state  $x2$  the annotation to the matched state 6, since the  $R$ -assignment in state  $x2$  assigns *false* to  $E$  and  $\tau$ .  $V \oplus X$  contains the deadlock  $[v0, x2, [E]]$ . For service  $W$ , all annotations are satisfied.  $V \oplus W$  is deadlock-free (see Fig. 2).

## 8 Implementation

All concepts used in this article have been defined constructively. For an actual implementation, it is, however, useful to add some ideas that increase efficiency. First, it is easy to see that, for constructing  $S$ , it is not necessary to start with the whole  $S_0$ . For the matching, only states that are reachable from the initial state, need to be considered. Furthermore, annotations can be generated as soon as a state is calculated. They can be evaluated as soon as the immediate

successors have been encountered. If the annotation evaluates to *false*, further exploration can be stopped [17]. In consequence, the process of generating  $S_0$  can be interleaved with the process of removing states that finally lead to  $S$ . This way, memory consumption can be kept within reasonable bounds.

The content of a state  $q$  of  $S$  can be reduced using, for instance, partial order reduction techniques. In ongoing research, we explore that possibility. We are further exploring opportunities for a compact representation of an operating guideline. For this purpose, we already developed a *binary decision diagram* (BDD, [18]) representation of an operating guideline for acyclic services that can be efficiently used for matching [19]. Most likely, these concepts can be adapted to arbitrary finite-state services.

We prototypically implemented our approach within the tool Fiona [20]. Among other features, Fiona can read an *open workflow net* [21, 2], that is a Petri net model of a service, and generate the operating guideline. The state space of that Petri net (built by ignoring the channels) can be interpreted as a service automaton as studied in this paper. The example Petri nets used below stem from specifications written in the language BPEL [16]. The BPEL processes have been translated automatically into Petri nets, based on the Petri net semantics for BPEL [15] and the tool BPEL2oWFN [20].

To measure the performance of our implementation, we calculated the operating guideline for several services of different domains.

The “Purchase Order” and “Loan Approval” processes are realistic services taken from the BPEL specification draft [16]. “Olive Oil Ordering” [22], “Help Desk Service Request” (from the Oracle BPEL Process Manager) and “Travel Service” [23] are other web services that use BPEL features like fault and event handling. The “Database Service” [24] shows that it may be necessary to calculate a number of situations which is a multiple of the number of states of the considered service automaton.

“Identity Card Issue” and “Registration Office” are models of administrative workflows provided by Gedilan, a German consulting company. Finally, we modeled parts of the Simple Mail Transfer Protocol (SMTP) [25]. Since it is a communication protocol, it yields the biggest operating guideline.

Table 1 provides the number of states of the corresponding service automaton (which is derived from the intermediate Petri net), the size (number of situations, states, and edges) of the calculated portion of  $S_0$ , the size (states and edges) of the operating guideline, and the time for its calculation from the given Petri net.

## 9 Conclusion

With the concept of an operating guideline for a service  $P$ , we proposed an artifact that can be directly executed, is expressed in terms of the interface of  $P$ , and gives complete information about deadlock-free communication with  $P$ . It can be manipulated in accordance with the annotations. This way, other partners can be crafted which, by construction, communicate deadlock-free with  $P$ , too.

**Table 1.** Experimental results running Fiona. All experiments were taken on a Intel Pentium M processor with 1.6 GHz and 1 GB RAM running Windows XP.

service $P$	$P$		$S_0$			operating guideline		time (seconds)
	states	situations	states	edges	states	edges		
Purchase Order	90	464	169	794	168	548	< 1	
Loan Approval	50	199	27	75	7	8	< 1	
Olive Oil Ordering	15	5101	1346	6413	40	69	< 1	
Help Desk Service	25	7765	1446	5678	8	10	< 1	
Travel Service	1879	5696	321	2149	320	1120	3	
Database Service	5232	337040	55	179	54	147	178	
Identity Card Issue	111842	707396	433	2869	280	1028	220	
Registration Office	7265	9049	21	62	7	8	< 1	
SMTP	19653	304284	28209	169297	392	1470	59	

Deciding deadlock freedom using an operating guideline amounts to checking the simulation relation between the partner service and the operating guideline, and evaluating the annotations. It has about the same complexity as model checking deadlock freedom in the composed system itself. Due to its completeness, and due to its explicit operational structure, it can be a valuable tool in service-oriented architectures.

