Guaranteeing Weak Termination in Service Discovery

Karsten Wolf
Universität Rostock, 18051 Rostock, Germany
karsten.wolf@uni-rostock.de

Christian Stahl
Department of Mathematics and Computer Science
Technische Universität Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
c.stahl@tue.nl

Daniela Weinberg, Janine Ott, Robert Danitz
Institut für Informatik
Humboldt-Universität zu Berlin,
Unter den Linden 6, 10099 Berlin, Germany
weinberg,jott,danitz@informatik.hu-berlin.de

Abstract. A big issue in the paradigm of Service Oriented Architectures (SOA) is service discovery. Organizations publish their services via the Internet. These published services can then be automatically found and accessed by other services, meaning, the services are composed. A fundamental property of a service composition is weak termination, which guarantees the absence of deadlocks and livelocks. In principle, weak termination can be verified by inspecting the state space of the composition of (public views of) the involved services.

We propose a methodology to build that state space from precomputed fragments, which are computed upon publishing a service. That way, we shift computation effort from the resource critical “find” phase to the less critical “publish” phase. Interestingly, our setting enables state space reduction methods that are intrinsically different from traditional state space reductions. We further show the positive impact of our approach to the computational effort of service discovery.

Keywords: deadlock- and livelock freedom, SOA, service discovery, state space reduction

Address for correspondence: Universität Rostock, 18051 Rostock, Germany
* supported by the DFG within grant “Operating Guidelines for Services” (WO 1466/8-1)
1. Introduction

1.1. Background and motivation

Today’s organizations are challenged to continuously adapt their systems to address changes in their environment. On the one hand, systems are highly complex, run in heterogeneous environments, and are often distributed over several organizations. On the other hand, the ever-changing market conditions and regulations require organizations to act flexibly. Systems are subject to ongoing changes, but the integration of these changes should not take much time. Hence, new technologies are needed to support the development and maintenance of such systems.

Service Oriented Computing (SOC) [30] is a novel computing paradigm that facilitates the design of complex systems by connecting less complex systems, called services. A service encapsulates a certain functionality and offers it to other services over a well-defined interface. The interface of a service consists of a set of message channels and is used to communicate with other services. To this end, services are composed by connecting their message channels.

The key technology to design and to execute systems according to the paradigm of SOC is a Service Oriented Architecture (SOA). A SOA provides an IT infrastructure for publishing services of an organization via the Internet. Using standardized interfaces and message protocols these published services can then be automatically found and accessed by other organizations. That way, a SOA enables interoperability between systems and hence reduces complexity of systems.

A SOA defines three roles of services: a service provider offers the functionality of his service by publishing it (or an abstract version, the public view) in a service registry; a service broker manages the registry; a service requester queries the registry of a broker to find a service which provides certain functionality. If an appropriate service has been found in the registry, the broker returns the relevant data needed to establish a connection between the service of the requester and the service of the provider. As each service is published only once, but potentially considered in the find request multiple times, we expect that the number of “find” requests at a broker’s registry is significantly greater than the number of publish events. Moreover, answering a “find” request appears to be more resource (time and memory) critical than processing a “publish” event, as the “find” request may be much closer in time to the actual execution of the composed system. Service discovery—that is, the selection of an appropriate service—is a complex task. The selection of a certain service needs to take care of issues concerning the requested functionality and semantical, behavioral, and nonfunctional aspects.

In this article, we consider the behavioral aspects of service discovery and concentrate on the proper interaction between the two services; that is, we abstract from semantical and nonfunctional aspects and consider only data types and message types and not their content. Most articles on this issue [24,43] use deadlock freedom as a definition of proper interaction, but in practice a stronger correctness criterion is required that ensures in addition to the absence of deadlocks also the absence of livelocks (a livelock is a set of states which does not contain a final state and which cannot be left) in a service composition. We refer to this criterion as weak termination. Although relevant service description languages, such as WS-BPEL [5], avoid deadlocks and livelocks in single services, they cannot prevent them in service compositions [21].

In a naive approach toward ensuring weak termination, the service provider would publish a description of its service or an abstract version of it (e.g., described in WS-BPEL) in a registry. If a service requester queries this registry, then the broker has to find a service \( P \) in this registry such that the com-
position of $P$ and the service $R$ of the requester is weakly terminating. To this end, the requester needs to provide information about his service $R$—usually in the form of a service description of $R$. Then, the broker would construct the transition systems that reflects the behavior of the composition of the two services $P$ and $R$ and model check it for weak termination. If the composition of two services is weakly terminating, we say that both services match. In the branching time logic CTL, weak termination can be formalized by $\text{AG}\, EF$ final state; that is, from every state that is reachable from the initial state, it is always possible to reach a final state. This procedure would then be executed upon every “find” request. We argued earlier that a “find” request is rather resource critical. For this reason, our approach shifts this computation effort from the “find” request to the execution of a “publish” event.

1.2. Approach

In the following, we illustrate our approach, and we explain in a second step how it can be applied to support service discovery.

Let $P$ and $R$ be services, and let $P \oplus R$ be their composition. As services interact asynchronously, the composition $P \oplus R$ interleaves transitions of $P$ and $R$. Thus, our idea is to precompute those parts of $P \oplus R$ that take place between two subsequent transitions of $R$. We refer to such a part as a fragment of $P$. We shall show that there exists a finite set of fragments of $P$ from which we can build arbitrary composed systems involving $P$. In particular, this set is independent of any $R$ to be composed with $P$.

A big advantage of this approach is that we can apply substantial state space reduction techniques to the fragments, thus computing a composed system from fragments of $P$ that is significantly smaller than the original system $P \oplus R$. As a result, we need to verify only a smaller state space for weak termination. The approach is applicable, because the proposed reduction techniques guarantee that the computed composed system is weakly terminating if and only if $P \oplus R$ is.

Interestingly, the described setting calls for a way of state space reduction that differs significantly from standard model checking approaches. In model checking [10], it typically does not make much sense to reduce state spaces a posteriori—that is, after their actual calculation. In our setting, however, it makes sense to reduce the internal state space of fragments once they have been computed, because their actual integration into a complete transition system takes place at a different point in time. For our a posteriori state space reduction, we use a set of reduction rules adapted from [18].

Our proposed approach can be applied to support service discovery. We propose that a service provider publishes its service $P$ in the repository of a broker. The broker then computes the (reduced) fragments of $P$. Clearly, this computation requires some computational effort, but in the “publish” phase time is not a critical resource. The reduced fragments provide the broker with sufficient information to check whether a requesting service $R$ matches with $P$.

At the moment of a “find” request of a service $R$, the broker receives the service description of $R$ and calculates the composed system of $P$ and $R$ from the reduced fragments of $P$. In a next step, the broker model checks the composed system for weak termination. As our approach results in a system with a smaller state space than the state space of the original system $P \oplus R$, the actual check will be faster. This speed up is achieved by shifting computation time (for the fragments reduction) from the “find” phase to the “publish” phase.

In practice, services $P$ and $R$ are usually not formally modeled. Instead, various service description languages have been proposed by different industrial consortiums. Most prominent languages are WS-BPEL [5] and BPMN [29]. As our approach is based on a formal service description—we use state
machines—the broker has to translate services $P$ and $R$ into a formal model. The existence of formal semantics for most service description languages and tools to automatically translate a service into a formal model proves suitability of our approach (e.g., [11, 20]).

This article is an extended version of [42] where we introduced the methodology of constructing the state space $P \oplus R$ from a given service $R$ and the set of (reduced) fragments of a service $P$. In this article, we extend the results of [42] by providing complete proofs for the fragment reduction. In addition, we present experimental results for matching (i.e., model checking the constructed state space $P \oplus R$ for weak termination). Our results show that the state space $P \oplus R$ calculated from reduced fragments of $P$ is significantly smaller than the state space of the composition of $P$ and $R$.

We continue by providing our service model and other basic concepts in Sect. 2. In Sect. 3, we introduce the computation of fragments of a service $P$ and the construction of a transition system $P \oplus R$ given the fragments of $P$ and a service $R$. We present reduction rules to condense fragments in Sect. 4. Experimental results in Sect. 5 validate the applicability of our methodology. In Sect. 6, we present related work, and conclusions are drawn in Sect. 7.

2. Basic concepts

In this section, we describe our service model, state machines, simulation between service models and the notion of a partner.

2.1. Behavioral models for services

A service consists of a control structure describing its behavior and of an interface to communicate asynchronously with other services. An interface is a set of (input and output) channels. In order that two services can interact with each other, an input channel of one service has to be connected to an output channel of the other service. Asynchronous message passing means that communication is nonblocking; that is, after a service has sent a message, it can continue its execution and does not have to wait until this message is received. Furthermore, messages can ‘overtake’ each other; that is, the order in which the messages are sent is not necessarily the order in which they are received. In practice, there exist many languages to describe the behavior of services. Most prominent are WS-BPEL and BPMN. In this article, we abstract from the actual syntax of service description languages and use state machines as a basic formalism to model service behavior. For the labeling, we fix a universe $\mathcal{MC}$ of message channels with $\tau \notin \mathcal{MC}$.

Definition 2.1. (State machine)

A state machine $P = [Q, L^I, L^O, \delta, \hat{q}, W]$ consists of

- a countable set $Q$ of states,
- a set $L^I \subseteq \mathcal{MC}$ of input labels and a set $L^O \subseteq \mathcal{MC}$ of output labels with $L^I \cap L^O = \emptyset$; the internal label is denoted by $\tau \notin (L^I \cup L^O)$,
- a transition relation $\delta \subseteq Q \times (L^I \cup L^O \cup \{\tau\}) \times Q$ on states,
- an initial state $\hat{q} \in Q$, and
P is deterministic if and only if, for all \(q, q', q'' \in Q\), \(x \in L^I \cup L^O\), \([q, \tau, q'] \in \delta\) implies \(q = q'\) and \([q, x, q'] \in \delta\) implies \(q'' = q'\). P is a finite state machine if and only if \(Q\), \(L^I\), and \(L^O\) are finite.

