Christian Stahl

Service Substitution

A Behavioral Approach Based on Petri Nets

Dissertation

Service Substitution
A Behavioral Approach Based on Petri Nets

by Christian Stahl

Eindhoven: Technische Universiteit Eindhoven, 2009. Proefschrift.

Cover design by Frans Goris

A catalogue record is available from the Eindhoven University of Technology Library

ISBN: 978-90-386-2065-7
NUR 993

This work has been partially supported by the DFG within grant "Service Substitution" (RE 834/16-1).

SIKS Dissertation Series No. 2009-39

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Printed by University Press Facilities, Eindhoven

Service Substitution

A Behavioral Approach Based on Petri Nets

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op dinsdag 1 december 2009 om 14.00 uur

door

Christian Stahl

geboren te Berlijn, Duitsland

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. K.M. van Hee
en
Prof.Dr. W. Reisig

Copromotor:
Prof.Dr. K. Wolf

Service Substitution

A Behavioral Approach Based on Petri Nets

Dissertation

zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
(*doctor rerum naturalium*, Dr. rer. nat.)
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät II der
Humboldt-Universität zu Berlin
im Rahmen einer Doppelpromotion mit der
Technische Universiteit Eindhoven, Niederlande

von
Herrn Diplom-Informatiker

Christian Stahl

geboren am 15. Juni 1978

1. Gutachter	Prof. Dr. Kees M. van Hee
2. Gutachter	Prof. Dr. Wolfgang Reisig
3. Gutachter	Prof. Dr. Karsten Wolf

eingereicht am 9. Oktober 2009
Tag der mündlichen Prüfung 1. Dezember 2009

Abstract

Service Substitution
A Behavioral Approach Based on Petri Nets

Service-Oriented Computing is an emerging computing paradigm that supports the modular design of (software) systems. Complex systems are designed by composing less complex systems, called *services*. Such a (complex) system is a distributed application often involving several cooperating enterprises. As a system usually changes over time, individual services will be *substituted* by other services. Substituting one service by another one should not affect the correctness of the overall system. Assuring correctness becomes particularly challenging, as the services rely on each other, and each of the involved enterprises only oversees a part of the overall system. In addition, services communicate asynchronously which makes the analysis even more difficult. For this reason, formal methods to support service substitution are indispensable.

In this thesis, we study service substitution at the level of service models. Thereby we restrict ourselves to *service behavior*. As a formalism to model service behavior, we use Petri nets.

The first contribution of this thesis is the definition of several *substitutability criteria* that are suitable in the context of Service-Oriented Computing. Substituting a service S by a service S' should *preserve some behavioral properties* of the overall system. For each set of behavioral properties and a given service S, there exists a set of behaviorally compatible services for S. A substitutability criterion defines which of these behaviorally compatible services of S have to be preserved by S'. We relate our substitutability criteria to preorders and equivalences known from process theory.

The second contribution of this thesis is to present, for each substitutability criterion, a procedure to *decide* whether a service S' can substitute a service S. The decision requires the comparison of the in general infinite sets of behaviorally compatible services for the services S and S'. Hence, we extend existing work on an abstract representation of all behaviorally compatible services for a given service. For each notion of behavioral compatibility, we present an algorithmic solution to represent all behaviorally compatible services. Based on these representations, we can decide substitutability of a service S by a service S'.

The third contribution of this thesis is a method to support the *design* of a service S' that can substitute a service S according to a substitutability criterion. Our approach is to derive a service S' from the service S by stepwise transformation. To this end, we present several transformation rules.

Finally, we formalize and we extend the equivalence notion for services specified in the language WS-BPEL. That way, we demonstrate the applicability of our work.

Kurzfassung

Service-Oriented Computing is ein vielversprechendes Paradigma der Softwarekonstruktion, das den modularen Entwurf von (Software-) Systemen unterstützt. Komplexe Systeme werden durch Komposition weniger komplexer Systeme, *Services* genannt, entworfen. Solch ein (komplexes) System ist eine verteilte Anwendung, die oft mehrere kooperierende Unternehmen einschliesst. Da ein System gewöhnlich zeitlichen Änderungen unterworfen ist, werden einzelne Services durch andere Services *ausgetauscht*. Der Austauch eines Service gegen einen anderen sollte dabei nicht die Korrektheit des Gesamtsystems verletzen. Korrektheit zuzusichern ist nichttrivial, weil Services voneinander abhängen und jedes involvierte Unternehmen nur einen Teil des Gesamtsystems überblickt. Zudem kommunizieren Services asynchron. Das erschwert die Analyse noch weiter. Aus diesem Grund ist der Einsatz formaler Methoden zur Austauschbarkeit von Services unabdingbar.

In der vorliegenden Dissertation studieren wir Austauschbarkeit von Services auf der Modellebene. Dabei beschränken wir uns auf das *Verhalten von Services*. Wir verwenden Petrinetze, um das Verhalten von Services formal zu fassen.

Der erste Beitrag dieser Dissertation ist die Definition mehrerer *Austauschbarkeitskriterien*, die im Rahmen des Service-Oriented Computing anwendbar sind. Der Austausch eines Service S gegen einen Service S' sollte bestimmte Verhaltenseigenschaften des Gesamtsystems *bewahren*. Zu jeder Menge von Verhaltenseigenschaften und einem Service S existiert eine Menge von Services, die verhaltenskompatibel zu S sind. Ein Austauschbarkeitskriterium definiert, welche dieser zu S verhaltenskompatiblen Services der Service S' bewahren soll. Wir erarbeiten den Bezug unserer Austauschbarkeitskriterien zu Quasiordnungen und Äquivalenzen aus der Prozesstheorie.

Als zweiten Beitrag dieser Dissertation präsentieren wir zu jedem Austauschbarkeitskriterium eine algorithmische Lösung um zu *entscheiden*, ob ein Service S gegen einen Service S' ausgetauscht werden kann. Um Austauschbarkeit zu entscheiden, müssen die beiden im Allgemeinen unendlichen Mengen verhaltenskompatibler Services der Services S und S' miteinander verglichen werden. Wir erweitern Vorarbeiten zur abstrakten Repräsentation aller verhaltenskompatibler Services und erarbeiten für jeden Begriff von verhaltenskompatibel eine algorithmische Lösung, um alle verhaltenskompatiblen Services zu repräsentieren. Mit Hilfe dieser Repräsentationen können wir entscheiden, ob ein Service S' einen Service S austauschen darf.

Der dritte Beitrag dieser Dissertation ist eine Methode, um den *Entwurf* eines Service S' zu unterstützen, so dass S' einen Service S bezüglich eines Aus-

tauchbarkeitskriteriums austauschen kann. Dazu verfeinern wir den Service S schrittweise zu einem Service S'. Zu diesem Zweck präsentieren wir mehrere Verfeinerungsregeln.

Unsere Ergebnisse ermöglichen es uns, den Äquivalenzbegriff für Services, die in der Sprache WS-BPEL spezifiziert sind, zu formalisieren und zu erweitern. Damit zeigen wir die Anwendbarkeit unserer Ergebnisse.

Contents

Part I.

Introduction

1. About This Thesis

In this chapter, we introduce services and the service-oriented world. We identify service substitution as an important and interesting research problem that we will address in this thesis. Finally, we describe the main results of our work and outline the organization of this thesis.

1.1. Background

In this section, we give a brief overview of the emerging computing paradigm Service-Oriented Computing.

1.1.1. Service-Oriented Computing

Since the early days of computer science, it is well-known that mastering the complexity of large (software) systems is a major challenge. One very successful approach for handling complexity is *modularization.* The principle of *compositionality* is one of the most desirable requirements for modular systems: A collection of modules that are properly connected to each other should behave as one module itself. During the last decade, modularization is considered as the most important feature of a system design.

The trend towards modularization is driven by enterprises being faced with the challenge of rapidly changing their systems. On the one hand, today's systems are highly *complex*, run in heterogenous environments, and are often distributed over several enterprises. On the other hand, the ever-changing market conditions, regulations etc. require enterprises to act very *flexibly.* Systems are subject to ongoing changes, but the integration of these changes should not take much time. Hence, enterprises need an IT infrastructure that can cope with these requirements.

Service-Oriented Computing (SOC) [Pap07] is a novel computing paradigm that aims to "support the development of rapid, low-cost and easy composition of distributed applications even in heterogeneous environments" [PTDL08]. It follows the idea to create a complex system by connecting modules, called *services*. Therewith, SOC reuses old ideas from component-based design [McI68, Szy98] or from programming-in-the-large [DK75], for instance.

A service encapsulates some functionality, which can be accessed via its *interface.* The interface of a service consists of a set of message channels and is used to communicate with other services via *asynchronous message passing.* To this end, services are composed by connecting their message channels. Hence, *interaction* is a first-class citizen in SOC.

SOC follows the paradigm to separate the functionality from the interface in a service. This separation has two advantages. On the one hand, a service is independent of applications and the computing platforms on which it runs. On the other hand, a service can be connected to other services without having knowledge of their technical details; services are loosely coupled. That way, SOC helps to reduce the complexity of integrating services within and across organizational boundaries.

One of the most prominent technologies based on SOC are *Web services*. A Web service is a service that can be accessed via the Internet. The interface of a Web service is specified using the Web Services Description Language (WSDL) [CMRW07]. For message exchange, standard-based protocols, such as SOAP [ML07], are used.

The key technology to design and to execute systems according to the paradigm of SOC is a *Service-Oriented Architecture* (SOA). An SOA provides an IT infrastructure for publishing services of an enterprise via the Internet [ABH+07]. These published services can then be automatically found and used by other enterprises. That way, an SOA enables interoperability between systems and hence reduces complexity of systems.

1.1.2. Service composition

According to the SOC paradigm, services are composed to form more complex services. Hence, a service is usually stateful. A service has a definition. This definition describes the behavior and the interface of the service. The *behavior* of a service is described by a partially-ordered set of *activities*. An activity is the atomic unit of work in a service. The execution of an activity is either internal to the service or yields the sending or the receiving of a message. In the literature, the term *business protocol* [Pap07] is used as a synonym for service behavior. A service can be executed; that is, an *instance* of this service is created. An instance can execute activities. Figure 1.1 illustrates these terms.

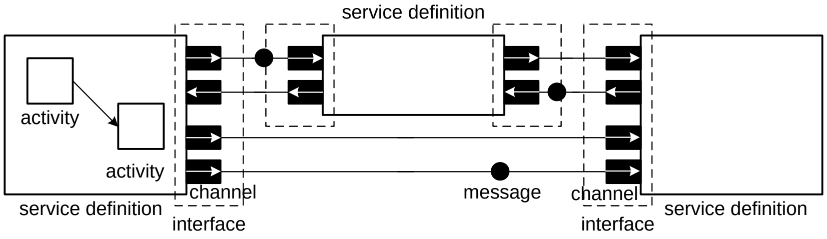


Figure 1.1.: Illustration of a service composition showing the main terms used for describing a service.