We showed that the calculation of an operating guideline is feasible in practical applications.

## References

1. Massuthe, P., Schmidt, K.: Operating guidelines - An automata-theoretic foundation for the service-oriented architecture. In Cai, K.Y., Ohnishi, A., Lau, M., eds.: Proceedings of the Fifth International Conference on Quality Software (QSIC 2005), Melbourne, Australia, IEEE Computer Society (2005) 452–457
2. Massuthe, P., Reisig, W., Schmidt, K.: An Operating guideline approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics* **1**(3) (2005) 35–43
3. Hull, R., Benedikt, M., Christophides, V., Su, J.: E-services: A look behind the curtain. In: PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, New York, NY, USA, ACM Press (2003) 1–14
4. Papazoglou, M.P.: Agent-oriented technology in support of e-business. *Communications of the ACM* **44**(4) (2001) 71–77
5. Aalst, W., Weske, M.: The P2P approach to interorganizational workflows. In: Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01). Volume 2068 of LNCS., Springer-Verlag, Berlin (2001) 140–156
6. Leymann, F., Roller, D., Schmidt, M.: Web services and business process management. *IBM Systems Journal* **41**(2) (2002)
7. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. 2nd edn. Springer-Verlag (2005)
8. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann (1996)

9. Arias-Fisteus, J., Fernández, L.S., Kloos, C.D.: Formal Verification of BPEL4WS Business Collaborations. In Bauknecht, K., Bichler, M., Pröll, B., eds.: EC-Web. Volume 3182 of LNCS., Springer (2004) 76–85
10. Fu, X., Bultan, T., Su, J.: Analysis of interacting BPEL web services. In: WWW '04: Proceedings of the 13th international conference on World Wide Web, ACM Press (2004) 621–630
11. Ferrara, A.: Web services: a process algebra approach. In: ICSOC, ACM (2004) 242–251
12. Fahland, D., Reisig, W.: ASM-based semantics for BPEL: The negative Control Flow. In Beauquier, D., Börger, E., Slissenko, A., eds.: Proceedings of the 12th International Workshop on Abstract State Machines (ASM'05), Paris XII (2005) 131–151
13. Farahbod, R., Glässer, U., Vajihollahi, M.: Specification and Validation of the Business Process Execution Language for Web Services. In: Abstract State Machines. Volume 3052 of Lecture Notes in Computer Science., Springer (2004) 78–94
14. Ouyang, C., Verbeek, E., van der Aalst, W.M., Breutel, S., Dumas, M., ter Hofstede, A.H.: Formal Semantics and Analysis of Control Flow in WS-BPEL. Technical report (revised version), Queensland University of Technology (2005)
15. Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to Petri nets. In van der Aalst, W., Benatallah, B., Casati, F., Curbera, F., eds.: Proceedings of the Third International Conference on Business Process Management (BPM 2005). Volume 3649 of LNCS., Nancy, France, Springer-Verlag (2005) 220–235
16. Alves, A., Arkin, A., Askary, S., Bloch, B., Curbera, F., Ford, M., Golland, Y., Guzar, A., Kartha, N., Liu, C.K., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., Yiu, A.: Web Services Business Process Execution Language Version 2.0. Public review draft, 23rd august, 2006, Organization for the Advancement of Structured Information Standards (OASIS) (2006)
17. Weinberg, D.: Analyse der Bedienbarkeit. Diploma thesis, Humboldt-Universität zu Berlin (2004)
18. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* **C-35**(8) (1986) 677–691
19. Kaschner, K.: BDD-basiertes Matching von Services. Diploma thesis, Humboldt-Universität zu Berlin (2006)
20. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing Interacting BPEL Processes. In Dustdar, S., Fiadeiro, J.L., Sheth, A., eds.: Proceedings of the 4th International Conference on Business Process Management (BPM 2006). Volume 4102 of LNCS., Springer-Verlag (2006) 17–32
21. Kindler, E., Martens, A., Reisig, W.: Inter-operability of Workshop Applications - Local Criteria for Global Soundness. In van der Aalst, W.M.P., Desel, J., Oberweis, A., eds.: Business Process Management. Volume 1806 of LNCS., Springer-Verlag (2000) 235–253
22. Arias-Fisteus, J., Fernández, L.S., Kloos, C.D.: Applying model checking to BPEL4WS business collaborations. In Haddad, H., Liebrock, L.M., Omicini, A., Wainwright, R.L., eds.: Proceedings of the 2005 ACM Symposium on Applied Computing (SAC), ACM (2005) 826–830
23. Juric, M.B., Mathew, B., Sarang, P.: Business Process Execution Language for Web Services. Packt Publishing (2004)
24. Gaur, H., Zirn, M., eds.: BPEL Cookbook: Best Practices for SOA-Based Integration and Composite Applications Development. Packt Publishing (2006)
25. Postel, J.B.: Simple Mail Transfer Protocol. RFC 821, Information Sciences Institute, University of Southern California, Network Working Group (1982)