The transition relation \(\delta\) reflects state changes of a state machine. For any two states \(q\) and \(q'\) and any action \(x \in (MC \cup \{\tau\})\), we write \(q \xrightarrow{x} q'\) if an \(x\)-labeled transition exists from \(q\) to \(q'\). We write \(q \xrightarrow{\tau}\) if there exists a \(q'\) such that \(q \xrightarrow{\tau} q'\). By \(q \xrightarrow{x} q'\), we represent that there exists a (possibly empty) sequence \(q \xrightarrow{x_1} \ldots \xrightarrow{x_n} q'\) of transitions from \(q\) to \(q'\) and say that \(q'\) is reachable from \(q\).

An \(x\)-labeled transition \([q, x, q'] \in \delta\) is a sending transition if \(x \in L^O\), a receiving transition if \(x \in L^I\), and an internal transition if \(x = \tau\). In the graphical representation of a state machine, we label a transition with \(!x\) if \(x \in L^O\) and \(?x\) if \(x \in L^I\).

In order that two services can interact with each other, they have to be composed; that is, an input channel of one service has to be connected to an output channel of the other service. Two state machines \(P\) and \(R\) are composable if they have disjoint sets of states, the input channels of \(P\) are the output channels of \(R\), and vice versa. The composition \(P \oplus R\) of composable state machines \(P\) and \(R\) introduces an internal message bag \(B\) (i.e., a multiset) of pending asynchronous messages that were sent by one state machine, but not yet received by the other one. That way, an \(x\)-labeled sending transition of \(P\) adds in \(P \oplus R\) one \(x\) element to the message bag \(B\). Correspondingly, a transition receiving an \(x\) removes an \(x\) from the message bag \(B\). Internal transitions remain as internal transitions in \(P \oplus R\). We assume that transitions in \(P \oplus R\) inherit their label from the originating transition in \(P\) and \(R\), respectively. In contrast to actual service models, however, a label in a composed system just serves as an annotation and is not meant to establish communication with a third service. The annotated labels will significantly simplify some considerations in the remainder.

We formalize composition of state machines in the following definition. Let \(\text{Bags}(MC)\) denote the set of all multisets over \(MC\), and let \([]\) denote the empty multiset. Addition of multisets is defined pointwise.

**Definition 2.2. (Composition of state machines)**

Two state machines \(P = [Q_P, L^I_P, L^O_P, \delta_P, \hat{q}_P, W_P]\) and \(R = [Q_R, L^I_R, L^O_R, \delta_R, \hat{q}_R, W_R]\) are composable if and only if \(Q_P \cap Q_R = \emptyset\), \(L^I_P = L^O_P\), and \(L^I_R = L^O_R\). For two composable state machines \(P\) and \(R\), their composition is the transition system \(P \oplus R = [Q, L^I, L^O, \delta, \hat{q}, W]\) defined as:

- \(Q = Q_P \times Q_R \times \text{Bags}(MC)\);
- \(L^I = L^I_P \cup L^I_R\) and \(L^O = L^O_P \cup L^O_R\);
- \(\delta \subseteq Q \times (L^I \cup L^O \cup \{\tau\}) \times Q\);
- \(\hat{q} = [\hat{q}_P, \hat{q}_R, []]\);
- \(W = W_P \times W_R \times [][]\);

such that the transition relation \(\delta\) contains for all \(B \in \text{Bags}(MC)\) the elements

- \([q_P, q_R, B], \tau, [q'_P, q_R, B]\) if and only if \([q_P, \tau, q'_P] \in \delta_P\);
- \([q_P, q_R, B], \tau, [q_P, q'_R, B]\) if and only if \([q_R, \tau, q'_R] \in \delta_R\).
Examples for service models are Bank, Cust, and $MP(Bank)$ in Figs. 1(a), 1(b), and 1(d), respectively. Figure 1(a) depicts the model Bank of an online bank service. Bank either sends a customer his annual statements (?as) or it requires the customer to make an appointment with his bank consultant (?req). It accepts additional information being sent by the customer (?i), but it always reminds him to make an appointment (?req). If the customer agrees on an appointment (?ap), Bank terminates. Figure 1(b) depicts a customer service Cust. Cust receives either the annual statements (?as) or a request for an appointment (?req). He always replies to such a request by sending some information to its customer consultant (ii) in the hope of receiving his annual statements eventually (?as). Bank and $MP(Bank)$ are deterministic; Cust is not—the state r0 has a $\tau$-labeled transition, which is not a self-loop. Bank and Cust are composable and Fig. 1(c) depicts their composition.

2.2. Simulation between state machines

For comparing two state machines $R_1$ and $R_2$, we use a simulation relation [26]. A simulation relation of $R_1$ by $R_2$ demands that every transition of $R_1$ can be mimicked by an equally-labeled transition of $R_2$. 
Definition 2.3. ((Bi-)Simulation relation)
Let \( R_1 = [Q_1, L^I, L^O, \delta_1, q_1, W_1] \) and \( R_2 = [Q_2, L^I, L^O, \delta_2, q_2, W_2] \) be state machines. A binary relation \( \varrho \subseteq Q_1 \times Q_2 \) is a simulation relation of \( R_1 \) by \( R_2 \) if and only if
- \([q_1, q_2] \in \varrho, \) and
- for all \([q_1, q_2] \in \varrho, a \in (L^I \cup L^O \cup \{\tau\}), q'_1 \in Q_1 \) such that \([q_1, a, q'_1] \in \delta_1, \) there exists a state \( q'_2 \in Q_2 \) such that \([q_2, a, q'_2] \in \delta_2 \) and \([q'_1, q'_2] \in \varrho. \)

If such a \( \varrho \) exists, we say that \( R_2 \) simulates \( R_1. \) If \( \varrho^{-1} \) is a simulation relation of \( R_2 \) by \( R_1 \) as well, then the two services are bisimilar [31] and \( \varrho \) a bisimulation relation. Let \( \varrho \) be a bisimulation relation between \( R_1 \) and \( R_2. \) If, for all \([q_1, q_2] \in \varrho, q_1 \in W_1 \) if and only if \( q_2 \in W_2, \) then \( \varrho \) respects final states.

We use a strong kind of simulation where even \( \tau \) is treated as a normal label. Like for finite state machines, we can also define a (bi-)simulation relation for two transition systems. As the definition is analogous, we do not show it.

There may exist several simulation relations of \( R_1 \) by \( R_2. \) Throughout this article, we shall always confine to minimal simulation relations where minimal refers to set inclusion. If \( R_2 \) is deterministic, then there exists a unique minimal simulation relation (if any simulation relation exists), because for each transition in \( R_1, \) there is a unique mimicking transition in \( R_2. \) The minimal simulation relation relates only reachable states of \( R_1 \) and \( R_2. \)

Definition 2.4. (Minimal simulation relation)
Let \( R_1 \) and \( R_2 \) be state machines and \( R_2 \) be deterministic. The minimal simulation relation \( \varrho \) of \( R_1 \) by \( R_2 \) is the smallest simulation relation of \( R_1 \) by \( R_2, \) i.e., \( \varrho \subseteq \varrho', \) for all simulation relations \( \varrho' \) of \( R_1 \) by \( R_2. \)

Example 2.2. In the example of Fig. 1, service MP(Bank) simulates service Cust, but Cust does not simulate MP(Bank). As a consequence, MP(Bank) and Cust are not bisimilar. As MP(Bank) is deterministic, there exists a minimal simulation relation \( \varrho \) of Cust by MP(Bank) with \( \varrho = \{[r_0, s_0], [r_1, s_0], [r_2, s_1], [r_3, s_2], [r_4, s_4], [r_2, s_5]\}. \)

2.3. Partners
The idea of SOC is to design complex services from smaller services. To this end, services have to be composed. Clearly, we are interested in service compositions where the involved services interact properly. In this article, proper interaction means that the service composition neither contains deadlocks nor livelocks. To decide proper interaction of service compositions, we restrict ourselves to finite state services (i.e., finite state machines). In this case, deadlocks and livelocks are terminal strongly connected components of a state machine which do not contain a final state.

Definition 2.5. ((Terminal) Strongly connected component, (T)SCC)
Let \( P = [Q, L^I, L^O, \delta, q, W] \) be a finite state machine. Two states \( q, q' \in Q \) of \( P \) are mutually reachable if and only if \( q \xrightarrow{\delta} q' \) and \( q' \xrightarrow{\delta} q. \) Mutual reachability is an equivalence relation on states of a finite state machine, and its equivalence classes are strongly connected components (SCCs). An SCC \( S \) is a terminal strongly connected component (TSCC) if and only if no state of another SCC is reachable from any state of \( S. \)
A terminal strongly connected component with a singleton nonfinal state without a self-loop is a deadlock; every other terminal strongly connected component, which does not contain a final state, is a livelock. Note that our definition of a deadlock differs from the standard definition in literature, as we discriminate between final states and deadlocks.

If a finite state machine neither contains a deadlock nor a livelock, it is weakly terminating. In the computation tree logic (CTL) [8], weak termination can be expressed as $\text{AG} \, \text{EF} \, (\bigvee_{q \in W} q)$; that is, from every state that is reachable from the initial state, it is always possible to reach a final state. Weak termination is a weaker property than (classical) soundness [1]. In addition to weak termination, soundness requires that each activity in a higher level model (sometimes referred to as structural transition) can be enabled.

A finite state machine $R$ is a partner of a finite state machine $P$ if and only if they are composable and the composition $P \oplus R$ of $P$ and $R$ is weakly terminating.

**Definition 2.6. (Weak termination, partner)**

Let $P$ and $R$ be two composable finite state machines. For $k \in \mathbb{N}$, the composition $P \oplus R = [Q_r, L_r, I_r, L_o, O_o, \delta, q_r, W]$ is $k$-weakly terminating if and only if, for all states $q = [q_p, q_r, B] \in Q$ with $q \xrightarrow{\tau} q'$ holds: (1) $\exists q' : q \xrightarrow{\tau} q'$ with $q' \in W$ and (2) $B(x) \leq k$, for all channels $x \in \mathcal{MC}$. If $P \oplus R$ is $k$-weakly terminating, then $R$ is a $k$-partner of $P$.

The introduction of the parameter $k$ establishes an artificial limit on the number of pending messages in a single channel. Technically, we use it to enforce (for any value of $k$) that the composition has a finite number of reachable states, which is a prerequisite for the algorithms studied in the sequel. Pragmatically, it could either represent a reasonable buffer size in the middleware—for example, the result of a static analysis of the communication behavior of a service—or be chosen sufficiently large. In the service discovery approach, we assume that the service provider fixes a $k$ when publishing its service $S$. In the remainder of this article, we assume an arbitrary, but fixed value of $k$ to be given, and we shall use terms, such as “weakly terminating” or “partner”, without preceding $k$.