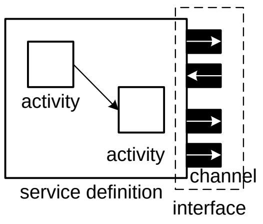


Figure 1.2.: Illustration of a service orchestration.

There exist two different service descriptions in the literature [Pel03]. A service *orchestration* describes the behavior of a service composition from the point of view of a single service of this composition. In contrast, a service *choreography* describes the behavior of a service composition from the perspective of all services. As a further difference, a choreography shows usually only the message exchange among the services in the composition and abstracts from implementation details, whereas an orchestration usually provides implementation details. Prominent service description languages are WS-BPEL [Alv07] to specify an orchestration and WS-CDL [KBR$^+$05], Let's dance [ZBDH06], and BPEL4Chor [DKLW07] to specify a choreography.

As an illustrating example, the service composition in Figure 1.1 is a service choreography, as it describes the whole service composition. In contrast, Figure 1.2 depicts an orchestration of the service definition on the left hand side in Figure 1.1.

A service definition covers various *aspects* of a service. We distinguish the control-flow perspective, the data perspective, and the resource perspective. The *control-flow perspective* focuses on the ordering of the activities of the service. The way in which data is presented and utilized in a service is described by the *data perspective*. The *resource perspective* specifies who actually executes an activity. A resource is either a human or a non-human.

We illustrate the three aspects of a service by the help of a credit approval of a bank. The control-flow perspective describes that the bank first receives documents from the customer, and based on these documents the bank decides whether the credit will be approved or not. The data perspective specifies how the customer's documents are stored. Finally, the resource perspective describes whether the decision is made by a customer consultant, by the bank manager, or by the computer system.

A business protocol focuses mainly on the control-flow perspective of a service, but it may also contain data and resource information. A service orchestration usually provides information about all three perspectives, whereas a service choreography does not specify data and resources in general.

An important property of a service composition is compositionality; that is, the composition is again a service. To achieve compositionality, a service composition must be *compatible*. Compatibility is focussing on four aspects of a service: its interface, its semantics, its behavior, and its quality of service (QoS). Interface compatibility ensures that pairwise connected message channels have the same message type. Semantical compatibility guarantees that messages and their content are correctly interpreted. Behavioral compatibility is devoted to exclude behavioral errors, such as deadlocks and livelocks. Finally, QoS compatibility ensures some quality parameters—for example, throughput time or security standards.

1.1.3. Research challenges in Service-Oriented Computing

The realization of SOC is still in its infancy. There are a lot of research challenges to be solved to make the idea of SOC come true. In [PTDL08], leading researchers in the area of SOC define the "grand challenges" in SOC research. These challenges are classified in four research themes: service foundations, service-oriented engineering, service management, and service composition.

The first research theme, service foundations, provides technologies to realize an SOA. To this end, a middleware is needed that allows to connect heterogeneous services, to dynamically bind services, to publish services via the Internet, and to find published services.

Service-oriented engineering covers the design and the deployment of services. Although the SOC paradigm reuses known ideas from component-based design, it requires novel methods for specifying, designing, and monitoring services.

Service management contains all tasks regarding controlling and monitoring of services. As services are executed in highly flexible environments, they should contain functionality for self-healing, self-adapting, and self-optimizing.

The forth theme is service composition. It covers the design of complex systems from services. Research challenges in this theme include:

- *Expanding services:* To automatically find compatible pairs of services, services need to know each other (to some degree). Hence, enterprises need to provide sufficient information about their published services. On the one hand, this information must allow to analyse for compatibility; that is, it must contain facts about the service interface, about the service behavior, about the service semantics, and about QoS properties of the service. On the other hand, enterprises do not want to reveal their trade secrets. In other words, providing the complete service definition is not an issue. Consequently, one open problem is to identify what information an enterprise has at least to publish about its service.

- *Finding a compatible service:* Information about published services will be stored in a service repository. Other services will search service repositories to find compatible pairs of services. Hence, efficient techniques to search in a repository are crucial. Otherwise, the idea of automatically finding services cannot be realized.

- *Composing services using adapters:* Services are usually designed by different enterprises. So a pair of services may not necessarily be compatible. One method to approach this problem is to calculate an adapter; that is, a service that can resolve the incompatibilities between the services. Clearly, an SOA should provide techniques to automatically calculate adapters.

- *Service substitution:* The modular design of services makes changes of the overall service more easy; that is, one service may be substituted by another service. However, this substitution must not affect the compatibility of the overall service.

In this thesis, we address the last research challenge, *service substitution.*

1.2. Problem description and problem statement

Systems inevitably evolve over time; for example, some new functionality is added, or some quality parameter of some functionality is improved. In monolithic systems, even small changes often cause much integration work in the overall system. In contrast, the modular design of (composed) services enables enterprises to substitute periodically individual services by better ones. Technically, such a substitution is supported by the loose coupling of services.

Substituting one service by another one should preserve compatibility of the overall service. *Service substitutability*—that is, deciding whether a service can substitute another service—is considered to be one of the most notable SOC research challenges for the near future [PTDL08].

In this thesis, we restrict ourselves to changes of the service behavior, which are also known as *business protocol changes* [Pap08]. This restriction implies that we assume that QoS properties and semantical properties are *not* violated when changing a service S to a service S'. That means, we mainly focus on the control-flow perspective of services, and we abstract from resources and consider only data/message types and not their content. For that reason, service substitutability will guarantee only behavioral compatibility of the overall service in this thesis.

Service substitutability is particularly challenging, as the services in a composition rely on each other. Furthermore, we cannot assume that an enterprise that substitutes an individual service has knowledge about the overall service composition—for example, if the individual services belong to different enterprises. Hence, a procedure to decide substitutability must be *independent* of the actual service composition. Moreover, services communicate asynchronously making the decision procedure even more complex [Alo08]. Asynchronous communication is non-blocking. After a service has sent a message, it can continue its execution and does not have to wait until this message is received. Furthermore, the order in which the messages are sent is not necessarily the order in which they are received.

A service S' can definitively substitute a service S if every service that interacts with S cannot distinguish between S and S'. In practice, however, more general substitutability criteria are relevant; for example, S' may guarantee a stronger termination criterion than S. In general, when substituting S by S' the composition of S' and a service S^* *preserves* some behavioral properties of the composition of S and S^*.

Service substitutability focuses on two different aspects: *static business protocol evolution* and *dynamic business protocol evolution*. The latter is also known as *instance migration*. As the main difference, static business protocol evolution assumes that the service S has *no running instances*. Dynamic business protocol evolution, in contrast, assumes that there exist running instances of S, and hence one is interested in migrating a (running) instance of the service S to an instance of the service S'. Deciding dynamic business protocol evolution is particularly important if the service S has long running instances—for example, in case of a life insurance. In this thesis, we restrict ourselves to static business protocol evolution. As dynamic business protocol evolution builds on static business protocol evolution, this thesis can be seen as a basis for studying dynamic business protocol evolution; see the work on projection inheritance [AB02], for instance.

In the rest of this section, we identify with *multiparty contracts* and *service improvement* two application scenarios of service substitutability in the context of SOC. For each scenario, we describe the problem and identify research questions.

1.2.1. Application 1: Multiparty contracts

An SOA enables an enterprise to publish services via the Internet. These services can then be automatically found and used by other enterprises. According to the SOC paradigm, interorganizational cooperation among enterprises should be realized in such a way. However, this approach has not become accepted in practice, because there is no accepted standard that can cope with all four aspects of service compatibility (see Section 1.1.2). An additional and the main limiting factor is that enterprises usually cooperate only with enterprises they already know.

Therefore, in practice a more pragmatic approach is used instead. The parties that will participate in an interorganizational cooperation specify together an abstract description of the overall service. This description is a choreography. The choreography consists of a set of activities. Each activity is assigned to one party. A connection between two activities is either internal—that is, both activities belong to the same party—or external—that is, both activities belong to different parties. A party's share of the choreography (i. e., its *public view*) is then the projection of the choreography to the party's activities. The choreography serves as a common *contract* among the parties involved in the cooperation.

The challenge of the contract approach is to balance the following two conflicting requirements: On the one hand, there is a strong need for *coordination* to optimize the flow of work in and among the different parties. On the other hand,

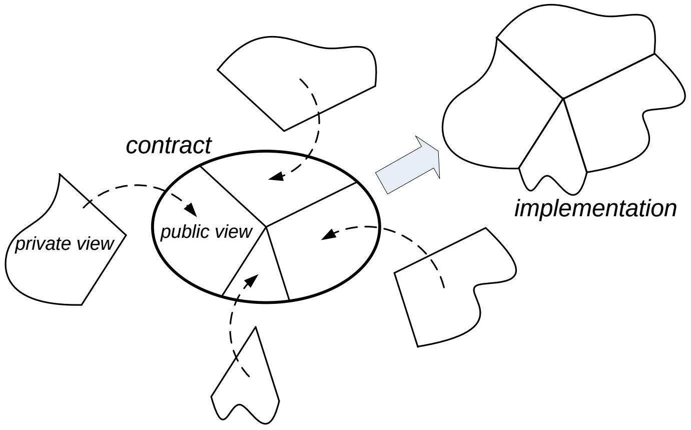


Figure 1.3.: Illustration of the contract approach. Each of the four public views is substituted by its corresponding private view yielding the overall implementation.

the parties involved in the cooperation are essentially *autonomous* and have the freedom to create or modify their services at any point in time. Furthermore, the parties do not want to reveal their trade secrets. Therefore, it has been proposed in [AW01, Aal03] to use a contract that defines "rules of engagement" without describing the internal services executed within each party.

After the parties have specified the contract, each party will implement its public view on its own. The implementation, the *private view*, will usually deviate significantly from its public view. Obviously, these local modifications have to *conform* to the agreed contract. This is, in fact, a nontrivial task, because it may cause global errors, such as deadlocks, as shown in [AW01]. As all parties are autonomous, none of them owns the overall service (i. e., the implemented contract). Therefore, none of the parties can verify the overall service. As a result, an approach is needed such that each party can check locally whether its private view guarantees global correctness of the overall service.

The basic idea of the contract approach is illustrated in Figure 1.3. The starting point is a contract partitioned over the four parties involved. The public view of each of the four parties is illustrated in the figure as a fragment of the contract. Based on the public view, each party implements its private view. Hence, the actual implementation of the contract consists of the four private views glued together as shown in the top-right corner in Figure 1.3.

Based on these considerations, we identify the following problems related to service behavior.

How can we decide locally correctness of a private view? We need an algorithm to decide locally whether the public view of a party can be substituted by its private view such that correctness of the contract is guaranteed.

How can we design a private view that is correct-by-construction? Each party involved in a contract has to design its private view. This is, however, a nontrivial and error-prone task even for experienced service designers. Hence, it is desirable to support the design of a private view that is correct-by-construction.