## Appendix A. Proofs

### Lemma 1

*Proof.* ( $\rightarrow$ ) Let  $[q_P, q_R, m]$  be a deadlock. Then the first item is true by definition of deadlocks. The second item must be true since otherwise  $R$  has a move.  $[q_P, m]$  must be stable since otherwise  $P$  has a move. For  $x \in \text{wait}(q_R)$ , we may conclude  $m(x) = 0$  since otherwise  $R$  has a move.

( $\leftarrow$ ) Assume, the three conditions hold. By the first item, the considered state is not a final state of  $P \oplus R$ .  $P$  does not have a move since  $[q_P, m]$  is stable.  $R$  does not have a move since internal and send moves are excluded by the second item, and receive moves are excluded by the last item.  $\square$

### Lemma 2

*Proof.* As  $cl([q_{0P}, []])$  is the set of situations that can be reached from the initial state without interference of any partner  $R$ ,  $k$ -boundedness is immediately violated.  $\square$

### Lemma 3

*Proof.* By structural induction on  $\delta$ . By definition of  $cl$ ,  $cl([q_{0P}, []])$  is the set of situations that can be reached from the initial state without interference from  $S_i$ . If  $K(q) = q$ , then  $cl(\text{event}(q, x))$  is by definition of  $\text{event}$  and  $cl$  exactly the set of situations that can be reached from situations in  $q$  by the event  $x$ . Thus,  $[q, x, q'] \in \delta_i$  implies  $K(q') = q'$ .  $\square$

### Corollary 2

*Proof.* Follows with Lemma 1 and Def. 8 from the fact, that all states of  $S$  satisfy their annotations.  $\square$

### Lemma 4

*Proof (Sketch).* The inclusion  $K(q_R) = \bigcup_{[q_R, q_S] \in L} q_S$  follows from the definition of  $q_{0S_0}$ ,  $\delta_{S_0}$  and the concepts  $cl$  and  $\text{event}$ . For the reverse inclusion, let  $[q_P, m] \in K_{q_R}$ , that is,  $[q_P, q_R, m] \in P \oplus R$ . Thus, there is a transition sequence in  $P \oplus R$  from the initial state  $[q_{0P}, q_{0R}, []]$  to that state. This sequence can be replayed in  $P \oplus S_0$  by replacing actions of  $R$  with actions of  $S_0$ , leading to a state  $q_S$  with  $[q_P, m] \in K(q_S) = q_S$ .  $\square$