From [41], we import a result concerning partners of finite state machines.

**Proposition 2.1. (Existence of a most permissive partner [41])**

For every finite state machine $P$, which has at least one partner, there exists a deterministic partner $MP(P)$ such that the following holds: If $R$ is a partner of $P$, then $MP(P)$ simulates $R$. Call $MP(P)$ a most permissive partner of $P$.

There may exist several most permissive partners of a finite state machine and by definition they simulate each other. Because they are deterministic, these simulation relations must be inverses of each other, thus implying a bisimulation relation. As a consequence, all most permissive partners are bisimilar. For this reason, the results of this article hold independently of the most permissive partner chosen. Hence, we use a functional notation $MP(P)$. In [41], we describe an algorithm for computing a most permissive partner $MP(P)$ of a given finite state machine $P$ and a given bound $k$ of pending messages in the composition $P \oplus MP(P)$.

As a most permissive partner $MP(P)$ must simulate partners of $P$ that may have arbitrary many $\tau$-transitions, every state of $MP(P)$ has a $\tau$-labeled self-loop. Note that this is only a technical construct. In an implementation these transitions can be left out while adjusting the simulation relation accordingly. Furthermore, every state of $MP(P)$ is a final state, thus ensuring that if any partner is in a final state, the corresponding related state of $MP(P)$ is a final state as well.
Once having found a most permissive partner, Proposition 2.1 gives us a necessary but unfortunately not a sufficient criterion for deciding whether a finite state machine \( R \) is a partner of a finite state machine \( P \): There must exist a simulation relation of \( R \) by the most permissive partner \( MP(P) \). As \( MP(P) \) is deterministic, the uniquely determined minimal simulation relation exists, too.

The existence of a minimal simulation relation provides us with the clue to our approach: If a most permissive partner \( MP(P) \) simulates every partner \( R \) of \( P \), the composition \( P \oplus R \) must follow those patterns, which are already present in \( P \oplus MP(P) \). To this end, we decompose the state space of \( P \oplus MP(P) \) into fragments. These fragments can then be connected to the actual composition \( P \oplus R \), where the minimal simulation relation of \( R \) by \( MP(P) \) determines the way in which fragments are connected.

**Example 2.3.** Service \( Cust \) is not a partner of service \( Bank \) in Fig. 1, because their composition may livelock if \( Bank \) sends message \( req \); see the four states forming a TSCC without a final state depicted bold in Fig. 1(c). Figure 1(d) depicts a most permissive partner \( MP(\text{Bank}) \) of service \( Bank \) (for \( k = 1 \)). Note that an infinite exchange of messages \( req \) and \( i \) is possible in the composition of \( Bank \) and \( MP(\text{Bank}) \), because weak termination only guarantees that a final state can be reached (if \( MP(\text{Bank}) \) sends \( !ap \)). State \( s5 \) is depicted only for technical purposes. Every edge to \( s5 \) shows a possible set of messages that \( MP(\text{Bank}) \) can receive, but that will never occur, because \( Bank \) cannot send them. Although \( MP(\text{Bank}) \) simulates \( Cust \) (see Fig. 1(b)), \( Cust \) is not a partner of \( Bank \). Hence, this example shows that the existence of a simulation relation of \( Cust \) by a most permissive partner of \( Bank \) is only a necessary but not a sufficient criterion for being a partner of \( Bank \).

### 3. Composing transition systems from fragments

In this section, we define the notion of a fragment and show how to compute fragments for a finite state machine \( P \). We then present how a transition system \( P \oplus R \) can be constructed given the fragments of \( P \) and a finite state machine \( R \).

#### 3.1. Fragments

Given a finite state machine \( P \) and a most permissive partner \( MP(P) \), the composition \( P \oplus MP(P) \) yields a transition system \( TS \) that consists of transitions of \( P \) and \( MP(P) \). We use this property to derive a canonical decomposition of \( TS \) into fragments. A fragment describes what \( P \) can do between two subsequent transitions of \( MP(P) \). The transitions of \( MP(P) \) link fragments. As \( P \) and \( MP(P) \) interact asynchronously, a transition of \( MP(P) \) may correspond to many transitions in the composed system. Consequently, we bundle these transitions in the composed system to a connection, which links two fragments. In the case of an internal transition \( \tau \) in \( MP(P) \), the connection is a bundle of self-loops, which links a fragment to itself. Otherwise, the linked fragments are different, because a noninternal transition has an impact on the pending messages which contribute to the state of the composed system.

**Example 3.1.** To illustrate the idea of fragments and connections, consider the transition system \( Bank \oplus MP(\text{Bank}) \) in Fig. 2. Dotted arcs denote transitions of \( Bank \); solid arcs denote transitions of \( MP(\text{Bank}) \) (and are hence transitions of connections). Each ellipse denotes a fragment of \( Bank \). Figure 2 illustrates that each connection either links two fragments or is a bundle of self-loops.
According to our approach, we shall construct the composition $P \oplus R$ of the finite state machines $P$ and $R$ from fragments of $P$. To this end, we formally define fragments and connections in terms of transition systems.

**Definition 3.1. (Fragment)**
A (state space) fragment $F = [V, L^I, L^O, E, \Omega]$ consists of

- a set $V$ of nodes,
- a set $L^I$ of input and a set $L^O$ of output transition labels with $\tau \notin L^I \cup L^O$,
- a set $E \subseteq V \times (L^I \cup L^O \cup \{\tau\}) \times V$ of (directed) labeled edges, and
- a set $\Omega \subseteq V$ of final nodes.

We do not define an initial state of a fragment, because a fragment may have several initial states. Instead, we introduce an initial state later on when we define the composition $P \oplus R$ that is calculated from fragments of $P$. The initial state shall be the initial state of $P \oplus MP(P)$.

When composing a state space from fragments, we may need several copies of one and the same fragment, because different states of $R$ may be related to the same state of $MP(P)$ and hence to the same fragment. To this end, we introduce fragment instances. Let some fixed set $FI$ denote the name space of all fragment instances.

**Definition 3.2. (Fragment instance)**
Let $n \in FI$. An instance $F(n)$ of a fragment $F = [V, L^I, L^O, E, \Omega]$ is built by renaming the constituents $v \in (V \cup \Omega)$ and $e \in E$ as follows: $v \mapsto [v, n]$, $e = [v_1, x, v_2] \land e \mapsto [[v_1, n], x, [v_2, n]]$.

Fragment instances are fragments again. For each fragment instance $n$, we use $n$ as an identifier to make all constituents disjoint from constituents of other fragment instances. Later on, the identifier of a fragment instance refers to a state of $R$. 

Figure 2. Dividing the state space of Bank $\oplus MP(Bank)$ into fragments.
For connecting fragments, more precisely, for connecting nodes of different fragment instances, we use the concept of *connections*.

**Definition 3.3. (Connection, instance)**

A connection $C$ from fragment $F_1 = [V_1, L^I, L^O, E_1, \Omega_1]$ to fragment $F_2 = [V_2, L^I, L^O, E_2, \Omega_2]$, with $V_1 \cap V_2 = \emptyset$, is a subset of $V_1 \times (L^I \cup L^O \cup \{\tau\}) \times V_2$. An *instance* $C(m, n)$ of a connection $C$ is defined as $C(m, n) = \{(v_1, m, x, v_2, n) \mid (v_1, x, v_2) \in C\}$, for $m, n \in FI$. Call an element of a fragment (instance) a *transition*.

If $C$ is a connection from fragment $F_1$ to fragment $F_2$, then $C(m, n)$ is a connection from fragment instance $F_1(m)$ to fragment instance $F_2(n)$.

Given a set of fragments and a set of connections we can build a transition system by connecting states of different fragments according to the connections.

**Definition 3.4. (Transition system of fragments)**

A set $F_1, \ldots, F_n$ of fragments with $F_i = [V_i, L^I, L^O, E_i, \Omega_i]$ for $i = 1, \ldots, n$ and pairwise disjoint sets of nodes, together with a set $C_1, \ldots, C_m$ of connections of $F_1, \ldots, F_n$ to $F_1, \ldots, F_n$ defines the labeled transition system $TS = [V, L^I, L^O, E]$ with

- a set $V = \bigcup_{k=1}^n V_k$ of states and
- a set $E = \bigcup_{i=1}^n E_i \cup \bigcup_{j=1}^m C_j$ of (directed) labeled transitions.

We do not define initial and final states of such a transition system, but we define these states later on when we consider fragments of a given finite state machine $P$.

Next, we show how to define the set of fragments and connections of a given finite state machine $P$.

### 3.2. Fragments and connections of a finite state machine $P$

In the following, we present how the sets $F(P)$ of fragments and $C(P)$ of connections for a given finite state machine $P$ can be computed. These sets provide those ingredients that are needed to calculate composed systems involving $P$.

To calculate the fragments and the connections of $P$, we construct the transition system $P \oplus MP(P)$ of $P$ and an arbitrarily chosen but fixed most permissive partner $MP(P)$ of $P$. As already mentioned, we can compute $MP(P)$ from $P$ and a fixed bound $k$ that limits the number of pending messages. Our results do not depend on the choice of $MP(P)$. As $P$ and $MP(P)$ are finite, we can indeed compute the sets of fragments and connections.

**Definition 3.5. ($F(P), C(P)$)**

Let $MP(P) = [Q, L^I, L^O, \delta, \hat{q}, W]$ be a most permissive partner of $P = [Q_P, L^I_P, L^O_P, \delta_P, q_P, W_P]$. Define the set $F(P) = \{F_q \mid q \in Q\}$ of fragments of $P$ with $F_q = [V, L^I_P, L^O_P, E, \Omega]$,

- $V = \{[q_P, q, B] \mid [q_P, q, B] \in Q_P \oplus MP(P)\}$,
- $E = \{[[q_P, q, B], x, [q_P, q, B']] \mid [[q_P, q, B], x, [q_P, q, B']] \in \delta_P \oplus MP(P) \land [q_P, x, q_P'] \in \delta_P\}$, and
- $\Omega = \{[q_P, q, B] \mid [q_P, q, B] \in V \land q_P \in W_P \land q \in W \land B = []\}$. 
Figure 3. (a) The sets of all fragments and (b) all connections (except for the \( \tau \)-loops) of Bank in Fig. 1(a). Dotted lines denote fragment internal transitions and solid lines connections.