Checking correctness of a public view and supporting the design process of correct private views is not a particular strengths of Business Process Management (BPM) tools currently available on the marketplace. For example, the service description language WS-BPEL offers a standard to model public and private views. It also defines an equivalence between a private view and its public view. As a limiting factor, these equivalence is defined on the syntax of services and does not consider the behavior of services. Consequently, the design of a private view is unnecessarily restricted.

1.2.2. Application 2: Service improvement

Today's enterprises consider themselves to be exposed to intense competition. Reasons are among others the ever-changing markets, the ongoing development of new technologies, and coping with the increasing requirements of the customers. To operate successfully, enterprises have to increase their profit wherever possible. *Service improvement* aims at *revising* services such that they become more profitable. To this end, weaknesses of this service, such as bottlenecks or unprofitable lines of business, have to be figured out. In addition, the quality and the reliability of the service is improved, or new features are provided to attract the customers. The approach is restricted to improvement rather than optimization, because the complexity of services makes it in general impossible to find an optimum.

On the level of service behavior, service improvement usually leads to restructuring of services (i. e., reordering of activities), to adding new functionality (i. e., adding activities), or to deleting functionality (i. e., deleting activities). As in the context of service contracts, improving a service S according to some criterion yields a service S' that shall substitute S. Beside financial or performance aspects, the correctness of the service behavior of S' is indispensable, because even small local changes in a service model may cause global errors, such as deadlocks. In addition, we need to check whether the newly designed service S' implements the expected functionality. From these considerations, we identify the following two problems related to service behavior.

How can we specify service behavior? When a service S is improved, some behavior of S is identified, which has to be preserved in an improved version S' of S. We need to formally specify this behavior.

How can we decide correctness of an improved service? We need an algorithm to decide whether an improved version S' of a service S preserves the desired behavior of S.

Similar to multiparty contracts, checking correctness of an improved service is also not a particular strengths of BPM tools currently available on the marketplace. For that reason, we identified with multiparty contracts and service improvement two interesting and practical relevant research questions that we will address in this thesis.

1.3. Thesis overview

In this section, we present an overview of our achieved results and outline this thesis.

1.3.1. Results overview

In this thesis, we study the question whether a service S' can substitute a service S. We present substitutability criteria for services, and we develop algorithms to decide substitutability according to a substitutability criterion. We also develop a method to construct a substitutable service S' for S.

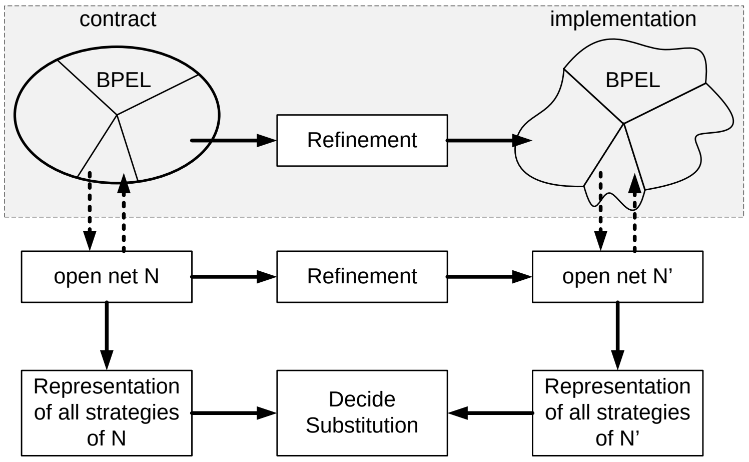


Figure 1.4.: Illustration of the thesis’ results.

Figure 1.4 illustrates the results of this thesis. As shown, we will study service substitution on the level of service models. As a formal service model, we use open nets [MRS05], a subclass of Petri nets tailored towards the modeling of services. Suitability of this model has been demonstrated by open-net semantics for various languages, such as BPMN, WS-BPEL, and BPEL4Chor [Loh08, OVA$^+$07, LKLR08, DDO08, LVD09].

In the following, we present an overview of our achieved results.

Substitutability criteria

Substituting a service S by another service S' should preserve behavioral compatibility of the overall service. In this thesis, we consider several notions of behavioral compatibility, which are combinations of

- deadlock freedom (i. e., the service cannot get stuck),
- weak termination (i. e., the possibility to always reach a final state),
- ensuring that all activities of the service can be potentially executed, and
- a criterion to restrict final states.

A service S communicates with other services; hence, we define the semantics of S by the set of all services R such that the composition of S and R is behaviorally compatible. We call R a *strategy* of S according to the notion of behavioral compatibility. With the help of strategies, we define two substitutability criteria: *conformance* and *preservation* [SMB09]. Conformance is used in the setting of multiparty contracts. It guarantees that no strategy of S can distinguish between S and S'; that is, every strategy of S is a strategy of S'. Preservation is used for service improvement. It is a less restrictive notion than conformance and requires that S' preserves a subset of the strategies of S.

As the notion of conformance is a classical preorder, we relate it to preorders known in process theory. In case of deadlock freedom, we show that conformance coincides with the stable failures preorder. In case of the stronger termination criterion weak termination, we identify fair testing as the closest known preorder for conformance and prove under which condition they coincide [MSV09].

Finite representations of sets of services

To decide service substitutability, we have to compare the two in general infinite sets of strategies of S and of S'. For that purpose, this thesis contributes in the development of a finite representation of the in general infinite set of strategies of a service; see Figure 1.4.

In case of deadlock freedom, it has been shown that the set of all strategies of a service S can be characterized by an automaton-based representation, the *operating guideline of* S. There exist an algorithm to calculate the operating guideline of S and an algorithmic solution to check *containment* of a service in the operating guideline of S.

In this thesis, we extend the notion of an operating guideline and define another *five representations* that characterize the set of all strategies for behavioral compatibility different from deadlock freedom. In particular, we represent all strategies of a given service in case of weak termination. For each finite representation, we present a construction algorithm and a procedure to check containment [SW08, WSOD09].

Deciding service substitutability

For each substitutability criterion, we present an algorithm to *decide* substitutability. The decision procedure is nontrivial, because we have to compare two in general infinite sets of strategies. As these sets of strategies can be represented in a compact manner, we compare their respective finite representations instead. This is shown by the box Decide Substitution in Figure 1.4.

A service S' conforms to a service S if every strategy of S is also a strategy of S'. Hence, given the finite representations of all strategies of S and of S', the decision procedure reduces to an inclusion check on these finite representations. For two of the six finite representations, we provide an algorithm to decide inclusion and hence to decide conformance [ALM$^+$09, SW09a].

In case of preservation, only a subset $\mathcal{S}$ of the strategies of S has to be preserved by S'. We define the *intersection* of two sets of strategies based on their finite representations. Intersection is used to calculate a finite representation that characterizes the restriction of the strategies of S to $\mathcal{S}$. So, the procedure to decide preservation reduces to an inclusion check of $\mathcal{S}$ in the set of all strategies of S', which can be done on their finite representations. As this decision procedure reduces to decide conformance, we present a decision algorithm only for two of the six finite representations (like for conformance). However, if the set $\mathcal{S}$ is finite, we present a solution, for each of the six finite representations [SMB09].

Constructing substitutable services

Besides algorithms for deciding substitutability, this thesis also contributes in the *construction of substitutable services.*

We define several conformance-preserving transformation rules [ALM$^+$08]. These rules allow for removing, for adding, and for reordering of activities. They can be used to derive a service S' from a service S by stepwise transformation such that each transformation step preserves every strategy of S. This is illustrated by the Refinement box in the center of Figure 1.4.

The service description language WS-BPEL defines an equivalence relation between a service specification (i. e., abstract process) and a service implementation (i. e., executable process). This equivalence relation is, however, only defined on the XML syntax of services. We formalize this equivalence relation in terms of strategies. By reformulating our transformation rules, we provide a sufficient condition to decide whether two services specified in WS-BPEL are equivalent [KLM$^+$08]; see the topmost Refinement box in Figure 1.4.

Most parts of this thesis have been published at international conferences and in international journals. We will indicate this in the beginning of each chapter.

For a proof-of-concept, most of the algorithms and strategy representations we will present in this thesis have been prototypically implemented in the service analysis tool Fiona[1] [MW08]. The core developers of Fiona are Peter Massuthe and Daniela Weinberg. The implementation work of the results presented in this thesis has been mainly done by Robert Danitz, Leonard Kern, and Janine Ott.

1.3.2. Road map

This thesis has five parts.

Part I continues in Chapter 2 with an introduction to the formalisms that are used for the modeling and the verification of service behavior.

Part II formalizes substitutability criteria for multiparty contracts and for service improvement and presents four compact representations for in general infinitely many services. In Chapter 3, we define the two notions conformance and preservation, and we relate conformance to known process preorders. In Chapter 4, we classify behavioral compatibility and present, for each class of behavioral compatibility, a finite representation of all strategies for a given service.

Part III describes methods to decide service substitutability in the setting of multiparty contracts and service improvement. In Chapter 5, we present a method to decide inclusion of two infinite sets of services based on their finite representations. Inclusion is used to decide conformance. In Chapter 6, we define the intersection of two infinite sets of services based on their respective finite representations. Intersection is used to restrict the strategies of a service to those strategies that have to be preserved by the substitution under preservation.

Part IV describes a method to construct substitutable services. In Chapter 7, we present transformation rules to incrementally transform a service S into a service S' such that all relevant properties are guaranteed by construction. These transformation rules can be applied in the setting of multiparty contracts. In Chapter 8, we apply our theoretical results on service substitutability to the service description language WS-BPEL. We formalize behavioral equivalence of WS-BPEL processes and present a decision procedure based on transformation rules.

Part V concludes this thesis. In Chapter 9, we compare the results of this thesis with existing work. Finally, we summarize the results of this thesis and discuss open problems (strongly related to the topics of this thesis) and future research (loosely related to the topics of this thesis) in Chapter 10.

[1]available at `http://www.service-technology.org/fiona`

2. A Formal Model for Service Behavior

In this chapter, we introduce the formalisms used for the modeling and the verification of service behavior. In particular, we introduce *open nets*, which refine classical place/transition Petri nets by an interface to model asynchronous message passing, and *service automata*, which basically model the transition system of the internal states of open nets. Open nets and service automata can be used to model the behavior of a service in isolation as well as to model the behavior of a service composition.

In Section 2.1, we introduce transition systems and equivalence notions for transition systems. In Section 2.2, we define Petri nets. Afterwards, we introduce open nets in Section 2.3. In Section 2.4, we present a notion of interface compatibility and formalize open-net composition. A special service composition are multiparty contracts, which we formalize in Section 2.5. Subsequently, we define behavioral properties of open nets and present several notions of behavioral compatibility of open nets in Section 2.6. In Section 2.7, we introduce service automata. Finally, we discuss in Section 2.8 the restrictions of our proposed models and show how services being specified in industrial service description languages can be automatically translated into open nets.

