Matching Nondeterministic Services with Operating Guidelines

Peter Massuthe and Karsten Schmidt

Humboldt-Universität zu Berlin
Institut für Informatik
Unter den Linden 6
D-10099 Berlin
{massuthe, kschmidt}@informatik.hu-berlin.de

Abstract Interorganizational cooperation is more and more organized by the paradigm of *services*. The *service-oriented architecture* (SOA) provides a general framework for service interaction. It describes three roles, *service provider*, *service requester*, and *service broker*, together with the operations *publish*, *find*, and *bind*.
We provide a formal method based on *nondeterministic automata* to model services and their interaction. We suggest *operating guidelines* as a convenient and intuitive artifact to realize *publish*. In our approach, the *find* operation reduces to a matching problem between the requester's service and operating guidelines.
In this paper, matching of deterministic as well as nondeterministic automata with operating guidelines is presented.

Keywords: Services, SOA, Formal Methods, (Nondeterministic) Automata, Operating guidelines, Matching

1   Introduction

Nowadays, cooperation across borders of enterprises is increasingly important. Functionalities are sourced out or so-called virtual enterprises for specific tasks are formed. In this setting, *services* play an important role. A service basically encapsulates self-contained functions that interact through a well-defined interface. Recent publications apply the term service in different contexts with varying denotations. In this paper, we assume the essentials of a service to include its *identifier* (id), its *interface*, and its *operational behavior*.

The well-known class of *web services* is an implementation of services with an interface specified in WSDL [2] and an id given by an URI. Web services have become particularly important since the establishment of BPEL [3] as a widely excepted language to describe web services.

Typically, a service is not executed in isolation, but in cooperation with other services. The *service-oriented architecture* (SOA) [4] provides a general framework for service interaction. It describes three roles of service owners: *service provider*, *service requester*, and *service broker*. A service provider *publishes* information about his service to a repository.

The service broker manages the repository and allows a service requester to *find* an adequate service provider. Then, the service of the provider and the service of the requester may *bind* and start interaction. The SOA triangle is depicted in Fig. 1.
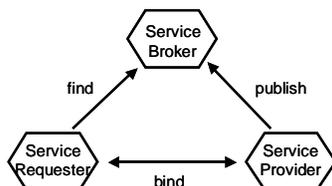

Figure 1. The service-oriented architecture (SOA).

The interaction of services may cause non-trivial communication. Thus, it is desirable that the broker selects, for a given requester's service R, only those provided services P from the repository that are guaranteed to interact properly with R: The services R and P must neither deadlock in their interaction nor send unanticipated messages, for instance. Thereby, compatibility of the interfaces of R and P is not sufficient to guarantee proper interaction.

The broker must decide this task by help of the published information about P. In a currently quite popular approach, the published information is a so-called *public view* [5, 6], i.e. an abstract version P' of P with a communication behavior equivalent to P. In this setting, the broker must perform a *compliance check*, i.e. a check whether the composed system of R and P' together behaves well in the above described way. The compliance check is mainly a combination of (a) the composition of R and P' and (b) a verification task for the (non-)reachability of states in the composed system.

We suggest an alternative: The provider does not publish information about *his* service P, but information about all properly interacting services R of potential *requesters*, instead. An implicit representation of this information is called *operating guideline* [7, 8] for P. In this setting, the broker must solve a *matching* problem. Matching R with OG_P means to check whether or not R is described by OG_P. This is basically a test for an inclusion relationship of R in OG_P. If R matches the published operating guideline of P then it is per construction guaranteed that P and R interact properly. We claim that *matching* a requester's service R with an operating guideline OG_P is less complex than matching R with the public view P' of P.

The rest of the paper is structured as follows. In Sec. 2, we provide a formal model, called *service automata*, to describe all aspects of the SOA triangle. With service automata, we can model both provider's and requester's services. Interaction of services is modeled as the composition of their models, and can therefore be seen as the result of *bind*. In our approach, we abstract from every other aspect of *bind* such as routing

and establishment of communication channels. We assume this to be managed by an underlying middleware.

Then, in Sec. 3, we show how a *set* of automata can be represented by an *annotated automaton*. An annotated automaton that represents the set of all proper interacting services R for a service P is called *operating guideline* for P. We propose operating guidelines as a convenient and intuitive artifact to be published to the service broker.