Define the set \( C(P) = \{ C[q, x, q'] | [q, x, q'] \in \delta \} \) of connections of \( P \), with \( C[q, x, q'] = \{ [[qP, q, B], x, [qP, q', B']], [[qP, q, B], x, [qP, q', B']] | (q, x, q') \in \delta_{P \oplus MP(P)} \land [q, x, q'] \in \delta \} \).

Definition 3.5 builds the composition \( P \oplus MP(P) \). For each state \( q \in Q \) of \( MP(P) \), there is a fragment \( F_q = [V, L_P^I, L_P^O, E, \Omega] \). Fragment \( F_q \) describes the behavior of \( P \) while \( MP(P) \) is in state \( q \). This behavior yields the sets \( V \) of nodes and \( E \) of edges of \( F_q \). Each transition of \( F_q \) is a transition of \( P \). Final states of fragment \( F_q \) are those states of \( F_q \), where \( P \) and \( MP(P) \) are in a final state and \( B \) is the empty multiset. A connection \( C[q, x, q'] \) describes a set of transitions in the composed system \( P \oplus MP(P) \) that originates from a single transition \( [q, x, q'] \) in \( MP(P) \). As \( MP(P) \) is by Proposition 2.1 deterministic, connections bundle either \( \tau \)-labeled self-loops, sending transitions or receiving transitions linking different fragments. Note that \( F_q \) may be empty for some state \( q \in Q \) that is not reachable in the composition \( P \oplus MP(P) \). Accordingly, we may have empty connections resulting from transitions to state \( q \) which do not exist in \( P \oplus MP(P) \). This fragment and also the empty connections do not play any role in our approach and thus shall not be shown in the graphics.

Recall that due to the introduced bound \( k \) of pending messages, \( F(P) \) and \( C(P) \) are finite.

Example 3.2. From the composition Bank \( \oplus MP(Bank) \) in Fig. 2, we calculate the fragments \( F(Bank) \) and the connections \( C(Bank) \) of the online bank service Bank; see Fig. 3. For each of the five reachable states \( s0, s1, s2, s3, \) and \( s4 \) of the most permissive partner \( MP(Bank) \), there is one fragment in Fig. 3(a). For instance, the fragment for the initial state \( s0 \) is defined as \( F_{s0} = \{ [v0, v1, v2], L_P^I, L_P^O, \{ [v0, las, v1], [v0, req, v2] \}, \emptyset \} \). Note that \( F(Bank) \) also contains a fragment \( F_{s5} \), but \( F_{s5} \) is unconnected and does not contain any node, because state \( s5 \) is not reachable in Bank \( \oplus MP(Bank) \). Connection \( C[s0, ?as, s1] \) is the singleton set \{ \[v1, ?as, v3]\} , where \( v0 \) relabels state \{p0, s0, []\}, \( v1 \) relabels state \{p1, s0, [as]\}, and so on. All connections (except for the self-loops) are singleton sets.

Next, we show how we can construct a transition system of \( P \oplus R \) given a finite state machine \( P \) and the set of fragments \( F(P) \) and connections \( C(P) \) of \( P \).

3.3. Composing a transition system from fragments

Throughout this section, we fix a finite state machine \( P \). Let \( MP(P) \) be the most permissive partner of \( P \) that has been used for constructing the sets \( F(P) \) of fragments and \( C(P) \) of connections of \( P \).
By Proposition 2.1, a minimal simulation relation of a finite state machine $R$ by $MP(P)$ is a necessary condition for $R$ being a partner of $P$. That means, if $MP(P)$ does not simulate $R$, then $R$ is not a partner of $P$ (with respect to the given message bound $k$) and there is no use in constructing a transition system $TS$ from $R$, $F(P)$, and $C(P)$ that reflects $P \oplus R$. For this reason, we may assume existence of a minimal simulation relation $\varrho \subseteq Q_R \times Q_{MP(P)}$ when constructing $TS$ from $F(P)$ and $C(P)$. The existence of $\varrho$ is checked during the construction of $TS$ by relating states of $R$ to fragments of $F(P)$. To this end, we apply a depth-first search through the state space of $R$. As the minimal simulation relation is unique, there is for each transition in $R$ only one continuation in $MP(P)$. As a consequence, we can check the simulation relation during this depth-first search. The time and space required for finding $\varrho$ is linear in the product of the number of states and edges of $MP(P)$ (i.e., the number of fragments and connections) and $R$.

We build the transition system $TS$ by gluing fragments and connections of $P$. For constructing $TS$, we have to define its set $F(TS)$ of fragments and $C(TS)$ of connections. We add, for each $[q_R,q] \in \varrho$, a fragment $F_q$ to the set $F(TS)$. More precisely, as $q_R$ might not be the only state of $R$ that is related to $q$, we add a fragment instance $F_q(q_R)$. Accordingly, the set $C(TS)$ is determined by the connections of $P$ and the fragments $F_q$ of $F(TS)$, where state $q$ is used in $\varrho$.

**Definition 3.6. (Construction of $TS$)**

For a finite state machine $P$ let $MP(P) = [Q,R^I,L^O,\delta,q^0,W]$ be the most permissive partner of $P$ that has been used for constructing the sets $F(P)$ of fragments and $C(P)$ of connections of $P$. Let $R = [Q_R,R^I,L^O,\delta_R,q_R,W_R]$ be a finite state machine, and let $\varrho \subseteq Q_R \times Q$ be the minimal simulation relation of $R$ by $MP(P)$. Compose transition system $TS$ from the following fragments and connections of $P$:

- $F(TS) = \{ F_q(q_R) \mid [q_R,q] \in \varrho \}$,
- $C(TS) = \{ C[q,x,q'](q_R,q') \mid [q_R,q],[q'_R,q'] \in \varrho \land [q_R,x,q'_R] \in \delta_R \land [q,x,q'] \in \delta \}$.

Let the initial state of $TS$ be the initial state of $P \oplus MP(P)$ in fragment instance $F_q(q_R)$. Let the set of final states of $TS$ consist of all final nodes occurring in those fragment instances $F_q(q_R)$ where $q_R \in W_R$.

Fragment instance $F_q(q_R)$ is definitely contained in $F(TS)$ and contains an instance of the initial state of $P \oplus MP(P)$; otherwise, there would not exist a minimal simulation relation of $R$ by $MP(P)$. We use this state as the initial state of $TS$. Final states of $TS$ are the final states of all fragment instances $F_q(q_R)$ where $q_R$ is a final state in $R$.

**Example 3.3.** In our example, we construct the state space of $Bank \oplus Cust$ from the fragments and connections of the online bank service $Bank$ (see Fig. 3) and the customer service $Cust$ (see Fig. 1(b)). We have $[r0,s0],[r1,s0] \in \varrho$. Hence, we add two instances of fragment $F_{s0}$, $F_{s0}(r0)$ and $F_{s0}(r1)$, to $F(TS)$. As we have transition $[r0,\tau,r1] \in \delta_{Cust}$ in $Cust$, we add connection $C_{s0,s0}(r0,r1)$ to $C(TS)$. Next, we add fragment $F_{s1}(r2)$ to $F(TS)$, because $[r2,s1] \in \varrho$. From transition $[r1,?as,r2] \in \delta_{Cust}$ in $Cust$ and from connection $[s0,?as,s1] \in \delta_{MP(Bank)}$ in $MP(Bank)$ we conclude that connection $C_{[s0,?as,s1]}(r1,r2)$ has to be added to $C(TS)$, and so on. Figure 4 shows the resulting transition system $TS$. Note that $[s4,s4] \in \varrho$ and presence of transition $[s4,?as,r2] \in \delta_{Cust}$ (see Fig. 1(b)) yields $[r2,s5] \in \varrho$. However, as $s5$ is not reachable in $MP(Bank)$, there is no $?as$-labeled transition leaving $F_{s4}(r4)$. Transition system
As the main result of this section, we prove that the constructed transition system $TS$ indeed reflects $P \oplus R$; that is, $TS$ and $P \oplus R$ are bisimilar. Intuitively, a stronger result than a bisimulation—for example, an isomorphism—cannot be achieved due to unfoldings of cycles.

**Theorem 3.1.** Let $R$ be a partner of $P$, and let $TS$ be as defined previously. Then there exists a bisimulation relation that respects final states between $TS$ and $P \oplus R$.

**Proof:**
As $R$ is a partner of $P$, there exists a minimal simulation relation $\rho$ of $R$ by $MP(P)$ that is used for constructing $TS$. We claim that the following relation $\rho^* \subseteq Q_{P \oplus R} \times Q_{TS}$ is the required bisimulation:

$$\rho^* = \{(\langle q_P, q_R, \mathcal{B} \rangle, \langle [q_P, q, \mathcal{B}], [q_R] \rangle) \mid [q_P, q, \mathcal{B}], [q_R] \in \rho, \text{ or } \langle [q_P, q, \mathcal{B}], [q_R] \rangle \in F_q(q_R)\}$$

By construction of $TS$, for every pair $[q_R, q] \in \rho$, $F_q(q_R) \in \mathcal{F}(TS)$; so $\rho^*$ is defined. The initial state of $P \oplus R$ is $\{\hat{q}_P, \hat{q}_R, []\}$, the initial state of $TS$ is $\{[\hat{q}_P, \hat{q}, []], [\hat{q}_R]\}$, so $\rho^*$ relates the initial states of the considered systems.