2.1. Transition systems and equivalence notions

In this section, we introduce labeled transitions systems as a basic formalism to model the behavior of a system. We also define several equivalence notions for labeled transition systems.

Definition 2.1.1 (labeled transition system (LTS)).
A *labeled transition system* (LTS for short) $TS = (Q, \Sigma, \delta, q_0)$ consists of

- a countable set Q of *states*;
- an *alphabet* Σ of visible actions; the internal action is denoted by $\tau \notin \Sigma$;
- a *transition relation* $\delta \subseteq Q \times (\Sigma \cup \{\tau\}) \times Q$ on states; and
- an *initial state* $q_0 \in Q$.

An LTS is *deterministic* iff, for all $q, q', q'' \in Q$, $a \in \Sigma$, (q, τ, q') implies $q = q'$ and $(q, a, q'), (q, a, q'') \in \delta$ implies $q' = q''$. It is *finite* iff Q is finite. Whenever necessary, we extend an LTS by a set $\Omega \subseteq Q$ of *final states*, i. e., $TS = (Q, \Sigma, \delta, q_0, \Omega)$.⌟

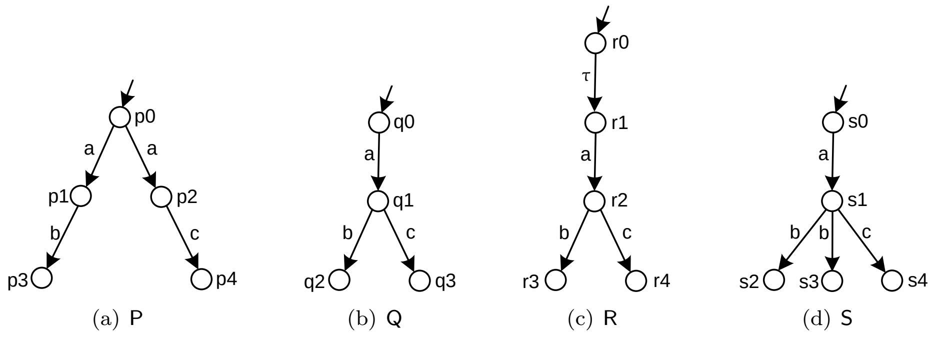


Figure 2.1.: Four LTSs: Q and S simulate P; Q and S are bisimilar; Q and R are branching bisimilar.

The transition relation δ reflects state changes of an LTS. For any two states q and q' and any action $a \in \Sigma \cup \{\tau\}$, we write $q \xrightarrow{a} q'$ if an a-labeled transition exists from q to q'. We write $q \xrightarrow{a}$ if there exists a q' such that $q \xrightarrow{a} q'$. By $q \xrightarrow{*} q'$, we denote that there exists a (possible empty) sequence $q \xrightarrow{a_1} \ldots \xrightarrow{a_n} q'$ of transitions from q to q' and say that q' is *reachable* from q.

If B is a set, then B^* denotes the set of all lists over B; ϵ denotes the empty list, and lists are concatenated by juxtaposition. For $w \in \Sigma^*$, $\xRightarrow{w}$ is the least relation satisfying:

- $q \xRightarrow{\epsilon} q$;
- $q \xRightarrow{w} q' \ \wedge \ q' \xrightarrow{a} q'' \quad \Rightarrow \quad q \xRightarrow{w\ a} q''$, for any action $a \in \Sigma$;
- $q \xRightarrow{w} q' \ \wedge \ q' \xrightarrow{\tau} q'' \quad \Rightarrow \quad q \xRightarrow{w} q''$.

We write $q \xRightarrow{w}$ if there exists a q' such that $q \xRightarrow{w} q'$.

The difference between the two relations $\rightarrow$ and $\Rightarrow$ is that $\rightarrow$ considers sequences of actions including τ, whereas $\Rightarrow$ only considers sequences of visible actions.

When the behavior of a service is analyzed, we will consider strongly connected components of an LTS.

Definition 2.1.2 (strongly connected component (SCC)).
Let $TS = (Q, \Sigma, \delta, q_0)$ be an LTS. Two states $q, q' \in Q$ of TS are *mutually reachable* iff $q \xrightarrow{*} q'$ and $q' \xrightarrow{*} q$. Mutually reachability is an equivalence relation on states of an LTS, and its equivalence classes are *strongly connected components* (SCCs). An SCC S is a *terminal strongly connected component* (TSCC) iff no state of another SCC is reachable from any state of S. ⌟

Figure 2.1 shows four LTSs; for example, R has five states r0, …, r4 and r0 $\xRightarrow{a\ b}$ r3. Each state of R is an SCC; states r3 and r4 are TSCCs.

Given two LTSs P and R, we are interested whether they are *equivalent*. Many preorders and equivalence notions to relate P and R exists in the literature—see the work of Van Glabbeek [Gla93, Gla01], for instance. We introduce the well-known relations of *strong simulation* [Mil89], *strong bisimulation* [Par81], and *branching bisimulation* [GW96].

A strong simulation relation (simulation for short) of P by R demands that every transition of P can be mimicked by an equally-labeled transition of R. A simulation treats τ-actions like any other action.

Definition 2.1.3 (simulation, bisimulation).
Let $P = (Q, \Sigma, \delta, q_0)$ and $R = (Q', \Sigma', \delta', q_0')$ be LTSs. A binary relation $\varrho \subseteq Q \times Q'$ is a *simulation relation of P by R* iff

- $(q_0, q_0') \in \varrho$;
- for every $(q_1, q_1') \in \varrho$, $a \in \Sigma \cup \{\tau\}$, $q_2 \in Q$ such that $q_1 \xrightarrow{a} q_2$ in P, there exists $q_2' \in Q'$, such that $q_1' \xrightarrow{a} q_2'$ in R and $(q_2, q_2') \in \varrho$.

R simulates P iff there exists a simulation relation ϱ of P by R. If, P and R are LTSs with final states and, for all $(q, q') \in \varrho$, $q \in \Omega_P$ iff $q' \in \Omega_R$, then *ϱ respects final states*. If ϱ and ϱ^{-1} are simulation relations (that respect final states), then ϱ is a *bisimulation relation* (that respects final states). ⌟

Consider again Figure 2.1. The LTS Q simulates the LTS P using simulation relation $\varrho = \{(\mathsf{p0}, \mathsf{q0}), (\mathsf{p1}, \mathsf{q1}), (\mathsf{p2}, \mathsf{q1}), (\mathsf{p3}, \mathsf{q2}), (\mathsf{p4}, \mathsf{q3})\}$. Furthermore, S simulates P, Q simulates S and vice versa, and no other simulation relation holds. In fact, the LTSs Q and S are even bisimilar. As a counterexample, P does not simulate Q: We had to relate states $(\mathsf{q0}, \mathsf{p0})$, $(\mathsf{q1}, \mathsf{p1})$, and $(\mathsf{q1}, \mathsf{p2})$; however, $\mathsf{q1} \xrightarrow{\mathsf{b}}$ and $\mathsf{q1} \xrightarrow{\mathsf{c}}$, but $\mathsf{p1} \not\xrightarrow{\mathsf{c}}$ and $\mathsf{p2} \not\xrightarrow{\mathsf{b}}$.

There may exist several simulation relations of P by R. Throughout this thesis, we shall *always* confine to a particular one that we call the *minimal simulation relation*. It restricts P and R to their *reachable states*. This relation is only uniquely defined for the case where R is deterministic. For example, S simulates Q, but there is no unique minimal simulation relation of Q by S, because the transition $(\mathsf{q1}, \mathsf{b}, \mathsf{q2})$ can be mimicked by two transitions of S, viz., $(\mathsf{s1}, \mathsf{b}, \mathsf{s2})$ and $(\mathsf{s1}, \mathsf{b}, \mathsf{s3})$.

Definition 2.1.4 (minimal simulation).
A *minimal simulation relation* ϱ of P by R is the smallest simulation relation of P by R, i. e., $\varrho \subseteq \varrho'$, for all simulation relations ϱ' of P by R. ⌟

An equivalence notion weaker than bisimulation is branching bisimulation. Branching bisimulation distinguishes (in contrast to bisimulation) visible actions from τ-actions. We define a branching bisimulation relation that respects final states.

Definition 2.1.5 (branching bisimulation).
Let $P = (Q, \Sigma, \delta, q_0, \Omega)$ and $R = (Q', \Sigma', \delta', q_0', \Omega')$ be LTSs with final states. P and R are *branching bisimilar* iff there exists a symmetric relation $\varrho \subseteq Q \times Q'$ such that

- $(q_0, q_0') \in \varrho$;
- for all $q_1, q_2 \in Q, q_1' \in Q'$ and for all $\alpha \in \Sigma \cup \{\tau\}$ such that $(q_1, q_1') \in \varrho$ and $q_1 \xrightarrow{\alpha} q_2$ in P implies
  - $\alpha = \tau$ and there exist $q_3', q_2' \in Q'$ such that $q_1' \overset{\epsilon}{\Longrightarrow} q_3' \wedge (q_3' \xrightarrow{\tau} q_2' \vee q_3' = q_2')$ and $(q_1, q_3'), (q_2, q_2') \in \varrho$; or
  - $\alpha \neq \tau$ and there exist $q_3', q_2' \in Q'$ such that $q_1' \overset{\epsilon}{\Longrightarrow} q_3' \wedge q_3' \xrightarrow{\alpha} q_2'$ and $(q_1, q_3'), (q_2, q_2') \in \varrho$.
- for each final state $q_1 \in \Omega$ with $(q_1, q_1') \in \varrho$ implies there exists $q_2' \in \Omega'$ such that $q_1' \overset{\epsilon}{\Longrightarrow} q_2'$ and $(q_1, q_2') \in \varrho$. ⌟

The LTSs Q and R in Figure 2.1 are branching bisimilar, but also the LTSs R and S. Suppose Q has final states q2 and q3. Then, Q and R are only branching bisimilar if R has final states r3 and r4

2.2. Basic definitions on Petri nets

Petri nets [Rei85, Mur89, DR98] consist of two kinds of nodes, *places* and *transitions*, and a *flow relation* on nodes. Graphically, a place is represented by a circle, a transition by a box, and the flow relation by directed arcs between them. Whilst transitions represent dynamic elements—for example, an activity of a service—places represent static elements—for example, a condition to perform an activity of a service. A *state* of a Petri net is represented by a marking, which is a distribution of tokens over the places. Graphically, a token is depicted by a black dot.

Definition 2.2.1 (Petri net).
A *Petri net* $N = (P, T, F, m_0)$ consists of

- two finite and disjoint sets P of *places* and T of *transitions*;
- a *flow relation* $F \subseteq (P \times T) \cup (T \times P)$; and
- an *initial marking* m_0, where a marking is a mapping $m : P \to \mathbb{N}$. ⌟

When referring to several Petri nets we use indices to distinguish the constituents of different Petri nets whenever necessary; for instance, P_N refers to the set of places of a Petri net N.