Finally, in Sec. 4, we consider the matching of service automata with operating guidelines in detail. In [7], we just sketched the algorithm for matching deterministic automata with operating guidelines. Here, the matching algorithm for deterministic automata, as well as for nondeterministic automata will be presented. This represents the SOA *find* operation.

In this paper, we present our approach only for acyclic service automata.

2   A Formal Model for Services

In this section, we propose service automata, a class of nondeterministic automata, as a formal model for services. The model reflects the three essentials of services as described in Sec. 1. It abstracts from other aspects of services such as real-time constraints, cost models, underlying middleware etc.

The communication behavior of a service is modeled as labels to transitions. Labels represent incoming or outgoing messages. We abstract from message content. Throughout the paper, we assume a set MC of message channels.

Definition 1 (Service automaton). *A* service automaton *is a nondeterministic automaton* $A = [I, Q, T, q_0, \Omega]$, *that consists of an* interface $I = [I_{in}, I_{out}]$ *such that* $I_{in} \cup I_{out} \subseteq MC$ *and* $I_{in} \cap I_{out} = \emptyset$, *a set* Q *of* states, *a set* $T \subseteq Q \times L \times Q$ *of* transitions *where* $L = \{?x \mid x \in I_{in}\} \cup \{!x \mid x \in I_{out}\} \cup \{\tau\}$, *an* initial state $q_0 \in Q$, *and a set* $\Omega \subseteq Q$ *of* final states. L *is called the set of* labels *of* A.

We denote service automata as A, P, R, or S. P and R are used if we emphasize the role of its owner as service provider and service requester, respectively. S is used if we want to emphasize that the automaton is a strategy (to be defined later). We denote the ingredients of a service automaton A with $I_A, Q_A, T_A, q_{0_A}, \Omega_A$, if A is not clear from the context. As a running example, we consider a provider's service of a beverage vending machine, modeled by service automaton P_V of Fig. 2(a). The service provided by this machine expects a coin to be inserted (?€) and one of two buttons (one for tea and one for coffee) being pressed (?T and ?C). The service then reacts with delivering a beverage (!B). Assume two final states of P_V: p5 and p6.

Consider now two requesters who want to use the provided vending machine service. Their services are modeled as automata R_C and R_E, depicted in Fig. 2(b) and 2(c), respectively. R_C models a customer who wants coffee, whereas the customer modeled by R_E apparently "forgets"
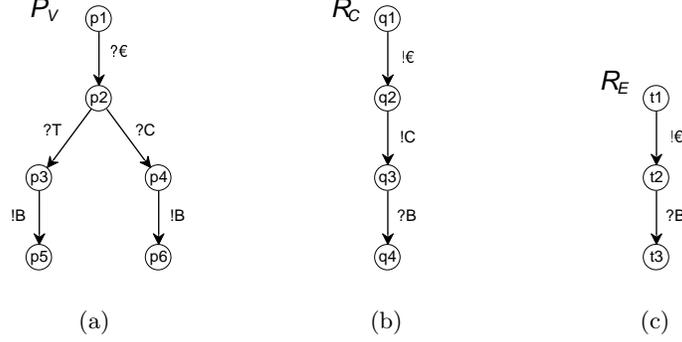
3

Figure 2. The provider's service automaton P_V of a vending machine and two service automata R_C and R_E of requesters.

to press one of the machine buttons, so that he is an erroneous customer of the vending machine. Assume the states q4 and t3 are the final states of R_C and R_E, respectively.

We assume an asynchronous model for message passing between two services. This model is formalized through the definition of the behavior of two service automata in interaction (Definition 2).

Two communicating automata A and B must have interfaces such that one automaton sends messages that are received by the other one, and vice versa, i.e. $I_{in_A} = I_{out_B}$ and $I_{out_A} = I_{in_B}$. From now on, we assume this to be given when composing to service automata. In the following definitions, $bags(MC)$ denotes the set of all multisets over MC. $M + a$ stands for incrementing the multiplicity of a in M by 1, and $M - a$ for decrementing the multiplicity of a in M by 1. $a \in M$ is *true* if the multiplicity of a in M is at least 1. $\{\}$ denotes the empty multiset.