Let $\langle [q_P, q_R, \mathcal{B}], [q_P, q, \mathcal{B}], [q_R] \rangle \in \rho^*$. A transition in $P \oplus R$ originates either from a transition in $P$ or from a transition in $R$. A transition of $P$ leads to some state $[q'_P, q_R, \mathcal{B'}]$ in $P \oplus R$. Obviously, the same transition is possible in state $[q_P, q, \mathcal{B}]$ of the composition $P \oplus MP(P)$ as well, leading to $[q'_P, q, \mathcal{B'}]$ there. Both states and the corresponding transition between them are, thus, part of fragment $F_q$, which proves that a transition from $[q_P, q, \mathcal{B}], [q_R]$ to $[q'_P, q, \mathcal{B'}, q_R]$ with the same label is present in $TS$. Analogously, a transition internal to a fragment (which again stems from a transition in $P$) used in $TS$ can be mimicked in $P \oplus R$. 
Consider now a transition from state \([q_P, q_R, B]\) to state \([q'_P, q'_R, B']\) of \(P \oplus R\) that originates from a transition \([q_R, x, q'_R]\) in \(R\). As \(q\) is a simulation, there is a transition \([q, x, q']\) with \([q'_R, q'] \in \rho\), for some \(q'\). Both transitions change message bag \(B\) in the same way, because they carry the same label. Fragment \(F_q\) and connection instance \(C_{[q, x, q']}(q_R, q'_R)\) have been included in the construction of \(TS\). This connection instance contains the required transition from \([q_P, q, B], q_R]\) with \(x\) to \([q_P, q', B'], q'_R]\) in \(TS\). The other way round, consider an \(x\)-labeled transition in \(TS\) from \([q_P, q, B], q_R]\) to some state \([q_P, q', B'], q'_R]\) that is part of an included connection instance \(C_{[q, x, q']}(q_R, q'_R)\). By construction of \(TS\), we have \([q_R, x, q'_R] \in \delta_r\). As enabledness of a transition in \(P \oplus R\) and its effect on \(B\) just depend on \(B\) and \(x\), transition \([q_P, q_R, B], x, [q_P, q'_R, B']\) must be present in \(P \oplus R\).

Let \([q_P, q_R, B]\) be a final state in \(P \oplus R\)—that is, \(q_P \in W_P\), \(q_R \in W_R\), and \(B = []\). This means that, for any \(q\), \([q_P, q, B], q]\) is a final state in \(F_q\) and, by construction of \(TS\), \([q_P, q, B], q_R]\) is final in \(TS\) (recall that every state in \(MP(P)\) is a final state). The other way round, a state in \(TS\) is a final state according to Definition 3.6 if \(q_R\) is a final state and the state of the fragment instantiated with \(q_R\) is final. By construction of fragments, this implies \(q_P \in W_P\) and \(B = []\), so \([q_P, q_R, B]\) is a final state in \(P \oplus R\).

Bisimilar systems are indistinguishable for any formula of the temporal logic CTL that uses atomic propositions which are preserved by the considered bisimulation relation (see [10], for instance). As weak termination can be expressed as \(AG \bigvee_{q \in W} q\) in this logic, Theorem 3.1 implies the following corollary.

**Corollary 3.1.** \(P \oplus R \) is weakly terminating if and only if \(TS\) is.

Corollary 3.1 enables us to verify \(TS\) instead of the state space of \(P \oplus R\), because the bisimulation relation between both transition systems guarantees that if weak termination holds in \(TS\), it also holds in the state space of \(P \oplus R\); and if weak termination does not hold in \(TS\), it also does not hold in the composition \(P \oplus R\).

We can directly apply the results presented in this section to the setting of service discovery. A service provider publishes its service \(P\) in a service registry. The service broker then computes the fragments and connections of \(P\). Upon a “find” request of a service \(R\), the broker checks first whether \(P\) and \(R\) are composable. In a second step, the broker computes a transition system \(TS\) by composing fragments of \(P\) and verifies it for weak termination.

Our goal is to speed up a “find” request. This can be achieved if the computed transition system is significantly smaller than the state space of \(P \oplus R\). In the next section, we show a way to reduce the size of the calculated fragments. That way, we can compute a transition system \(TS'\), which is significantly smaller than \(TS\), while guaranteeing that \(TS'\) is weakly terminating if and only if \(TS\) is.

### 4. Fragment reduction

In this section, we present techniques that reduce the size of the calculated fragments and the number of transitions of each connection. That way, we achieve the actual reduction of computational effort that we would like to have at the time of a “find” request. To make this reduction applicable in our setting, we have to guarantee that the transition system computed from a finite state machine \(R\) and the reduced fragments of a finite state machine \(P\) is weakly terminating if and only if the transition system composed
The reduction we are using is different from usual state space reduction known from explicit state space verification [10, 39]. Whereas traditional state space reduction is typically applied during the state space construction, we can apply our techniques after the state space construction. This is not to say that usual state space reduction techniques like the symmetry method [9, 12] or partial-order reduction [15, 32, 37] are out of scope in our setting. The problem is that application of these techniques must not interfere with the procedure of composing fragments to a transition system $TS$ as described in the previous section.

To this end, we concentrate on an *a posteriori state space reduction* that can be applied immediately after constructing the fragments. The reductions are local to a fragment or local to a connection. A reduction local to a fragment corresponds to a state space reduction in every instance of the fragment present in the state space. A reduction local to a connection impacts the transitions of this connection and the source and target fragments. It is the same as a reduction in every instance of the connection in the generated state space. Each fragment of $P$ is reduced only once, but the outcome of the reduction is available in every transition system that is constructed from the reduced fragments of $P$.

We continue by first collapsing strongly connected components inside fragments to single states. In a second step, we define dedicated reduction rules that reduce the number of states of the fragments. Hereby, we neither remove a fragment nor change the connectivity between fragments (i.e., we do not remove connections). We may, however, remove redundant transitions in connections.

### 4.1. Strongly connected component based fragment reduction

Deadlocks and livelocks are terminal strongly connected components which do not contain a final state. If this component is a singleton state without a self-loop, we have a deadlock, otherwise a livelock.

In this light, collapsing strongly connected nodes inside a fragment to single nodes is an obvious reduction. It may turn a livelock into a deadlock, but the reduced system is weakly terminating if and only if the original system is.

**Definition 4.1. (SCC reduction)**

Let $\mathcal{F}$ be a set of fragments and $\mathcal{C}$ be a set of connections. The reduced sets $\mathcal{F}'$ and $\mathcal{C}'$ are defined as follows. For each fragment $F = [V, L^I, L^O, E, \Omega]$, let $\equiv_F$ be the equivalence on $V$ where $q \equiv_F q'$ if and only if $q$ and $q'$ are mutually reachable using edges in $E$. Let $F' = [V', L^I, L^O, E', \Omega']$ be defined by

- $V' = [V]_{\equiv_F}$;
- $E' = \{[[q]_{\equiv_F}, [q']_{\equiv_F}] \mid \exists q_1 \in [q]_{\equiv_F}, q'_1 \in [q']_{\equiv_F} : (q_1, q'_1) \in E\}$; and
- $\Omega' = \{[q]_{\equiv_F} \mid [q]_{\equiv_F} \cap \Omega \neq \emptyset\}$.

Let $\mathcal{F}' = \{F' \mid F \in \mathcal{F}\}$. For a connection $C$ from fragment $F_1$ to fragment $F_2$, let $[[q]_{\equiv_{F_1}}, [q']_{\equiv_{F_2}}] \in C'$ if and only if there exist $q_1 \in [q]_{\equiv_{F_1}}$ and $q'_1 \in [q']_{\equiv_{F_2}}$ with $(q_1, q'_1) \in C$. Let $\mathcal{C}' = \{C' \mid C \in \mathcal{C}\}$.
For each SCC, we only store one representative. Accordingly, we replace each transition from a state of an SCC by a transition from the representative of this SCC. Likewise, we replace each transition to a state of an SCC by a transition to the representative of this SCC. Note that we can ignore the transition labels in fragments. These labels were only necessary to construct the fragments and connections of a finite state machine $P$.

Given the sets $F$ of fragments and $C$ of connections, we calculate for each fragment $F \in F$ an SCC-reduced fragment $F'$ and for each connection $C \in C$ an SCC-reduced connection $C'$. It suffices to store only the SCC-reduced sets $F'$ and $C'$ rather than $F$ and $C$, because we can construct an SCC-reduced transition system $TS'$ from $F'$ and $C'$ by applying Definition 3.6. To this end, we replace each fragment $F \in F(TS)$ by $F'$ and each connection $C \in C(TS)$ by $C'$. As all replacements are executed simultaneously and consistently, the reduction is well defined. Furthermore, all SCCs of single fragments are strongly connected in every transition system composed of the fragments. Mutual reachability of states in different SCCs is left invariant. As a consequence, the following lemma holds.

**Lemma 4.1.** A transition system composed of fragments in $F$ and connections in $C$ is weakly terminating if and only if the corresponding reduced transition system is where, for every fragment $F \in F$, $F'$ is used instead, and every connection $C \in C$ is replaced by the corresponding $C'$.

SCC reduction on fragments is applicable if the given service $P$ has cycles in its internal activities. As none of the fragments of Fig. 3(a) has an internal cycle, SCC reduction does not affect these fragments. The previous lemma shows that a broker only needs to store SCC-reduced fragments and connections of a published service.

### 4.2. Rule-based reduction

Given an SCC-reduced set of fragments and connections, we can further reduce these fragments and connections using local state space transformations. We present a set of applicable rules that have been adapted from the deadlock-preserving reduction rules by Juan et al. [18]. Most of the rules in [18] indeed do preserve livelocks. Thus, the adaptation was straightforward. The actual challenge in adapting the rules was to ensure that the rules can be applied in any transition system that can be constructed using the considered fragments and connections.

For simplifying presentation, we introduce some notation. For a state $q$, let $F_q = [V_q, L^I, L^O, E_q, \Omega_q]$ be the fragment where $q \in V_q$. For an edge $[q, x, q']$, let $E_{[q, x, q']} = E_q$ if $F_q = F_{q'}$ and $[q, x, q'] \in E_q$, and let $E_{[q, x, q']} = C_{[q, x, q']}$, otherwise. With this notation, we are able to refer to the fragment that some state belongs to. We can further refer to transitions internal to a fragment and to transitions in connections in a single notation. For a set $T$ of transitions, $q'$ is reachable by $T$ from $q$, denoted by $(q \xrightarrow{T} *q')$, if and only if $[q, q']$ is in the reflexive and transitive closure of $T$. For a state $q$, let $\bullet q = \{q' \mid \exists x \in (L^I \cup L^O \cup \{\tau\}) : [q', x, q] \text{ appears in any fragment or connection}\}$ and $q^\bullet = \{q' \mid \exists x \in (L^I \cup L^O \cup \{\tau\}) : [q, x, q'] \text{ appears in any fragment or connection}\}$ denote the set of predecessor and successor states of $q$, respectively.