Let $x \in P \cup T$ be a node of a Petri net N. As usual, the *pre-set* of x is denoted by ${}^\bullet x = \{y \mid (y, x) \in F\}$, and the *post-set* of x is denoted by $x^\bullet = \{y \mid (x, y) \in F\}$.

The sum $m_1 + m_2 : P \to \mathbb{N}$ of two markings m_1, m_2 of a Petri net N is defined by $(m_1 + m_2)(p) = m_1(p) + m_2(p)$, for all $p \in P$. We canonically extend the notion of a marking of N to supersets $Q \supseteq P$ of places; that is, for a mapping $m : P \to \mathbb{N}$, we extend m canonically to the marking $m : Q \to \mathbb{N}$ with $m(p) = 0$, for all $p \in Q \setminus P$. Analogously, a marking can be restricted to a subset $Q \subseteq P$ of the places of N. For a mapping $m : P \to \mathbb{N}$, the restriction of m to the places in Q is denoted by $m|_Q : Q \to \mathbb{N}$. We extend this restriction also to sets of markings. Let M be a set of markings of N and $Q \subseteq P$, then $M|_Q$ denotes the restriction of markings m of M to the places in Q. Finally, the *set of all possible markings* of a Petri net N is denoted by $\mathcal{M}(N)$.

A marking of a Petri net N is changed by the *firing* of a transition of N. A transition t is *enabled* at a marking m if there is a token on every place in t's pre-set. The firing of an enabled transition t yields a new marking m', which is derived from m by consuming (i. e., removing) a token from each place of t's pre-set and producing (i. e., adding) a token on each place of t's post-set.

Definition 2.2.2 (behavior of Petri nets, step).
Let $N = (P, T, F, m_0)$ be a Petri net. A transition $t \in T$ is *enabled* at a marking m, denoted by $m \xrightarrow{t}$, iff $m(p) > 0$, for all $p \in {}^\bullet t$. If t is enabled at m, it can *fire*, reaching a marking m', where

$$m'(p) = \begin{cases} m(p) - 1, & \text{if } p \in {}^\bullet t \setminus t^\bullet, \\ m(p) + 1, & \text{if } p \in t^\bullet \setminus {}^\bullet t, \\ m(p), & \text{otherwise.} \end{cases}$$

The firing of t is a (t-)*step* of N and denoted by $m \xrightarrow{t} m'$. ⌟

The behavior of a Petri net N can be enhanced from single steps to potentially infinite sequences of steps. A finite or infinite sequence of steps $m_1 \xrightarrow{t_1} m_2 \xrightarrow{t_2} \ldots$ is a *run* of N if $m_i \xrightarrow{t_i} m_{i+1}$ is a step of N, for all $i > 0$. A marking m' is *reachable* from a marking m, denoted by $m \xrightarrow{*} m'$, if there exists a finite (possibly empty) run $m_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{k-1}} m_k$ with $m = m_1$ and $m' = m_k$. Let $R_N(m) = \{m' \mid m \xrightarrow{*} m'\}$ be the set of markings reachable from a marking m of N. The set $R_N(m_0)$ contains all markings of N that are reachable from the initial marking m_0. It can be represented as a graph, called *reachability graph* of N, with the set $R_N(m_0)$ as its nodes and the transitions between these markings as its labeled edges. A reachability graph can be represented by an LTS.

An example of a Petri net is illustrated in Figure 2.2. In its initial marking $m_0 = [\mathsf{p0}]$ the transitions $\mathsf{t0}$ and $\mathsf{t1}$ are enabled. The firing of transition $\mathsf{t0}$ yields the marking $[\mathsf{p1}]$. So $[\mathsf{p0}] \xrightarrow{\mathsf{t0}} [\mathsf{p1}]$ is a run of N, and the marking $[\mathsf{p1}]$ is reachable from the marking $[\mathsf{p0}]$.

Next, we define some properties of Petri nets. The first property refers to the structure of N, whereas the other three properties refer to the behavior of N.

Definition 2.2.3 (Petri net properties).
A Petri net $N = (P, T, F, m_0)$ is

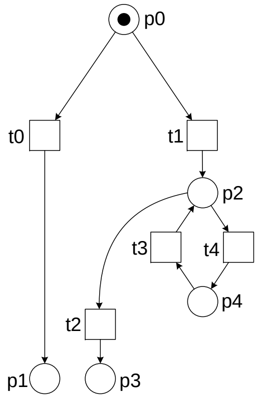


Figure 2.2.: An example Petri net N.

- *acyclic* iff the reachability graph of N is acyclic.
- *b-bounded* (or bounded for short) iff there exists a $b \in \mathbb{N}$ such that, for every reachable marking $m \in R_N(m_0)$, $m(p) \leq b$, for all $p \in P$.
- *live* iff, for every reachable marking $m \in R_N(m_0)$ and transition $t \in T$, there is a reachable marking $m' \in R_N(m)$ such that m' enables t.
- *quasi-live* iff, for all transitions $t \in T$, there is a reachable marking $m \in R_N(m_0)$ such that m enables t. ⌟

Boundedness of a Petri net is equivalent to have a finite set of reachable markings. Liveness ensures that, for every reachable marking m and every transition t, there exists a run from m_0 to a marking m' that enables t. A weaker property than liveness is quasi-liveness, which ensures that every transition is at least enabled in a reachable marking. These properties can be verified using standard state-space verification techniques [CGP00].

The example Petri net N in Figure 2.2 contains a run $[\mathsf{p0}] \xrightarrow{\mathsf{t1}} [\mathsf{p2}] \xrightarrow{\mathsf{t4}} [\mathsf{p4}] \xrightarrow{\mathsf{t3}} [\mathsf{p2}]$. Thus, N contains a cycle. The Petri net N is 1-bounded and quasi-live, but it is not live.

2.3. Modeling service behavior with open nets

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 the one service has to be connected with an output channel of the other service. Asynchronous message passing means that communication is non-blocking; 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.

We model services as *open nets*, which have been introduced as 'open workflow nets' in [MRS05]. An open net is a Petri net as defined in the previous section. As Petri nets have proved to be successful for the modeling of business processes and workflows (see the work of Van der Aalst [Aal98, AH02], for instance), open nets can adequately model the control structure of a service. The set of final states of a service—that is, the states in which it may successfully terminate—is modeled by a set of final markings. The service interface is reflected by two disjoint sets of input and output places. Thereby, each input (output) place corresponds to an input (output) channel. An input place has an empty pre-set and is used for receiving messages from a distinguished channel, whereas an output place has an empty post-set and is used for sending messages via a distinguished channel.

Definition 2.3.1 (open net).
An *open net* $N = (P, T, F, P^I, P^O, m_0, \Omega)$ consists of a Petri net (P, T, F, m_0) and

- an *interface* $(P^I \cup P^O) \subseteq P$ defined as two disjoint sets P^I of *input places* and P^O of *output places* such that ${}^\bullet p = \emptyset$, for any $p \in P^I$ and $p^\bullet = \emptyset$, for any $p \in P^O$; and
- a set Ω of *final markings*.

We further require that in the initial and the final markings no interface place is marked; that is, we demand $m(p) = 0$, for all $m \in \Omega \cup \{m_0\}$ and all $p \in P^I \cup P^O$.⌟

Graphically, we represent an open net like a Petri net with a dashed frame around it. The interface places are depicted on the frame. Final markings have to be described separately.

On the first sight, it might not be intuitive that a final marking of an open net may enable a transition. However, a restriction to final markings that do not enable any transition would not affect our theory. Therefore, we decided to be as general as possible in our definition.

As our running example, consider the open net Bank in Figure 2.3 with the initial marking $m_0 = [\mathsf{p0}]$. The set of final markings is defined as $\Omega = \{[\mathsf{p1}], [\mathsf{p3}]\}$. The open net Bank has input places $P^I = \{\mathsf{ap}, \mathsf{i}\}$ and output places $P^O = \{\mathsf{as}, \mathsf{req}\}$.

In this thesis, we restrict ourselves to the service behavior. Hence, open nets model only the service behavior and abstract from other important aspects of services, such as quality of service or semantical aspects. In addition, we also *abstract from data*, because open nets are low-level Petri nets with undistinguishable black tokens. We will discuss these restrictions at the end of this chapter in more detail.

Open nets are a generalization of Van der Aalst's *workflow nets* (WFNs) [Aal98]. A WFN is a Petri net that is specially tailored towards the

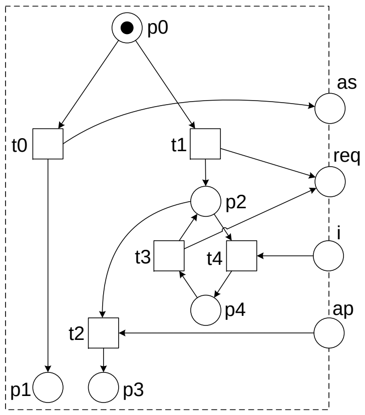


Figure 2.3.: An open net Bank modeling an online bank service. Bank either sends a customer his annual statements (t0) or it requires the customer to make an appointment with his bank consultant (t1). It accepts additional information being sent by the customer (t4), but it always reminds him to make an appointment (t3). If the customer agrees on an appointment (t2), Bank terminates.

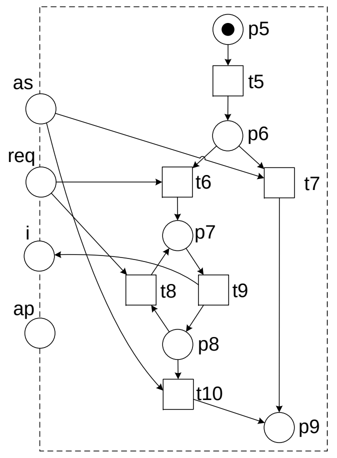
modeling of workflow processes. A WFN has a distinguished initial place and a distinguished final place, and every place and transition belongs to some path from the initial to the final place. Open nets do not follow those structural restrictions. A set of final markings is a more convenient way to model expected successful termination of a service. As a fundamental difference, WFNs do not have an interface. Martens extends the formalism of WFNs with an interface [Mar05]. The resulting class of Petri nets is called *workflow modules*. As workflow modules follow the structural restrictions of WFNs, open nets also generalize workflow modules.

Petri nets with an interface have also been considered in [Vog92, Che93, Kin97], for instance.

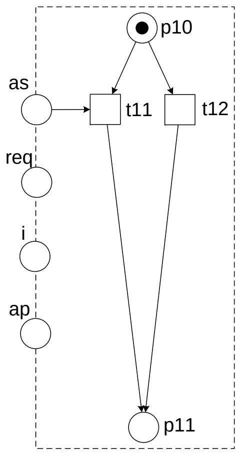
An open net N usually has transitions that are connected to an interface place and transitions that are not. The set $T^{IO} = \{t \mid \exists p \in P^I \cup P^O : t \in {}^\bullet p \cup p^\bullet\}$ defines the set of *interface transitions* of N, and $T \setminus T^{IO}$ defines the set of *internal transitions* of N.