Definition 2 (Interaction of service automata). *Let P and R be two service automata. Let, without loss of generality, $Q_R \cap Q_P = \emptyset$. The transition system $(R \oplus P) = [Q, T, q_0]$, describing the interaction of R and P, consists of a set of states $Q \subseteq Q_R \times Q_P \times bags(MC)$ and a set of labeled transitions $T \subseteq Q \times (T_P \cup T_R) \times Q$ inductively defined as follows:*
Basis: $q_0 = [q_{0_R}, q_{0_P}, \{\}]$ is a state of the transition system, called the initial state. Step: If $q = [q_R, q_P, M]$ is a state, and

- *there is a transition $t = [q_R, !a, q'_R] \in T_R$, then $q' = [q'_R, q_P, M + a] \in Q$ and $[q, t, q'] \in T$.*
- *there is a transition $t = [q_P, !a, q'_P] \in T_P$, then $q' = [q_R, q'_P, M + a] \in Q$ and $[q, t, q'] \in T$.*
- *$a \in M$, and there is a transition $t = [q_R, ?a, q'_R] \in T_R$, then $q' = [q'_R, q_P, M - a] \in Q$ and $[q, t, q'] \in T$.*
- *$a \in M$, and there is a transition $t = [q_P, ?a, q'_P] \in T_P$, then $q' = [q_R, q'_P, M - a] \in Q$ and $[q, t, q'] \in T$.*
- *there is a transition $t = [q_R, \tau, q'_R] \in T_R$, then $q' = [q'_R, q_P, M] \in Q$ and $[q, t, q'] \in T$.*
- *there is a transition $t = [q_P, \tau, q'_P] \in T_P$, then $q' = [q_R, q'_P, M] \in Q$ and $[q, t, q'] \in T$.*

4

A state $[q_R, q_P, M]$ with no successors in this transition system is called end state *if* $q_P \in \Omega_P$, $q_R \in \Omega_R$, *and* $M = \{\}$. *Otherwise, a state without successors is called* deadlock. *We say that the composed system of P and R is* weakly terminating *iff $R \oplus P$ does not have deadlocks.*

A state in the composed system represents a state of P, a state of R, and a multiset M of pending messages. Every transition corresponds either to a transition in P or to a transition in Q. Transitions labeled $!a$ create a message in channel a, transitions labeled $?a$ consume a message from channel a and can only occur if a is present in the channel. τ-transitions do neither create nor consume messages.

As an example, the transition systems $R_C \oplus P_V$ and $R_E \oplus P_V$ are depicted in Fig. 3. It is easy to see that $[\mathsf{q4}, \mathsf{p6}, \{\}]$ is an end state, whereas $[\mathsf{t2}, \mathsf{p2}, \{\}]$ is a deadlock. Hence, $R_C \oplus P_V$ is weakly terminating and $R_E \oplus P_V$ is *not* weakly terminating.

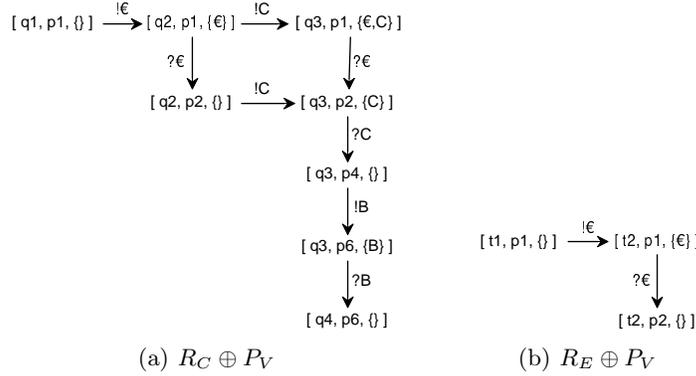(a) $R_C \oplus P_V$ (b) $R_E \oplus P_V$

Figure 3. The two transitions systems $R_C \oplus P_V$ and $R_E \oplus P_V$.

In this paper, for a given service automaton P, we are interested in the set of *all* service automata R such that $R \oplus P$ is weakly terminating.