We shall now present the actual reduction rules. Again, we can ignore the transition labels in fragments, because these labels are not needed when checking a transition system constructed from fragments for weak termination.
4.2.1. Rule Transitive Reduction

The transitive reduction rule [4] aims at removing a transition \([q, x, q']\) if and only if there is another transition sequence \(T'\) from state \(q\) to state \(q'\).

**Constraint:** There is a transition \([q, x, q']\) where \(q \xrightarrow{(E \cup E_{\{q,x,q'\}} \cup \{q,x,q'\})} *q'\).

**Application:** \(E_{[q,x,q']} := E_{[q,x,q']} \setminus \{[q,x,q']\}\).

**Example 4.1.** Figure 5 illustrates the transitive reduction rule. It shows that the rule can be applied internally to a fragment (see Fig. 5(a)) but also to remove redundant transitions in connections (see Fig. 5(b)).

The following lemma proves that we can apply this reduction rule in our setting.

**Lemma 4.2.** A transition system composed of fragments in \(\mathcal{F}\) and connections in \(\mathcal{C}\) is weakly terminating if and only if the corresponding reduced transition system is where, every fragment \(F \in \mathcal{F}\) is replaced by \(F'\) and every connection \(C \in \mathcal{C}\) is replaced by \(C'\) which result from applying Rule Transitive Reduction.

**Proof:**
Let \(TS\) be constructed of fragments and assume that it contains an instance of \([q,x,q']\). If this transition is internal to a fragment \(F\), then the complete transition sequence required in the constraint is internal to the same fragment. So \(q'\) is still reachable from \(q\) in \(F'\). If \([q,x,q']\) belongs to a connection \(C\) from fragment \(F_1\) to fragment \(F_2\), then fitting instances of both fragments and the connection are present in \(TS\), and the constraint again assures reachability of \(q'\) from \(q\) by a transition of the same connection in \(C'\). Thus, mutual reachability of states is left invariant, which is sufficient for asserting preservation of weak termination.

4.2.2. Rule Sequence

The intuition behind this rule is to reduce sequences of states and synchronization states to reducing all paths to a final state in a transition system \(TS\). We can achieve this by reducing within a fragment the paths to deadlock states or final states. A nonfinal state \(q\) can be removed if either all predecessor states of \(q\) are in the same fragment as \(q\) or all successor states of \(q\) are in the same fragment as \(q\).
Constraint: There is a nonfinal state \( q \) such that there is a transition \([q, z, q''] \in E_q\) (i.e., \( q \) is not a deadlock in its own fragment), \( q \) is not the initial state of \( \mathcal{F}(TS) \), and either \( q \times (L^I \cup L^O \cup \{\tau\}) \subseteq E_q \) or \( \{q\} \times (L^I \cup L^O \cup \{\tau\}) \times q'' \subseteq E_q \).

Application: (1) Add a transition from every predecessor state of \( q \) to each successor state of \( q \), (2) remove all transitions from state \( q \), (3) remove all transitions to state \( q \), and (4) finally remove state \( q \).

Formally, let \( q \neq q' \) and

1. For all \( q' \in \mathcal{L} \), all \( x \in (L^I \cup L^O \cup \{\tau\}) \), and all \( q'' \in \mathcal{L} \) such that \([q', x, q], E_{[q', x, q']} := E_{[q', x, q']} \cup \{[q', x, q'']\};
2. For all \( q' \in \mathcal{L} \) and all \( x \in (L^I \cup L^O \cup \{\tau\}) \) such that \([q, x, q'], E_{[q, x, q']} := E_{[q, x, q']} \backslash \{[q, x, q']\};\n3. For all \( q' \in \mathcal{L} \) and all \( x \in (L^I \cup L^O \cup \{\tau\}) \) such that \([q', x, q], E_{[q', x, q]} := E_{[q', x, q]} \backslash \{[q', x, q]\};\n4. \( V_q := V_q \backslash \{q\}\).

Example 4.2. Figure 6 shows an example for each of the two types of this rule. State \( q \) is the state that is removed.

The following lemma proves that the reduction rule presented above is applicable in our setting.

Lemma 4.3. A transition system composed of fragments in \( \mathcal{F} \) and connections in \( \mathcal{C} \) is weakly terminating if and only if the corresponding reduced transition system is where, every fragment \( F \in \mathcal{F} \) is replaced by \( F'' \) and every connection \( C \in \mathcal{C} \) is replaced by \( C' \) which result from applying Rule Sequence.
Proof: Let $TS$ be a transition system built from unreduced fragments, and let $F_q$ be the fragment to be changed to $F'_q$ by the reduction rule. We prove both forms of this rule. First, we show for an arbitrary transition system that it is weakly terminating if and only if it is after the reduction. Afterward, we show that this property holds for any transition system that can be constructed from fragments.

Suppose the unreduced $TS$ is weakly terminating. Thus, from every reachable state in $TS$, there is a path to a final state. We have to show that the transition system $TS'$ after replacing $F_q$ by $F'_q$ is weakly terminating, too. Consider reachable state $q_1$ in $TS'$. In $TS$, there is a path from $q_1$ to a final state. If that path does not pass through $q$, the same path can be taken in $TS'$. Otherwise, this path passes through some $q' \in q$ and a $q'' \in q^*$. Now, $TS'$ contains a bypass transition $[q', x, q''']$, for all $q', q''$, and the only removed state is $q$. Clearly, reachability remains invariant by adding bypass transition $[q', x, q''']$. As $q$ is by assumption not a final state, the path can still reach a final state in $TS'$. Thus, $TS'$ is weakly terminating.

Suppose now the unreduced transition system $TS$ is not weakly terminating. We have to show that the reduced transition system $TS'$ does not weakly terminate either. Consider a livelock or a path to a deadlock in $TS$. If the livelock or the deadlock path does not pass through state $q$, it also exists in $TS'$. Assume now that the livelock or the deadlock path passes through state $q$. By assumption, $q$ is not a deadlock and hence the bypass transition $[q', x, q''']$ can be constructed in $TS'$. As all states reachable from $q$ in $TS$ can be reached from any predecessor state of $q$ via a transition in $TS'$, we conclude that $TS'$ is not weakly terminating either.

It remains to show that these properties hold for any unreduced transition system $TS$. Clearly, instances of manipulated connections $C'$ are only present in $TS'$ with instances of $F'_q$. Consider again a path through some $q' \in q$ and a $q'' \in q^*$ in $TS$. By our constraint, at least one of $[q', x, q]$ and $[q, y, q''']$ is internal to $F_q$. If one of these transitions is outside the fragment $F_q$, the bypass transition $[q', x, q''']$ (respectively $[q', y, q''']$) in $F'_q$ is a transition of the same connection. Thus, an instance of $[q', x, q''']$ (respectively $[q', y, q''']$) is present in $TS'$ if and only if transitions $[q', x, q]$ and $[q, y, q''']$ are present in $TS$. So reachability remains invariant in any transition system. By the same argumentation as before, we conclude that if $TS$ weakly terminates so does $TS'$ and if $TS$ does not weakly terminate, then $TS'$ does not weakly terminate either.

Example 4.3. Applying this rule to the fragments of our example (see Fig. 3) results in removing node $v5$ from fragment $F_{s3}$ and nodes $v7$ and $v8$ from fragment $F_{s4}$. Accordingly, we have to change connections $C_{s2,[lap,s3]} = \{[v4,lap,v5]\}$ to $\{[v4,lap,v6]\}$ and $C_{s2,[li,s4]} = \{[v4,li,v7]\}$ to $\{[v4,li,v9]\}$.

The example shows that this rule is a rather powerful reduction technique.

4.2.3. Rule Equivalent States

With this rule, we aim at detecting states $q$ and $q'$ of the same fragment that share the same successors. If $q$ and $q'$ are both either final states or no final states, then they can be merged while preserving weak termination.

Constraint: There are states $q$ and $q'$ such that $F_q = F_q'$, $q \in \Omega_q$ if and only if $q' \in \Omega_q$, and, for all fragment-internal edges $E^*$, $[q, x, q] \in E^*$ if and only if $[q', x, q'] \in E^*$ and for all connections $E^*$, $[q, x, q'] \in E^*$ if and only if $[q', x, q'] \in E^*$.
Application: (1) Redirect every transition from a predecessor of state \( q \) to \( q' \), (2) remove all transitions from \( q \) to its successor states, and finally (3) remove \( q \). Formally,

1. For all \( q^* \in \bullet q \) and all \( x \in (L^I \cup L^O \cup \{\tau\}) \) such that \([q^*, x, q], E[q^*, x, q'] := E[q^*, x, q'] \cup \{[q^*, x, q']\}\)
   and \( E[q^*, x, q] := E[q^*, x, q] \setminus \{[q^*, x, q]\}\);

2. For all \( q'' \in q^* \) and all \( x \in (L^I \cup L^O \cup \{\tau\}) \) such that \([q, x, q''], E[q, x, q''] := E[q, x, q''] \setminus \{[q, x, q'']\}\);

3. \( V_q := V_q \setminus \{q\}\).

Example 4.4. Figure 7 illustrates this reduction rule. In Fig. 7(a), the removal of state \( q \) is local to the fragment of \( q \). The example in Fig. 7(b) shows that also surrounding fragments may be involved, and hence also transitions in connections have to be redirected or removed.

The following lemma justifies the applicability of this reduction rule in our setting.

Lemma 4.4. A transition system composed of fragments in \( F \) and connections in \( C \) is weakly terminating if and only if the corresponding reduced transition system is where, every fragment \( F \in F \) is replaced by \( F' \) and every connection \( C \in C \) is replaced by \( C' \) which result from applying Rule Equivalent States.

Proof:
Let \( TS \) be any transition system built from unreduced fragments, and let \( F_q \) be the fragment to be changed to \( F'_q \) by the reduction rule. This fragment contains states \( q \) and \( q' \) in its unreduced version. We prove both forms of this rule. First, we show for any \( TS \) that it is weakly terminating if and only if it
is after the reduction. Afterward, we show that this property holds for any transition system that can be constructed from fragments.