If an open net N has an empty interface (i. e., $P^I = P^O = \emptyset$), then N is a *closed net*. A closed net can be used to model a service composition, for instance.

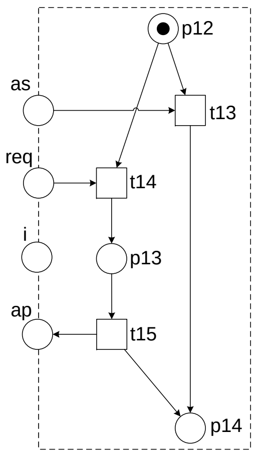
Definition 2.3.2 (inner subnet).
Let $N = (P, T, F, P^I, P^O, m_0, \Omega)$ be an open net, and let $P^{Int} = P \setminus (P^I \cup P^O)$ be the set of *internal places* of N. The *inner subnet of* N is defined by $inner(N) = (P^{Int}, T, F \cap ((P^{Int} \times T) \cup (T \times P^{Int})), \emptyset, \emptyset, m_0|_{P^{Int}}, \Omega|_{P^{Int}})$. ⌟



(a) Cust1



(b) Cust2



(c) Cust3

Figure 2.4.: Customers of Bank in Figure 2.3. Cust1 receives either the annual statements (t7) or an request for an appointment (t6). He always replies to such a request by sending some information to its customer consultant (t9) in the hope of receiving his annual statements eventually (t10). Cust2 either receives the annual statements (t11) or terminates immediately (t12). Cust3 receives either the annual statements (t13) or the request (t14). In case of a request, he makes an appointment with his customer consultant immediately (t15).

The inner subnet defines the Petri net that results from removing the interface places and their adjacent arcs from N. The behavior of N is basically the reachability graph of the inner subnet of N. Clearly, $\mathit{inner}(N)$ and N coincide if N is a closed net.

To decide boundedness of an open net, we assume arbitrary many tokens on each input place of N such that the enabledness of a transition of N does not depend on the interface places. Consequently, an open net is *bounded* if and only if its inner subnet $\mathit{inner}(N)$ is bounded.

Two open nets N_1 and N_2 may have the same set of interface places—that is, $P_1^I = P_2^I$ and $P_1^O = P_2^O$. In this case, they are *interface equivalent*.

The inner subnet $\mathit{inner}(\mathsf{Bank})$ of the open net Bank in Figure 2.3 is the Petri net N in Figure 2.2. As this Petri net is bounded, the open net Bank is bounded as well.

Now we complete our running example by introducing three customer services: the open nets Cust1, Cust2, and Cust3. The three open nets are depicted in Figure 2.4. As their sets of final states, we fix the singleton sets $\Omega_{\mathsf{Cust1}} = \{[\mathsf{p9}]\}$, $\Omega_{\mathsf{Cust2}} = \{[\mathsf{p11}]\}$, and $\Omega_{\mathsf{Cust3}} = \{[\mathsf{p14}]\}$, respectively. Each open net has input places $P^I = \{\mathsf{as}, \mathsf{req}\}$ and output places $P^O = \{\mathsf{ap}, \mathsf{i}\}$; that is, the three customers are interface equivalent.

2.4. Composition of open nets

The general idea of SOC is to use services as building blocks for designing complex services. To this end, services have to be composed; that is, pairs of input and output channels of these services are connected. Communication between these services is achieved by exchanging messages via these connected channels. Composing two open nets is modeled by fusing pairwise equally labeled input and output places. Such a fused interface place models a connected channel, and a token on such an interface place corresponds to a pending message in the respective channel.

For the composition of open nets we assume that all sets of transitions are pairwise disjoint and every internal place of an open net is not contained in the set of places of any other open net. This can be achieved easily by renaming. In contrast, the interfaces intentionally overlap. For a reasonable concept of composition of open nets, however, it is convenient to require that all communication is bilateral and directed; that is, every interface place p of N has only one open net that sends into p and one open net that receives from p. Thereby the sending open net has the output place, and the receiving open net has the corresponding equally labeled input place. We refer to open nets that fulfill these properties as interface compatible.

Definition 2.4.1 (interface compatible open nets).
Let $N_1 = (P_1, T_1, F_1, P_1^I, P_1^O, m_{01}, \Omega_1)$ and $N_2 = (P_2, T_2, F_2, P_2^I, P_2^O, m_{02}, \Omega_2)$ be two open nets with $T_1 \cap T_2 = \emptyset$, $P_1^{Int} \cap P_2 = \emptyset$, and $P_2^{Int} \cap P_1 = \emptyset$. If only input places of one open net overlap with output places of the other open net, i. e., $P_1^I \cap P_2^I = \emptyset$ and $P_1^O \cap P_2^O = \emptyset$, then N_1 and N_2 are *interface compatible*. ⌟

We compose two open nets N_1 and N_2 by merging input places of N_1 with equally labeled output places of N_2 (and vice versa); that is, composition corresponds to *place fusion*, which is well-known in the theory of Petri nets. Therein, bilateral and directed communication between N_1 and N_2 is guaranteed. Composition of N_1 and N_2 results in an open net again.

Definition 2.4.2 (composition of open nets).
Let $N_1 = (P_1, T_1, F_1, P_1^I, P_1^O, m_{01}, \Omega_1)$ and $N_2 = (P_2, T_2, F_2, P_2^I, P_2^O, m_{02}, \Omega_2)$ be two interface compatible open nets. The *composition* $N = N_1 \oplus N_2$ is the open net $(P, T, F, P^I, P^O, m_0, \Omega)$ defined as:

- $P = P_1 \cup P_2$;
- $T = T_1 \cup T_2$;
- $F = F_1 \cup F_2$;

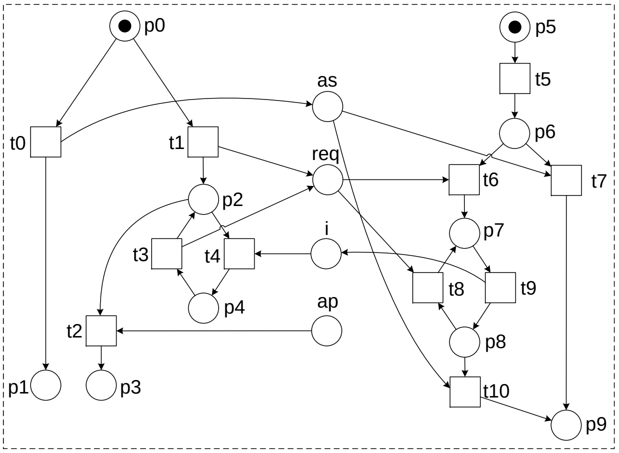


Figure 2.5.: Composition of open nets Bank and Cust1. The interface place ap becomes internal in the composition, but it is never marked.

- $P^I = (P_1^I \cup P_2^I) \setminus (P_1^O \cup P_2^O)$;
- $P^O = (P_1^O \cup P_2^O) \setminus (P_1^I \cup P_2^I)$;
- $m_0 = m_{01} + m_{02}$; and
- $\Omega = \{m_1 + m_2 \mid m_1 \in \Omega_1 \wedge m_2 \in \Omega_2\}$. ⌟

The initial marking of the composition is the sum of the initial markings of N_1 and N_2, and the set of final markings of the composition is the Cartesian product of the sets of final markings of N_1 and N_2. This is reasonable, because Definition 2.3.1 ensures that the only shared constituents of N_1 and N_2, the interface places, are not marked in the initial or final markings.

In the example, each of the customers in Figure 2.4 is interface compatible with the online bank in Figure 2.3. Hence, we can compose each customer with the online bank by merging equally labeled interface places. The resulting composition for Bank and Cust1 is the closed net in Figure 2.5.

The definition of final markings of the composition may result in unreachable final markings. As an example, the composition in Figure 2.5 has a final marking [p3, p9]. This marking is not reachable.

To apply composition to an arbitrary number of open nets, we require these open nets to be pairwise interface compatible. This requirement ensures bilateral communication, as for a third open net N_3, a communication taking place inside the composition of open nets N_1 and N_2 is internal matter.

Open net composition is commutative and associative. Thus, composition of a set of open nets can be broken into iterative pairwise composition.

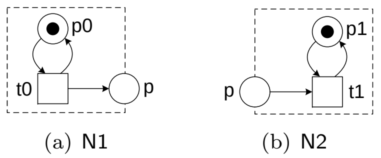


Figure 2.6.: Motivation for b-limited communication: Although N1 and N2 are 1-bounded open nets, their composition N1 $\oplus$ N2 is unbounded due to place p.

Lemma 2.4.3 (composition is commutative and associative).
For (pairwise) interface compatible open nets N_1, N_2, and N_3 holds:

- The composition of open nets is commutative, i. e., $N_1 \oplus N_2 = N_2 \oplus N_1$.
- The composition of open nets is associative, i. e., $N_1 \oplus N_2 \oplus N_3 = (N_1 \oplus N_2) \oplus N_3 = N_1 \oplus (N_2 \oplus N_3)$. ⌟

In this thesis, we will often consider interface compatible open nets N_1 and N_2 such that their composition results in a closed net. We refer to those open nets as *partners*. The notion of partners is also known as *syntactical compatibility* [Mar05] or *strong structural compatibility* [DW07].

Boundedness of an open net concerns the inner subnet of an open net. The composition of two bounded open nets may, however, result in an unbounded open net, because tokens may accumulate on the prior interface places. An example is illustrated in Figure 2.6. To achieve a bounded open-net composition, we have to restrict the number of tokens at those interface places. To this end, we define a notion of boundedness for interface places, which has been introduced in [LMW07b].

Definition 2.4.4 (*b*-limited communication).
Let $N = (P, T, F, P^I, P^O, m_0, \Omega)$ be an open net with $N = N_1 \oplus N_2$, and let $b \in \mathbb{N}$. Open net N has *b-limited communication* iff, for all reachable markings $m \in R_N(m_0)$, $m(p) \leq b$, for all $p \in (P_1^I \cup P_2^I \cup P_1^O \cup P_2^O) \subseteq P$. ⌟

If two open nets N_1 and N_2 are bounded and their composition N has b-limited communication, then N has only finitely many reachable markings.

As an example, each customer in Figure 2.4 is a partner of the online bank in Figure 2.3, and the composition of any of the three customers and the online bank has 1-limited communication.