Definition 3 (Strategy). *Let P be a service automaton. A service automaton R is called* strategy *for P iff $R \oplus P$ is weakly terminating.*

In our example, R_C is a strategy for P_V and R_E is no strategy for P_V. The term strategy originates from a control-theoretic point of view (see [1, 9], for instance): We may see R as a controller for P enforcing the property of weak termination.

Several results in the forthcoming sections are based on a state-by-state characterization of those service automata that are strategies. This characterization uses a mapping that we call *knowledge*.

Definition 4 (Knowledge). *Let R and P be two service automata. Then, the knowledge function $k_{(R,P)}$ is a mapping $k_{(R,P)} : Q_R \to Q_P \times bags(MC)$, such that $k_{(R,P)}(q_R) = \{[q_P, M] \mid [q_R, q_P, M] \in Q_{R \oplus P}\}$.*

Informally, $k_{(R,P)}(q_R)$ represents the set of states where P and the message channels can be (in $R \oplus P$), while R is in q_R, i.e. the knowledge of R about P and M.

Using the knowledge $k_{(R,P)}$, we can characterize strategies as follows.

Lemma 1. *R is a strategy for P iff, for all $q_R \in Q_R$ and all $[q_P, M] \in k_{(R,P)}(q_R)$, at least one of the following conditions holds:*

- *$q_R \in \Omega_R$, $q_P \in \Omega_P$, and $M = \{\}$;*
- *there is a transition $t = [q_P, l, q'_P]$ in T_P, such that there is a transition in $R \oplus P$ that leaves $[q_R, q_P, M]$ and is labeled with t;*
- *there is a transition $t = [q_R, l, q'_R]$ in T_R, such that there is a transition in $R \oplus P$ that leaves $[q_R, q_P, M]$ and is labeled with t.*

We omit the proof since the result is basically a reformulation of the definition of weak termination. Nevertheless, the theorem shall turn out to be useful since it describes explicitly the obligations for strategies R: Whenever P has no transition leaving a state in $R \oplus P$ (which is determined since we assume P to be given) then R is obliged to have one (i.e. we have to design R such that it is capable of leaving such states). It is again easy to check the service automata R_C and R_E for these criteria. Whereas R_C fulfills Lemma 1, R_E violates all three criteria in state t2: There is a tuple $[\mathsf{p2}, \{\}]$ in $k_{(R_C, P_V)}(\mathsf{t2})$ such that neither $[\mathsf{t2}, \mathsf{p2}, \{\}]$ is an end state, nor there is a transition possible, since all transitions leaving t2 (p2) are consuming transitions in R_E (P_V), but M is empty.

3 Operating Guidelines

Operating guidelines for P are a compact representation of the set of *all* strategies for P. We first consider a subset $\mathcal{D}_P$ of that set, i.e. all *deterministic* strategies, and then show how nondeterministic strategies can be matched with an operating guideline that (directly) represents only the deterministic subset. A service automaton is deterministic if it does not contain τ-transitions and does not have states that are left by multiple transitions with equal label. As stated in the introduction, we restrict ourselves to *acyclic* automata. An automaton is acyclic if its transition relation does not contain cycles.

If P is acyclic, we may unroll every deterministic strategy R of P to an equivalent strategy R' which is a *tree automaton*, i.e., a service automaton where each state has at most one incoming transition. Since P is acyclic, R' has limited depth. Matching non-tree automata with OG_P can therefore be reduced to matching tree automata with OG_P. It is thus sufficient to consider the set $\mathcal{DT}_P$ of all (deterministic) tree automata which are strategies for P.

$\mathcal{DT}_P$ may become large, so an explicit enumeration of its elements is intractable. Fortunately, we are able to implicitly represent this set in a size that is not much larger than the size of one of its members. To this end, we proceed as follows.

In a first step, we recall the concept of annotated automata from [10] which we use as an implicit representation of a set of sub-automata of the

annotated automaton. In a second step, for an arbitrary tree automaton A, we translate the criteria of Lemma 1 into particular annotations to represent exactly that set of sub-automata of A which are strategies. In the final step, we exhibit tree automata S which have *all* deterministic strategies for P as sub-automata. We annotate S and get a structure that we call *operating guideline* for P. It represents exactly the set $\mathcal{DT}_P$ and comprises therefore every proper deterministic interaction with P.