Suppose the unreduced $TS$ is weakly terminating. We have to show that the transition system $TS'$ after replacing $F_q$ by $F'_q$ is weakly terminating, too. Consider fragment $F_q$. The constraints assure that $q$ and $q'$ have the same successors. Thus, redirection of all transitions $[q^*, x, q]$ to $[q^*, x, q']$ in $F'_q$ guarantees that reachability remains invariant in the reduced transition system $TS'$, for all states that reach $q$ in $TS$. If $q$ is a final state, so is $q'$ (follows from the constraint). Hence, reachability of a final state remains invariant in $TS'$, for all states that reach $q$. Thus, $TS'$ is weakly terminating.

Suppose now the unreduced transition system $TS$ is not weakly terminating. We have to show that the reduced transition system $TS'$ does not weakly terminate either. Consider a livelock or a path to a deadlock in $TS'$ that passes through $q$. As the reduction preserves reachability and final states, this livelock or deadlock path is also present in $TS'$. Hence, $TS'$ does not weakly terminate either.

We continue by proving that these properties hold for any transition system. The redirected transitions appear in the same fragment or connection as the original one. So replacing $F_q$ by the reduced fragment $F'_q$ is independent of the actual construction of any transition system. Consider now the effect of the reduction on the successors of $q$ and $q'$. If all successors $q^*$ appear in $F_q$, the removal of transitions $[q, x, q^*]$ is guaranteed to be independent of the actual construction of any transition system. Suppose now there is a transition $[q, x, q^*]$, and $q^*$ is not a state of $F_q$. The constraint assures the existence of transition $[q', x, q^*]$, which is also a transition of the same connection. Hence, removing transition $[q, x, q^*]$ affects only one connection, which is because of the presence of $q'$ in the reduced fragment $F'_q$ still present in the reduced transition system $TS'$. Hence, replacing $F_q$ by $F'_q$ is independent of the actual construction of any transition system. By the same argumentation as before, we conclude that if $TS$ weakly terminates so does $TS'$ and if $TS$ does not weakly terminate, then $TS'$ does not weakly terminate either.

From Lemma 4.2, from Lemma 4.3, and from Lemma 4.4 we conclude that we can apply the three reduction rules in any order, and any transition system composed of the reduced fragments is weakly terminating if and only if the transition system composed of the unreduced fragments is.

**Corollary 4.1.** Let $TS'$ result from applying any sequence of the reduction rules as described previously to $TS$. Then, $TS$ is weakly terminating if and only if $TS'$ is weakly terminating.

The corollary states that we can reduce fragments of any finite state machine $P$ by applying the three reduction rules in any order, and every transition system $TS$ computed from a finite state machine $R$ and the original fragments is weakly terminating if and only if the transition system $TS'$ computed from reduced fragments is. The bisimulation between $TS$ and the composition $P \oplus R$ (see Theorem 3.1) and Corollary 4.1 justify the abstraction approach, which is formalized in the following theorem.

**Theorem 4.1.** For any two composable finite state machines $P$ and $R$ such that $R$ is simulated by a most permissive partner $MP(P)$ of $P$ holds: $P \oplus R$ weakly terminates if and only if the transition system $TS'$ computed from reduced fragments of $P$ weakly terminates.

In the setting of service discovery it is, therefore, sufficient for the broker to store reduced fragments of a published service $P$. Upon a “find” request of a service $R$, the broker can compute the transition system of $R$ and the reduced fragments of $P$ and verify it for weak termination.
In the next section, we show the positive impact of the proposed reduction techniques by some experimental results.

5. Experimental results

The results concerning the fragment calculation and reduction have been prototypically implemented in our service analysis tool Fiona\(^1\) [23]. Fiona provides a vast spectrum of service analysis techniques. Among others, it can be used to read two service models \(P\) and \(R\), calculate a most permissive partner \(MP(P)\) of \(P\) together with the (reduced) fragments and connections of \(P\), and construct the state space \(P \oplus R\) from these fragments. The resulting transition system can then be passed on to the model checker LoLA [36]\(^2\) to check whether it is weakly terminating.

In the following, we first show experimental results\(^3\) concerning the calculation of a most permissive partner and of the full as well as the reduced fragments (see Table 1). Second, we analyzed the performance of our approach in the setting of service discovery (see Table 2). Here, with respect to a partner \(R\), we compare our matching approach to the naive approach, which model checks the complete transition system of the composition of \(P\) and \(R\).

5.1. Evaluating the publish phase

Our service models are open nets [24]—Place/Transition Petri nets extended by an interface. For the analysis, Fiona computes the reachability graph of the open net (i.e., a labeled transition system).

Table 1 provides the results of our experiment including 12 services (specified as open nets). The first five examples are industrial services taken from the WS-BPEL specification [5] (“Loan Approval”, “Purchase Order” and “Travel Service 1”), and from [6] (“Olive Oil Ordering”). “Travel Service 2” is a modification of “Travel Service 1”. Service “Registration” is an industrial service. As these examples were specified in the service description language WS-BPEL [5], we had to translate them into open nets using the compiler BPEL2oWFN [20].

The “Beverage Machine” is taken from [24]; the online shops are taken from [22]. “Philosophers” are an open net model of three and five dining philosophers. “SMTP Protocol” models the SMTP protocol.

For each service \(P\), Table 1 provides information about the state space of \(P\) (columns 2–4), the state space of a most permissive partner \(MP(P)\) of \(P\) (columns 5 and 6), the size of the fragments (columns 7–10), and the time for calculating \(MP(P)\) and computing the reduced fragments (columns 11 and 12). More precisely, \(|Q_P|\), \(|L_P^I \cup L_P^O|\), and \(|\delta_P|\) refer to the number of states, interface channels, and transitions in \(P\), respectively; \(|Q|\) and \(|\delta|\) refer to the number of states and edges in \(MP(P)\), respectively; \(|V|\) and \(|V_{red}|\) refer to the number of states of all fragments of \(P\) before and after applying the reduction rules. \(\frac{|V|}{|Q|}\) and \(\frac{|V_{red}|}{|Q|}\) show the average number of states per fragment of \(P\) before and after applying the reduction rules. Finally, \(t_{MP(P)}\) denotes the time for computing a most permissive partner of \(P\), and \(t_{red}\) shows the time needed for reducing the fragments of \(P\).

As an example, the “Online Shop 1” service has 205 states, 7 message channels, and 463 transitions. A most permissive partner of “Online Shop 1” has 12 states and 19 transitions. Thus, Fiona computed

---

\(^1\)Available at http://www.service-technology.org/fiona
\(^2\)Available at http://www.service-technology.org/lola
\(^3\)All experiments were taken on an UltraSPARC III processor with 900MHz and 4 GB RAM running Solaris 10.
Table 1. Service $P$, its most permissive partner, and application of the reduction rules to the fragments of $P$.

<table>
<thead>
<tr>
<th>Service</th>
<th>$P$</th>
<th>$MP(P)$</th>
<th>$F(P)$</th>
<th>time (h:min:sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Q_{P}]</td>
<td>$L_P \cup L_P^\phi$</td>
<td>$\delta_P$</td>
<td>[V]</td>
</tr>
<tr>
<td>Loan Approval</td>
<td>26</td>
<td>6</td>
<td>33</td>
<td>7</td>
</tr>
<tr>
<td>Purchase Order</td>
<td>12</td>
<td>10</td>
<td>15</td>
<td>168</td>
</tr>
<tr>
<td>Olive Oil Ordering</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td>Travel Service 1</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>56</td>
</tr>
<tr>
<td>Travel Service 2</td>
<td>10</td>
<td>12</td>
<td>11</td>
<td>288</td>
</tr>
<tr>
<td>Online Shop 1</td>
<td>205</td>
<td>7</td>
<td>463</td>
<td>12</td>
</tr>
<tr>
<td>Online Shop 2</td>
<td>308</td>
<td>8</td>
<td>744</td>
<td>7</td>
</tr>
<tr>
<td>Beverage Machine</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Philosophers #3</td>
<td>46</td>
<td>6</td>
<td>70</td>
<td>67</td>
</tr>
<tr>
<td>Philosophers #5</td>
<td>574</td>
<td>10</td>
<td>1,476</td>
<td>1,432</td>
</tr>
<tr>
<td>SMTP Protocol</td>
<td>60</td>
<td>14</td>
<td>92</td>
<td>470</td>
</tr>
<tr>
<td>Registration</td>
<td>2,128</td>
<td>6</td>
<td>6,889</td>
<td>24</td>
</tr>
</tbody>
</table>

12 fragments (i.e., one for each state of the most permissive partner). The sum of all states of these 12 fragments is $|V| = 137$. Applying the proposed abstraction rules results in $|V_{\text{red}}| = 15$ states. Thus, we have an average of 11.4 states for each fragment before and 1.3 states for each fragment after the reduction. Hence, at a “find” request only $\frac{15}{137} = 11\%$ of the actual state space has to be model checked. The time for computing a most permissive partner of “Online Shop 1” is four seconds; reducing the fragments takes no time.

With respect to the calculation of the fragments and the reduction thereof we can draw the following three observations:

The first observation is that the average number of states for each fragment is not high. Consequently, there is little scope for reduction. One reason might be that all open nets have been structurally reduced using Murata rules [28] before transforming their state spaces. That way, we reduce sequences of $\tau$-transitions, for instance. We apply these rules, because they significantly speed up the computation of a most permissive partner while preserving all relevant properties.

As a second observation, the numbers in Table 1 show that the proposed abstraction rules are powerful indeed. The states of the fragments are reduced significantly. None of the reduced example processes has more than two states for each fragment on average, and seven processes contain only a single state for each fragment; see column 10.

Third, we observe that for two examples, the overall computation time takes more than one hour. The computation time of a most permissive partner of the “SMTP protocol” is more than three hours, for instance. This reflects the high worst-case complexity of the algorithm for constructing a most permissive partner [41]. The algorithm computes an overapproximation of the possible states of a most permissive partner and then iteratively removes all states which cause violations of weak termination. Also the fragment reduction takes in one example, the “Registration” service, more than eight hours. A reason for the high computation time might be that the transitive reduction rule has a worst-case complexity of $O(|Q|^3)$ [4], and we were interested in the best possible reduction. There is obviously a trade-off...
between fragment size and run time that could be tackled—for example, by limiting the number of iterations through the state space to detect nodes and edges that can be removed. However, we think that the current implementation of the algorithms for calculating a most permissive partner and the fragment reduction gives room for improvement, on the one hand. On the other hand, our approach is based on the assumption that a service is published only once (i.e., the fragments have to be computed only once) and the number of “find” requests is significantly greater. In particular, the “publish” phase is not time critical and a higher computation time of reduced fragments is hence negligible.