### Corollary 3

*Proof.* Since the annotations are conjunctions built for every element of  $K(q_S)$ , the annotation corresponding to the union of these values is the conjunction of the individual formulas.  $\square$

*Lemma 5*

*Proof.* Let  $i$  be the smallest number such that there exist  $q_R \in Q_R$  and  $q_S \in Q_{S_i} \setminus Q_{S_{i+1}}$  holding  $[q_R, q_S] \in L$ . That is,  $q_S$  is, among the states of  $S_0$  appearing in  $L_{R, S_0}$ , the one that is removed first during the process described in Def. 11.

By the construction of Def. 11,  $\phi(q_S)$  evaluates to *false* with the assignment  $ass_{S_i}(q_S)$ . Thus, there is a  $[q_P, m] \in K(q_S)$  such that  $[q_P, q_S, m]$  is a deadlock in  $P \oplus S_i$ . As a deadlock, it is also a wait state in  $S_i$ , so  $q_S \in F_{S_i}$ .

In  $S_0$ , there is, for every  $x \in C \cup \{\tau\}$ , a transition leaving  $q_S$ . If such a transition is not present from  $q_S$  in  $S_i$ , this means that the corresponding successor state has been removed in an earlier iteration of the process described in Def. 11. Such a transition cannot leave  $q_R$  in  $R$  since otherwise a successor of  $R$  were matched with a state  $q'_S$  that has been removed in an earlier iteration than  $q_S$  which contradicts the choice of  $i$  and  $q_S$ . Consequently, for every  $x$  with an  $x$ -transition leaving  $q_R$  in  $R$ , there is an  $x$ -transition leaving  $q_S$  in  $S_i$ . This means that, for all  $x \in C \cup \{\tau, final\}$ ,  $ass_{S_i}(q_S)(x) \geq ass_R(q_R)(x)$ . Since  $\phi(q_S)$  is monotonous (only built using  $\vee$  and  $\wedge$ ), and  $\phi_R$  is a conjunction containing  $\phi(q_S)$  (by Cor. 3),  $\phi(q_R)$  evaluates to *false* with the assignment  $ass_R(q_R)$ . Consequently, by Cor. 1,  $P \oplus R$  has a deadlock.  $\square$

*Corollary 4*

*Proof.* If  $S$  is well-defined then, by Cor. 2, at least  $S$  is a partner of  $P$  such that  $P \oplus S$  is deadlock-free. If  $P$  has a partner  $R$  such that  $P \oplus R$  is deadlock-free, Lemma 5 asserts that  $L_{R, S_0}$  contains only states of  $S$ . In particular, since in any case  $[q_{0R}, q_{0S_0}] \in L_{R, S_0}$ , this implies  $q_{0S_0} = q_{0S} \in Q_S$ .  $\square$

*Theorem 1*

*Proof.* If  $P \oplus R$  is deadlock-free, then Lemma 5 asserts that the matching of  $R$  with  $S$  (or  $S^*$ ) coincides with the matching of  $R$  with  $S_0$ . Thus, requirement (topology) holds. Furthermore, Cor. 3 guarantees that requirement (annotation) is satisfied.

Assume that both requirements hold. By requirement (topology), the matching of  $R$  with  $S$  (or  $S^*$ ) coincides with the matching of  $R$  with  $S_0$ , since the matching with  $S_0$  can lead to states outside  $S$  only if there is an  $x$  such that an  $x$ -transition is present in a state  $q_R$  but not in the corresponding state  $q_S \in S$ . Given that both matchings coincide, Cor. 3 states that  $\phi(q_R)$  is the conjunction of the  $\phi(q_S)$ , for the matching states  $q_S$ . Then, we can deduce from Cor. 1 and requirement (annotation) that  $P \oplus R$  is deadlock-free.  $\square$