Definition 5 (Sub-automaton). *An automaton A' is a sub-automaton of an automaton A, $A' \sqsubseteq A$, iff $Q_{A'} \subseteq Q_A$, $T_{A'} \subseteq T_A$, $q_{0_{A'}} = q_{0_A}$, and $\Omega_{A'} \supseteq Q_{A'} \cap \Omega_A$.*

Consider a deterministic tree automaton A and a function Φ that maps every state q of A to a boolean formula $\Phi(q)$. Thereby, the propositions of $\Phi(q)$ are labels of transitions that leave q in A. Φ is called an *annotation* to A. A tree automaton with an annotation is called *annotated (tree) automaton* and denoted A^{Φ}.

Definition 6 (Φ-compliance). *Let A^{Φ} be an annotated automaton, A' a sub-automaton of A, and $q \in Q_{A'}$ (and therefore $q \in Q_A$). q is compliant with $\Phi(q)$ iff $\Phi(q)$ is true under the assignment assigning true to all propositions that are labels of transitions leaving q in A', and false to all other propositions. A' is compliant with A^{Φ} (denoted $A' \models A^{\Phi}$) iff all states $q \in Q_{A'}$ are compliant with $\Phi(q)$.*

With a compliant sub-automaton A' of A, we call every automaton A'' that is isomorphic to A' compliant to A^{Φ}, too.
Let $\mathcal{A}(A^{\Phi}) = \{A' \mid A' \sqsubseteq A, A' \models A^{\Phi}\}$. This way, a single annotated automaton A^{Φ} represents a set of automata $\mathcal{A}(A^{\Phi})$.
In the next step, we consider an arbitrary service automaton P and a deterministic tree automaton R. We aim at constructing a particular annotation Σ to R such that $\mathcal{A}(R^{\Sigma})$ is exactly the set of those sub-automata of R which are in $\mathcal{DT}_P$. For this purpose, let $q_R \in Q_R$. Then the formula $\Sigma(q_R)$ is built as a straight coding of the criteria in Lemma 1: $\Sigma(q_R)$ is the *conjunction* of sub-formulae $\sigma_{(q_R, q_P, M)}$, for all $[q_P, M] \in k_{(R,P)}(q_R)$. If $k_{(R,P)}(q_R) = \emptyset$, let $\Sigma(q_R) = true$.
The sub-formula $\sigma_{(q_R, q_P, M)}$ is

- *true* if $q_R \in \Omega_R$, $q_P \in \Omega_P$, and $M = \{\}$;
- *true* if there is a transition of P leaving $[q_R, q_P, M]$ in $R \oplus P$;
- the disjunction of all l occurring as labels of transitions in R that leave $[q_R, q_P, M]$ in $R \oplus P$, otherwise.

Lemma 2. *Let R, P, Σ as described above. Then $\mathcal{A}(R^{\Sigma})$ is the set of those sub-automata of R which are in $\mathcal{DT}_P$.*

Proof. (Sketch) For a tree automaton A and a sub-automaton A' it is easy to verify that, for all $q \in Q_{A'}$, $k_{(A',P)}(q) = k_{(A,P)}(q)$. Then, by the construction of Σ, $A' \models \mathcal{A}(A^{\Sigma})$ iff every state of A' satisfies the criteria stated in Lemma 1. q.e.d.

As the remaining step towards the definition of operating guidelines, we discuss tree automata S such that every tree automaton which is

a strategy of the given service automaton P, is a sub-automaton of S. Then, we can annotate S and gain a representation of all (tree) strategies for P.

We can construct, for an arbitrary number n, a *complete* tree automaton that has depth n in all paths, and where every inner node has, for every $x \in I_{out_P}$, a leaving transition labeled $?x$, and for every $y \in I_{in_P}$, a leaving transition labeled $!y$. This automaton, denoted A_n^C, has *every* deterministic tree automaton with depth at most n as sub-automaton. Thus, it covers all deterministic tree strategies of depth at most n. By assumption, P is acyclic. Thus, there is a maximum value l_P for the lengths of paths through P, starting from the initial state. In a deterministic strategy, every transition produces or consumes a message. Every message consumed by a strategy must have been produced by P beforehand. Every message produced by a strategy must be consumed by P subsequently. Thus, a deterministic tree strategy of P cannot be deeper than l_P.