2.5. Modeling multiparty contracts with open nets

In Section 1.2.1, we introduced the notion of a (multiparty) contract as a specification of an interorganizational cooperation among multiple parties. A contract

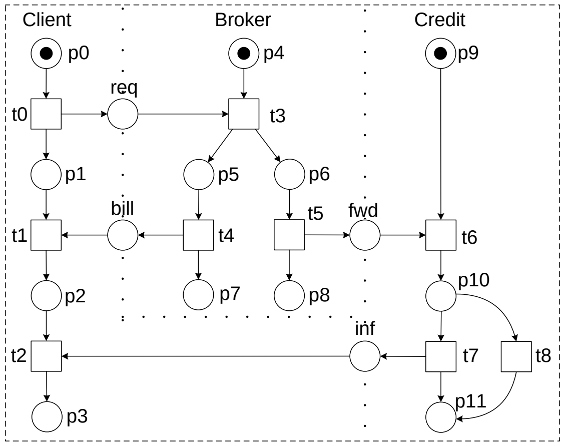


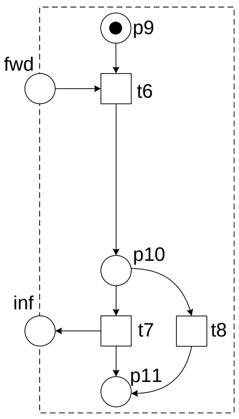
Figure 2.7.: A contract for a credit request that involves three parties: a client Client, a broker Broker, and a credit institute Credit.

corresponds to the 'rules of engagement' the parties agreed on. As in case of a service, we restrict ourselves to the behavioral aspect of a contract and abstract from all other aspects. The resulting contract models the basic interaction structures among different parties. In this section, we formalize contracts with open nets. The formalization follows [ALM$^+$09].

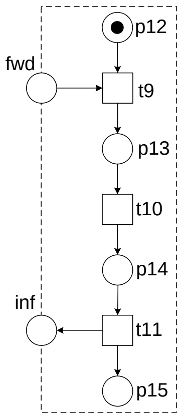
Basically, we see a contract as a closed net N, where every transition is assigned to one of the involved parties $A_1, \ldots, A_k$. We impose only one restriction: If a place is accessed by more than one party, it should act as a directed bilateral communication place. This restriction reflects the fact that a party's public view of the contract is a service again. A contract N can be cut into parts $N_1, \ldots, N_k$, each representing the agreed public view of a single party A_i $(1 \le i \le k)$. Hence, we define a contract as the composition of the open nets $N_1, \ldots, N_k$.

Definition 2.5.1 (contract for k parties, public view).
Let $\mathcal{A} = \{A_1, \ldots, A_k\}$ be a set of parties. Let $\{N_1, \ldots, N_k\}$ be a set of pairwise interface compatible open nets such that $N = N_1 \oplus \cdots \oplus N_k$ is a closed net. Then the closed net N is a *contract for $\mathcal{A}$*. For $i = 1, \ldots, k$, open net N_i is the *public view of A_i in N*. ⌟

An example contract is illustrated in Figure 2.7. The closed net models a credit request. It involves three parties: a *client* who requests at a broker for a credit; a *broker* who receives the client's credit request, forwards it to a credit institute, and sends the bill to the client; and a *credit institute*, which sends information to the client if it is interested. The three parties are modeled by the three open nets Client, Broker, and Credit. The dotted swimlanes in the figure show the partitioning of transitions over the parties involved in the contract.



(a) public view Credit



(b) private view Credit′

Figure 2.8.: Public view and private view of the credit service in Figure 2.7.

The respective interface places are depicted along the swimlanes. The set of final markings of the open nets is defined to be $\Omega_{\mathsf{Client}} = \{[\mathsf{p2}], [\mathsf{p3}]\}$, $\Omega_{\mathsf{Broker}} = \{[\mathsf{p7}, \mathsf{p8}]\}$, $\Omega_{\mathsf{Credit}} = \{[\mathsf{p11}]\}$, and according to Definition 2.4.2 for the Contract $\Omega = \{[\mathsf{p2}, \mathsf{p7}, \mathsf{p8}, \mathsf{p11}], [\mathsf{p3}, \mathsf{p7}, \mathsf{p8}, \mathsf{p11}]\}$.

A *private view* N' is an implementation of a public view N. Technically, N' is an (arbitrary) open net such that N and N' are interface equivalent.

Figure 2.8(a) recalls the public view of the service Credit in Figure 2.7. A possible private view Credit′ of this open net is shown in Figure 2.8(b). The service decides to be more customer-oriented. It receives the client request (t9), then gathers information (t10), and finally sends information to each client (t11). The set of final markings of Credit′ is defined as $\{[\mathsf{p15}]\}$. Credit′ and Credit are interface equivalent.

For the sake of simplicity, this example is rather atypical in the sense that a private view usually tends to have much more activities than the public view. However, Credit′ is less restrictive to the client than Credit, as it will always send information.

2.6. Behavioral compatibility of open nets

So far we defined composition of open nets and provided with the notion of interface compatibility a syntactical criterion for describing a correct composition. Interface compatibility is only a necessary condition for correctness of a composition. In addition, a composition has to be behaviorally compatible—for example, deadlock-free or sound [HSV03]. If a composition is both, interface and behaviorally compatible, then it is *compatible*.

In Section 2.6.1, we introduce some behavioral properties for open nets. In Section 2.6.2, we combine these properties and define several notions of behavioral compatibility.

2.6.1. Behavioral properties of open nets

We introduce with deadlock freedom, weak termination, covering of open-net nodes, and strict termination four behavioral properties of open nets.

Deadlock freedom and weak termination

A minimal requirement to termination is the absence of deadlocks in an open net N. However, the absence of deadlocks does not guarantee that an open net will eventually terminate, because it does not exclude livelocks.

Definition 2.6.1 (deadlock, livelock).
Let $N = (P, T, F, P^I, P^O, m_0, \Omega)$ be an open net.

- A reachable, non-final marking $m \in R_N(m_0) \setminus \Omega$ of N is a *deadlock* iff no transition of N is enabled at m. If N has no deadlock, then it is *deadlock-free*.
- Open net N has a *livelock* iff there is a reachable marking $m \in R_N(m_0)$ of N such that every marking $m' \in R_N(m)$ is not a deadlock and not a final marking. ⌟

The composition $\mathsf{Bank} \oplus \mathsf{Cust1}$ in Figure 2.5 is deadlock-free for the defined set $\Omega_{\mathsf{Bank}\oplus\mathsf{Cust1}} = \{[\mathsf{p1}, \mathsf{p9}], [\mathsf{p3}, \mathsf{p9}]\}$ of final markings, but it contains a livelock. In the marking $[\mathsf{p2}, \mathsf{p7}]$, only the infinite run $[\mathsf{p2}, \mathsf{p7}] \xrightarrow{\mathsf{t9}} [\mathsf{p2}, \mathsf{p8}, \mathsf{i}] \xrightarrow{\mathsf{t4}} [\mathsf{p4}, \mathsf{p8}] \xrightarrow{\mathsf{t3}} [\mathsf{p2}, \mathsf{p8}, \mathsf{req}] \xrightarrow{\mathsf{t8}} [\mathsf{p2}, \mathsf{p7}] \ldots$ is enabled, and hence neither a deadlock nor a final state is reachable from the marking $[\mathsf{p2}, \mathsf{p7}]$. The reason is that Cust1 always replies to each request of the Bank with the sending of additional information in the hope that these information satisfy the customer consultant. This is, however, never the case. Transition t10 cannot be enabled at any reachable marking of $\mathsf{Bank} \oplus \mathsf{Cust1}$. The composition $\mathsf{Bank} \oplus \mathsf{Cust3}$ is deadlock-free as well, whereas $\mathsf{Bank} \oplus \mathsf{Cust2}$ can reach a deadlock $[\mathsf{p2}, \mathsf{p11}, \mathsf{req}]$ from its initial marking.

Deadlocks and livelocks in a reachability graph (i. e., LTS) can be characterized by the help of SCCs (cf. Definition 2.1.2). In fact, a deadlock or livelock is a TSCC which does not contain a final state. If this TSCC is a singleton state without a self-loop, we have a deadlock; otherwise, we have a livelock.

In the sequel, we define *weak termination*, a termination criterion that guarantees the absence of deadlocks and of livelocks in an open net N. In the computation tree logic (CTL) [CE82, CES86], weak termination can be expressed as $AG\,EF\,m_f$, $m_f \in \Omega$; that is, from every reachable marking of N, it is always possible to reach a final marking of N. Weak termination coincides with the notion of *soundness* [HSV03] that has been introduced for workflow nets.

Definition 2.6.2 (weak termination).
An open net N *weakly terminates* iff, for all reachable markings $m \in R_N(m_0)$, there is a marking $m_f \in \Omega_N$ with $m \xrightarrow{*} m_f$. ⌟

Only the composition of the online bank Bank and the customer Cust3 weakly terminates; Bank $\oplus$ Cust2 may deadlock, and Bank $\oplus$ Cust1 in Figure 2.5 may livelock.

Weak termination is a stricter termination criterion than deadlock freedom, as it excludes livelocks. For bounded open nets, a livelock corresponds to a cycle in the reachability graph. Hence, for bounded and acyclic open nets in the sense of Definition 2.2.3 (i.e., open nets with an acyclic reachability graph), the notions weak termination and deadlock freedom coincide.

Lemma 2.6.3 (relating deadlock freedom and weak termination).
Let N be an acyclic and bounded open net. N weakly terminates iff N is deadlock-free. ⌟

Covering open-net nodes

Besides termination it is obviously desirable that the functionality of a service can potentially be used by other services. In other words, we want that a (sub)set of activities of a service is not dead. On the modeling level, we require that certain open-net nodes can be covered.

Definition 2.6.4 (cover open-net node).
Let $N = (P, T, F, P^I, P^O, m_0, \Omega)$ be an open net. Let an *α-run* be a run that starts in the initial marking m_0 of N. Open net N *covers* a

- place $p \in P$ iff some α-run of N includes a marking m with $m(p) \geq 1$;
- transition $t \in T$ iff some α-run of N includes a t-step;
- set $Y \subseteq P \cup T$ iff, for each $y \in Y$, there is an α-run that covers y. ⌟

The idea of covering open-net nodes is inspired by the notion of *classical soundness* for workflow nets [Aal98]. Classical soundness ensures that a workflow net weakly terminates and each transition is covered. Definition 2.6.4 allows to cover arbitrary nodes of an open net. An open net N that covers all its transitions is by Definition 2.2.3 quasi-live.

As an example, the composition Bank $\oplus$ Cust1 in Figure 2.5 covers all open-net nodes except ap, p3, t2, and t10.

Strict termination

The notion of a final marking of an open net, as introduced in Definition 2.3.1, is very general, as it allows an open net to receive or to send messages being in a final marking. Even internal transitions may be enabled at a final marking. In industrial service description languages, final states are usually more restrictive;

for example, in WS-BPEL, it is not possible to define a final state in which the service can send, receive, or perform an internal transition. Inspired by this restriction to final states, we introduce *strict termination.* Strict termination guarantees that no transition can be enabled at any final marking; that is, the restriction of every final marking of an open net N to its inner subnet $\mathit{inner}(N)$ does not enable any transition in $\mathit{inner}(N)$.