5.2. Evaluating the find phase

In the following, we evaluate matching; that is, how the verification of the composition \( P \oplus R \) for weak termination performs, if \( P \oplus R \) is constructed from reduced fragments of \( P \). In Table 2, we exemplify the application of the fragment approach to matching.

| Service                | \( |Q| \) | \( |\delta| \) | \( |Q_{TS'}| \) | \( |\delta_{TS'}| \) | \( |Q_{P \oplus R}| \) | \( |\delta_{P \oplus R}| \) | \( |Q_{TS'}| \) | \( |\delta_{TS'}| \) | \( |Q_{P \oplus R}| \) | \( |\delta_{P \oplus R}| \) |
|------------------------|--------|--------|-------------|-------------|----------------|----------------|-------------|-------------|----------------|----------------|
| Loan Approval          | 7      | 8      | 10          | 11          | 71             | 97             | 10          | 11          | 47             | 48             |
| Purchase Order         | 11     | 10     | 11          | 10          | 68             | 109            | 11          | 10          | 38             | 37             |
| Olive Oil Ordering     | 10     | 13     | 17          | 25          | 84             | 140            | 17          | 25          | 43             | 45             |
| Travel Service 1       | 9      | 8      | 9           | 8           | 48             | 65             | 9           | 8           | 30             | 29             |
| Travel Service 2       | 26     | 42     | 27          | 43          | 552            | 1,587          | 27          | 43          | 56             | 57             |
| Online Shop 1          | 11     | 13     | 14          | 16          | 390            | 898            | 14          | 16          | 103            | 106            |
| Online Shop 2          | 6      | 5      | 7           | 6           | 138            | 236            | 7           | 6           | 83             | 86             |
| Beverage Machine       | 4      | 4      | 5           | 5           | 35             | 53             | 5           | 5           | 17             | 17             |
| Philosophers #3        | 60     | 89     | 67          | 96          | 1,881          | 4,280          | 67          | 96          | 620            | 742            |
| Philosophers #5        | 11     | 10     | 11          | 10          | 81             | 114            | 11          | 10          | 47             | 46             |
| SMTP Protocol          | 203    | 568    | 239         | 646         | 11,361         | 37,513         | 239         | 646         | 565            | 656            |
| Registration           | 19     | 36     | 20          | 37          | 73,105         | 376,555        | 20          | 37          | 82             | 81             |

For each of the 12 services in Table 1, we computed a partner \( R \) using the method in [40]. Columns 2 and 3 in Table 2 show the number of states and transitions of this service \( R \). Then, we let Fiona calculate the transition system \( TS' \) from \( R \) and reduced fragments of \( P \) as described in the previous section. Columns 4 and 5 show the state space of \( TS' \) (number of states and transitions). The sixth and seventh column present the size of the state space for \( P \oplus R \).

To check matching, we passed, for each service \( P, TS' \) and \( P \oplus R \) to the model checker LoLA and let it check for weak termination. Thereby LoLA has been configured to use all reduction techniques that are applicable when model checking a liveness property [35]. In both cases, we show the number of states and edges that LoLA used for analyzing weak termination (see columns 8 and 9 for \( TS' \) and columns 10 and 11 for \( P \oplus R \) ). We omit presenting the overall time of our approach, because it took zero seconds for all services.

As an example, let us take a closer look at the “Registration” service. The state space of this service is with 2,128 states and 6,889 transitions (see Table 1) the greatest of our experiment. The partner \( R \) of
the “Registration” service is rather small—19 states and 36 transitions (see columns 2 and 3 in Table 2). The state space of $TS'$ constructed of $R$ and reduced fragments has 20 states and 37 transitions. In contrast, the composition $P \oplus R$ of “Registration” and $R$ results in a transition system with a huge state space—73,105 states and 376,555 transitions. When model checking $TS'$ LoLA explored 20 states and 37 transitions. LoLA explored 82 states and 81 transitions when model checking the transition system $P \oplus R$.

The “Registration” service clearly shows the positive impact of the reduction techniques. The fragment statistics in Table 1 show that each fragment has many internal states—on average there are 463.7 states for each fragment before reduction. After applying the reduction rules to the fragments, this number decreases tremendously to 1.0 states for each fragment. These numbers are reflected in the state space of $TS'$ and $P \oplus R$ in Table 2. Because of the implemented reduction rules in LoLA only 82 states and 81 transitions had to be explored when model checking the transition system $P \oplus R$. Still, the reduced fragment approach only considered about one fourth of the number of states when model checking $TS'$.

Based on the experimental results we observe that all other services show a similar result. The state space of the transition system of $TS'$ is always significantly smaller than the state space of $P \oplus R$, which is also reflected in the numbers of the spaces actually explored whilst model checking the respective transition system.

We believe that this difference in size becomes even greater if industrial service models $R$ are used for matching rather than generated partners. The reason is that the services $R$ computed according to the method in [40] do not have internal transitions. Internal transitions in $R$ may, however, significantly increase the size of the state space of $P \oplus R$. As our abstraction rules are powerful and experimental results in [13] showed that industrial service models can be verified by state-of-the-art model checkers within several milliseconds, we assume that the state space of the computed transition system is still tractable.

Our approach has yet another advantage in comparison to model checking $P \oplus R$. When composing fragments of $P$ according to the minimal simulation relation of $R$ by the fragments of $P$, we can detect already during the construction of $TS'$ whether the resulting state space can deadlock. To this end, we use a method proposed in [24]. The idea is that, for each fragment, we can calculate a Boolean formula encoding which connections leaving the fragment have to be present in the simulation relation. This formula can be evaluated during the construction of $TS'$ and thus upon violation, the construction of $TS'$ can be aborted immediately. Hence, once $TS'$ is constructed, we only need to check for livelocks.

Summarizing, our experiments validated our assumption that model checking a reduced state space lessens the computational effort due to the smaller number of states and hence justifies our proposal for shifting computation effort from the resource critical “find” phase to the “publish” phase. We could, however, not verify a speed up in time. Here, our matching approach and the naive approach took zero seconds.

6. Related Work

There is a lot of research being done to provide a methodology to support matching in Service Oriented Architectures. As already mentioned in the introduction, some approaches propose to compute and publish a public view $P'$ of a provided service $P$ [3, 19]. Then, upon a “find” request of a service $R$ matching means model checking the composition $R \oplus P'$ to decide proper interaction. However, in
these approaches the computational effort takes place during every “find” request whereas our approach
shifts this effort to the “publish” phase. Other approaches such as [24, 43] reduce the implementation
of a “find” request to a matching problem, but in these approaches proper termination is restricted to
deadlock freedom whereas we also consider livelock freedom.

The reduction rules presented in this article are related to work on reduction of transition systems.
For example, bisimulation [31] preserves properties that can be specified in the logic CTL and hence
weak termination [10]. The weaker notation, weak bisimulation [26], preserves weak termination [17].
A special case of reduction is minimization of transition systems (see [14, 33, 38], for instance). These
papers present congruences with respect to composition which never equate a transition system with a
property such as deadlock or livelock freedom with one which does not satisfy that property. In contrast
to our reduction, minimization produces the smallest possible result.

To reduce service $P$, one can also compute an interface specification $I$ [16] rather than a most permis-
sive partner. The composition $P \oplus I$ is a transition system (i.e., a closed system) that can be reduced. To
relate two services $P$ and $P'$, the accordance preorder has been introduced in [2]. As noticed by [7, 25],
the notion of fair testing [34] implies accordance (called conflict preorder in [25], and subcontract in [7]),
buts accordance does not imply fair testing. In [27], the differences between accordance and fair testing
have been identified. Hence, any reduction rule that preserves fair testing can be applied to reduce ser-
vice $P$ to a service $P'$ such that for any partner $R$ of $P$ the composition $P \oplus R$ weakly terminates if and
only if $P' \oplus R$ weakly terminates. Juan et al. [18] define several deadlock-preserving reduction rules for
finite state machines modeling reactive systems. In Sect. 4, we have generalized these rules and adapted
them to preserve livelocks as well.

7. Conclusion

We proposed a methodology to construct the state space $P \oplus R$ of finite state machines $P$ and $R$ from
a finite set of (state space) fragments of $P$. A fragment contains that part of $P \oplus R$ that takes place
between two subsequent transitions of $R$. That way, $P \oplus R$ can be computed by composing fragments
of $P$ according to the behavior of $R$. We further presented several rules to reduce the state space of the
fragments.

Our work is motivated by a scenario in Service Oriented Architectures where services are published
in a registry and have to be selected upon a service request. That means, given a service $R$, one has to
select a service $P$ from a registry and to check whether the composition $P \oplus R$ is weakly terminating.
This property guarantees the absence of deadlocks and livelocks. As the number of “find” requests is
much greater than the number of “publish” events, we proposed to publish (reduced) fragments of $P$.
That way, computational effort in reducing the state space $P \oplus R$ is shifted from “find” to “publish”,
and thus at a “find” request only the reduced state space of $P \oplus R$ has to be model checked for weak
termination.

In a prototypical implementation, we experimented with several scientific and industrial service mod-
els. The results obtained have shown that our reduction techniques reduce the state space of fragments
significantly. We further showed the application of our approach in service discovery. We validated our
assumption that the reduction of fragments before publishing a service will have a positive effect to the
resource critical “find” phase. Our experimental results show that the computational effort of matching—
to model check the state space $TS' \oplus R$ (with reduced fragments) for weak termination—is smaller than
checking $P \oplus R$ for weak termination in the naive approach.

In ongoing work, we are working on a tool suite consisting of small tools that are tailored for solving one specific task. For example, we plan on re-implementing the fragment reduction in a separate tool, which will be trimmed for a good performance with respect to memory consumption and time. In addition, we will work on a procedure to decide whether every partner of a service $P$ is a partner of service $P'$ given their sets of fragments. That way, we can decide whether $P$ can be substituted by $P'$ without affecting any client of $P$.

Acknowledgments

The authors thank Dirk Fahland and Niels Lohmann for their help on the experimental results and the two anonymous referees for their valuable comments and suggestions.

References