Consequently, $A_{l_P}^C$ is a tree automaton that contains every tree strategy as sub-automaton. Thus it holds, after adding the annotation Σ defined earlier in this section:

Theorem 1 (Characterization of strategies). $\mathcal{A}((A_{l_P}^C)^{\Sigma}) = \mathcal{DT}_P$.

Thus, the following definition of operating guidelines is justified:

Definition 7 (Operating guideline). *Let P be an arbitrary service automaton. Let S be a tree automaton such that every tree automaton that is a strategy for P, is a sub-automaton of S. Then S^{Σ} is called an* operating guideline *for P.*

Above, we showed that every P has an operating guideline, namely $(A_{l_P}^C)^{\Sigma}$. The exhibited one is, however, not the perfect one. It has exponential size in the size of the interface of P. In particular, it may be significantly larger than any actual tree strategy of P. In the remainder of this section, we show that there exists a canonical operating guideline, OG_P, which is not significantly larger than a particular tree strategy.

For this purpose, consider $(A_{l_P}^C)^{\Sigma}$. $A_{l_P}^C$ itself is not necessarily compliant with Σ. This is in particular the case if $A_{l_P}^C \oplus P$ contains a deadlock $[q^*, q_P, M]$. In that case, $\Sigma(q^*)$ is equivalent to *false*. Removing states q (and the subtree beyond q) from $A_{l_P}^C$ which are not compliant with $\Sigma(q)$ results in a smaller automaton still covering all tree strategies of P. This process can be iterated since removing a state may turn the annotation of its predecessor state from *true* to *false*. During the whole process, it can be shown that none of the removed states can be an element of any tree strategy of P. If the process terminates, the resulting automaton is either empty (then P does not have strategies), or is a strategy itself, the unique *most permissive tree strategy*, S_P, for P. Since it still covers *all* tree strategies, S_P^{Σ} is a valid operating guideline for P, the *canonical operating guideline*, OG_P. Its size is the size of S_P times the length of the largest annotation.

As an example, we recall our vending machine service automaton P_V. The canonical operating guideline OG_{P_V} for P_V (Fig. 2(a)) is depicted
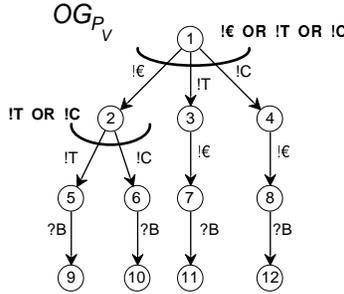
Figure 4. OG_{P_V} for the vending machine service automaton P_V.

in Fig. 4. It is constructed by removing states from the complete automaton of depth 3 with labels !€, !T, !C, and ?B leaving each state. The annotations of states with less than two successors are skipped.

It is easy to see that the service automaton R_C of Fig. 2(b) is compliant with the annotations. The automaton R_E of Fig. 2(c), instead, is not compliant since its state t2 (wich is corresponding to state 2 of OG_{P_V}) violates the formula attached to that state: There is no transition leaving t2 that is labeled with !T or !C.

We propose to use OG_P as an artifact generated by the owner of a provided service P which can be published to the service broker. Matching a deterministic service automaton R with OG_P amounts to unrolling R to a tree, to map the nodes of R to nodes of OG_P, and to evaluate the annotations in OG_P. All steps can be performed during a single depth-first search through the unrolled version of R and is thus linear in the size of R.

4   Matching nondeterministic automata with OG_P

In the previous section, we proposed the canonical operating guideline OG_P as a characterization of all *deterministic* strategies for P. In this section, we are interested in *all* strategies, including the nondeterministic ones. Our result in this regard is quite nice. OG_P is, without any change, capable of characterizing *all* strategies: We present an algorithm that receives an arbitrary acyclic service automaton R and OG_P as input, and is capable of deciding whether R is a strategy for P or not. Without loss of generality, we assume R to be given in its unrolled shape, i.e., as a nondeterministic *tree* automaton. We proceed with presenting the algorithm, followed by a justification of its correctness.

Our algorithm is based on a coordinated depth-first traversal of R and OG_P. This traversal assigns, to each state $q_R \in Q_R$, a "fitting" state in OG_P. Then, we evaluate the annotation of the assigned state, but only in those states of R that do not have leaving τ-transitions. We claim that R is a strategy if and only if all executed evaluations yield *true*. Let S be

9

the automaton underlying OG_P, i.e. $OG_P = S^\Sigma$. This amounts to the following pseudo-code for our matching algorithm:

```
01 proc main(R: strategy, OG_P: operating guideline)
02     dfs(q_0R, q_0S);
03     exit("yes")
04
05 proc dfs(q_R, q_S)
06     hasTau := False;
07     for all [q_R, l, q'_R] ∈ T_R do
08         if l = τ then dfs(q'_R, q_S); hasTau := true;
09         else if ¬∃q'_S : [q_S, l, q'_S] ∈ T_S then exit("no");
10         else dfs(q'_R, q'_S); /* note that q'_S is unique if it exists */
11     if ¬ hasTau then
12         result := evaluate Σ(q_S) with assignment defined by q_R;
13         if ¬ result then exit("no");
14     return.
```

As an example, consider a slightly changed vending machine, wich now distinguishes between tea (BT) and coffee (BC). The relevant part of its operating guideline is depicted in Fig. 5 (a)-right and (b)-right, together with two new nondeterministic requesters. The requester of Fig. 5(a) is in state p3 capable of receiving a coffee or to internally decide for a tea. The call dfs(p3, 3) results in hasTau being *true* and hence the formula $\Sigma(3) = ?\mathsf{BT}$ is not evaluated for p3. All executed evaluations in Fig. 5(a) yield *true*. Thus, Fig. 5(a) is an example for positive matching.
In contrast, in Fig. 5(b) the algorithm returns "no". The depicted requester is dual to the left one: In state q3, he is capable of receiving a tea or, e.g. after a timeout, to decide for a coffee. In call dfs(q5, 3), there is no τ-transition leaving q5 and thus the formula $\Sigma(3) = ?\mathsf{BT}$ is evaluated: to *false*.

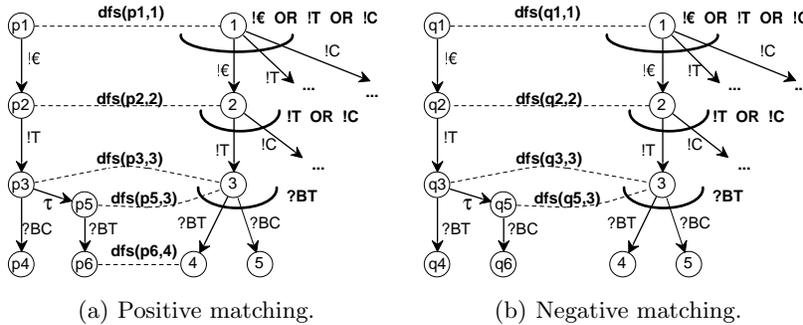(a) Positive matching.   (b) Negative matching.

Figure 5. An example for the matching algorithm calls of $\mathrm{dfs}(q_R, q_S)$.

The first observation involved in justifying this algorithm is:

Lemma 3. *For each called instance* $\mathrm{dfs}(q_R, q_S)$*:* $k_{(R,P)}(q_R) = k_{(S,P)}(q_S)$.

Proof. (Sketch, induction over the transition relation of R) Basis: The k-values of the initial states coincide since they correspond to the states reachable in P without any transition of R nor S. Step: Let $[q_R, l, q_R'] \in T_R$ and $k_{(R,P)}(q_R) = k_{(S,P)}(q_S)$. If $l = \tau$ then we call $\mathrm{dfs}(q_R', q_S)$ and it holds $k_{(R,P)}(q_{R'}) = k_{(R,P)}(q_R) = k_{(S,P)}(q_S)$ since the τ-transition does not change the status of message channels and does not enable nor disable any transition in S. If $l \neq \tau$, it can be shown that, for the unique q_S' holding $[q_S, l, q_S'] \in T_S$, $k_{(R,P)}(q_{R'}) = k_{(S,P)}(q_{S'})$ since, in a tree automaton, the k-value of the target of a transition is uniquely determined by the k-value of the source state of the transition and its label. q.e.d.

With this observation, it is easy to show

Theorem 2 (Justification: Implication). *If the above algorithm exits with "yes", R is a tree strategy for P.*

Proof. For every state $q_R \in Q_R$, the conditions of Lemma 1 are satisfied: Either, there is a τ-transition leaving q_R. Then the third condition is true since the τ-transition is executable in q_R independently from q_P and M. Or, there is no τ-transition leaving q_R. Then, the conditions hold since the state q_S assigned to q_R by the coordinated depth-first search has the same k-value as q_R, and so the annotation of q_S correctly codes the conditions of Lemma 1 for q_R. q.e.d.

For the justification, it remains to show that, whenever the algorithm exits with "no", then R is not a strategy. We use the following lemma.

Lemma 4. *Let R be a tree strategy that has a transition $[q, \tau, q']$. Let $Q_R' = Q_R \setminus \{q\}$. If $q = q_{0_R}$ then let $q_{0_{R'}} = q'$, otherwise let q'' be the unique state such that $[q'', l, q] \in T_R$ and let $T_{R'} = (T_R \cup \{[q'', l, q']\}) \setminus \{[q'', l, q], [q, \tau, q']\}$. Then holds: R' is a tree strategy, too.*

Proof. (Sketch) Since the occurrence of a τ-transition in R is not constrained by P, R may decide to execute it whenever it is possible. If R is a strategy then it is still one if it obeys this "τ first" rule. R' describes basically R under this rule for a particular τ-transition. q.e.d.

Assume, our algorithm exits with "no" in line 9, during a call $\mathrm{dfs}(q_R, q_S)$ but R is a strategy. Then, using Lemma 4, there would exists a deterministic strategy that has a state q_R corresponding to q_S with a transition leaving q_R labeled l. Since q_S does not have this transition but is known to contain all deterministic strategies as sub-automaton, we obtain a contradiction.

Assume that our algorithm answers, though R is a strategy, "no" in line 13 during a call $\mathrm{dfs}(q_R, q_S)$. Then, with the same argument as above, we can device a deterministic strategy that runs into the same situation and contradicts the relation between OG_P and deterministic strategies established in the previous section. Consequently:

Theorem 3 (Justification: Replication). *If the above algorithm exits with "no", R is not a tree strategy of P.*

11

The number of calls to dfs is at most the number of states of R. Our algorithm is thus an efficient instrument for matching an acyclic service automaton R with an operating guideline OG_P.

5 Conclusion

We introduced *operating guidelines* for a service P as an artifact that characterizes all services which interact properly with P. Though the guidelines originally cover all *deterministic* strategies, they can, without change, be used for a characterization of *all* strategies. The matching algorithm is linear in the size of the requesting service.

In current research, we are extending the approach to services with cycles, and apply the concept of operating guidelines to further problems, e.g. exchangeability of services.

References

1. C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer Academic Publishers, 1999.
2. Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weeravarana. Web Service Discription Language (WSDL) 1.1. Technical report, Ariba, International Business Machines Corporation, Microsoft, March 2001.
3. F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Specification, BEA Systems, IBM, Microsoft, SAP, Siebel, 05 May 2003. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbiz2k2/html/bpel1-1.asp.
4. Karl Gottschalk. Web Services Architecture Overview. Ibm whitepaper, IBM developerWorks, 01 September 2000. http://ibm.com/developerWorks/web/library/w-ovr/.
5. F. Leymann, D. Roller, and M. Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2), 2002.
6. A. Martens. *Verteilte Geschäftsprozesse - Modellierung und Verifikation mit Hilfe von Web Services.* PhD thesis, Institut fr Informatik, Humboldt-Universität zu Berlin, 2004.
7. P. Massuthe, W. Reisig, and K. Schmidt. An operating guideline approach to the SOA. Techn. Report 191, Humboldt-Universität zu Berlin, 2005. Submitted to a conference.
8. P. Massuthe and K. Schmidt. Operating guidelines - an alternative to public view. Techn. Report 189, Humboldt-Universität zu Berlin, 2005.
9. P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization*, 25(1):206–230, 1987.
10. A. Wombacher, P. Fankhauser, B. Mahleko, and E. Neuhold. Matchmaking for business processes based on choreographies. *International Journal of Web Services*, 1(4):14–32, 2004.