Definition 2.6.5 (strict termination).
Let N be an open net, and let $\mathit{inner}(N) = (P, T, F, \emptyset, \emptyset, m_0, \Omega)$. Open net N *strictly terminates* iff, for all final markings $m \in \Omega$ of $\mathit{inner}(N)$, m does not enable any transition $t \in T$ in $\mathit{inner}(N)$. ⌟

Note that strict termination does not imply weak termination.

The open nets of the online bank and the customers strictly terminate. For a violation of this property, consider the open net Client in the contract in Figure 2.7. In final marking [p2] transition t2 is enabled in inner(Client).

2.6.2. Open-net properties and strategies

Now we combine the four properties deadlock freedom, weak termination, cover, and strict termination that we defined in the previous subsection and. In addition, we add quasi-liveness as a fifth property and define the notion of an open-net property.

Definition 2.6.6 (open-net property, X-open net).
Let N be an open net, and let $Y \subseteq P \cup T$ be a set of nodes of N.

- Each $x \in \mathit{Prop} = \{$deadlock freedom, weak termination, cover(Y), strict termination, quasi-liveness$\}$ is an *open-net property* .
- Let $X \subseteq \mathit{Prop}$ be a set of open-net properties. Open net N is an X-*open net* iff N satisfies all properties as given in X. ⌟

For a reasonable concept of open-net properties, we only consider 9 subsets of *Prop*; see Table 2.1. The motivation is that deadlock freedom is a minimal requirement for the correctness of services. Hence every subset of *Prop* in Table 2.1 contains at least deadlock freedom or the stricter termination criterion weak termination. Weak termination ensures that a final state can be reached, and strict termination further restricts the set of final markings. Therefore, strict termination has to be combined with weak termination. In contrast, a deadlock-free open net may never reach a final state (due to a livelock). Hence, restricting final states for deadlock freedom is not reasonable. Finally, we consider either quasi-liveness or cover(Y).

In Section 2.4, we introduced the notion of a partner for two interface compatible open nets N and S. Now we extend this notion such that the composition of

Table 2.1.: Nine subsets of open-net properties considered in this thesis.

deadlock freedom	weak term.	strict term.	cover(Y)	quasi-liveness
+	−	−	−	−
+	−	−	+	−
+	−	−	−	+
+	+	−	−	−
+	+	+	−	−
+	+	−	+	−
+	+	−	−	+
+	+	+	+	−
+	+	+	−	+

N and S has b-limited communication and, in addition, satisfies a set X of open-net properties. In this case, we say that N and S are (interface and behaviorally) compatible, and we refer to N and to S as (X, b)-strategies.

Definition 2.6.7 (strategy).
Let N and S be bounded open nets that are partners, let $X \subseteq \mathit{Prop}$ be a set of open-net properties, and let $b \in \mathbb{N}$. If X contains strict termination, then N as well as S must be strictly terminating. Open net S is an (X, b)-*strategy* of N iff $N \oplus S$ is an X-open net, and $N \oplus S$ has b-limited communication. With $\mathit{Strat}_{X,b}(N)$ we denote the set of all (X, b)-strategies of N. ⌟

Strict termination is treated differently than the other open-net properties. The composition of two strictly terminating open nets is also strictly terminating. The other way around, the composition of two open nets that are not strictly terminating may be strictly terminate. So for a reasonable concept of strict termination, we require N *and* S to be strictly terminating.

We do not require that nodes of (X, b)-strategies S have to be covered, because (X, b)-strategies will be generated. In this case, every node of this (X, b)-strategy is covered by construction. However, the cover property violates symmetry. If X does not contain cover(Y), the notion of an (X, b)-strategy is as the notion of a partner *symmetric*; that is, if N is an (X, b)-strategy of S, then S is an (X, b)-strategy of N, too. This is a consequence of the commutativity of open-net composition (see Lemma 2.4.3). An open net S may cover all nodes of N; however, as S may contain transitions that cannot be enabled in $N \oplus S$, the converse does not hold in general. Thus, cover violates the symmetry property. In contrast, quasi-liveness ensures that all transitions of the composition $N \oplus S$ are covered, and hence it preserves the symmetry property.

The term strategy is used in the field of controller synthesis [RW87]. For open nets, it has been introduced in [Sch05], where the considered set X of open-net properties is restricted to deadlock freedom. In that paper, a strategy S serves as a controller for an open net N, meaning, S controls N in a way such that their composition will never reach a deadlock. Wolf [Wol09] extends the notion of a strategy to (X, b)-strategies by considering behavioral properties deadlock freedom, weak termination, and quasi-liveness.

If the set of (X, b)-strategies of an open net N is nonempty, then N is (X, b)-controllable.

Definition 2.6.8 (controllability).
Let N be a bounded open net, let X be a set of open-net properties for N, and let $b \in \mathbb{N}$. If $\mathit{Strat}_{X,b}(N) \neq \emptyset$, then N is (X, b)-*controllable*. ⌟

The open net Cust1 is a $(\{\text{deadlock freedom}\}, 1)$-strategy of the open net Bank, and Cust3 is a $(\{\text{weak termination}\}, 1)$-strategy of the open net Bank. Hence, the open nets Cust1 and Bank are $(\{\text{deadlock freedom}\}, 1)$-controllable, and Cust3 and Bank are $(\{\text{weak termination}\}, 1)$-controllable.

Throughout this thesis, we will restrict ourselves to *bounded open nets* as already done in Definitions 2.6.7 and 2.6.8. Thus, we require partners to be bounded and to satisfy b-limited communication. For unbounded open nets, controllability has been proved to be undecidable even for the simplest class of $\{\text{deadlock freedom}\}$-strategies [MSSW08]. As services in practice are finite state services, this restriction does not harm our approach.

To simplify the notation, we fix a bound b in the following and write X-strategy instead of (X, b)-strategy. In the examples, the bound will usually be $b = 1$.

2.7. Modeling service behavior with service automata

In the previous sections, we introduced open nets. Open nets are well-suited to model service behavior, as they have a more implicit and compact model than LTSs, for instance. As open nets have a formal foundation, they can be used to analyze service models. However, most of the analysis techniques, we will present in the forthcoming chapters, deal only with the reachable markings of an open net and do not make use of its structural information—for example, the explicit representation of concurrency.

For that reason, we introduce *service automata* in this section. A service automaton is an automaton-based representation of an open net taking into account the concept of asynchronous message passing. In Section 2.7.1, we define a normal form for open nets, where every transition is connected to at most one interface place. Afterwards, in Section 2.7.2, we show that every open net N in normal form can be translated forth and back into a service automaton without affecting the set of X-strategies of N.

2.7.1. A normal form for open nets

We aim at translating open nets into service automata, a special kind of automata. As the main difference between both formalisms, the explicit sending and receiving of messages from an interface place in an open net will be in a service automaton modeled implicitly by the help of transition labels. An open net is capable to send and receive *several* messages by firing a *single* transition. To avoid a *set* of labels at each transition in the corresponding service automaton, we introduce a *normal form* for open nets, where every transition is connected to at most one interface place. Only open nets being in normal form will be translated into service automata.

Definition 2.7.1 (normal form for open nets).
An open net N is in *normal form* iff every transition of N is connected to at most one interface place, i. e., $|({}^\bullet t \cup t^\bullet) \cap (P^I \cup P^O)| \leq 1$, for all $t \in T$. ⌟

We define a mapping that assigns a label to each open-net transition. These labels are used to represent the LTS of an open net N. A label denotes to which interface place a transition is connected. In case the label is an internal transition, we define the transition label to be τ. Hence, the label denotes the communication behavior of N and abstracts from all other aspects.

Definition 2.7.2 (transition label).
The *transition labels* of an open net N in normal form are defined by the mapping $l : T \to P^I \cup P^O \cup \{\tau\}$ ($\tau \notin P^I \cup P^O$), such that $l(t)$ is the unique interface place adjacent to t if one exists, and $l(t) = \tau$ if t is not adjacent to any interface place.⌟

Consider the open net Cust1 in Figure 2.4(a). Examples for transition labels are $l(\mathsf{t5}) = \tau$, $l(\mathsf{t6}) = \mathsf{req}$, and $l(\mathsf{t7}) = \mathsf{as}$.

An open net N can be transformed easily into an open net in normal form. Intuitively, we relabel each interface place x to p_x and add a transition t_x and a new interface place x to N. If x was an output place, we add two arcs (p_x, t_x) and (t_x, x). For an input place, the two arcs have the opposite direction. In addition, we add for each input place a complementary place $\overline{p}_x$ that contains in the initial marking b tokens. This ensures that the inner subnet of the normalized open net has the same message bound than *inner*(N). The sum of tokens in p_x and $\overline{p}_x$ is always b, for all reachable markings. We also have to adapt the set of final markings of N: In every final marking, all complementary places $\overline{p}_x$ must contain b tokens guaranteeing that p_x is not marked. Figure 2.9 illustrates the construction for $b = 1$. As the construction is straight-forward, we refrain from a formal definition.

This transformation guarantees that *normal*(N) and N are interface equivalent, and every transition of *normal*(N) is connected to at most one interface place. The following theorem proves that the normal form of an open net N preserves all X-strategies of N.

Theorem 2.7.3 (correctness of the normalization).
Let N be an open net, and let $X \subseteq \mathit{Prop}$ be a set open-net properties of N. An open net S is an X-strategy of N iff S is an X-strategy of *normal*(N). ⌟

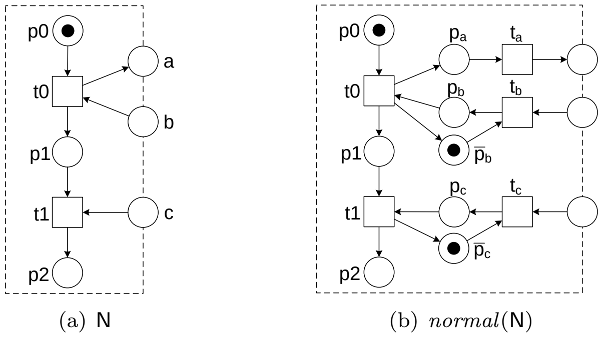


Figure 2.9.: A 1-bounded open net N, which is not in normal form, and its corresponding open net *normal*(N) in normal form. For each interface place a, b, and c of N, a buffer place $\mathsf{p_a}$, $\mathsf{p_b}$, and $\mathsf{p_c}$, respectively, is added to *normal*(N). Places $\overline{\mathsf{p}}_\mathsf{b}$ and $\overline{\mathsf{p}}_\mathsf{c}$ are the complementary places of $\mathsf{p_b}$ and $\mathsf{p_c}$, respectively. The inner subnet of both, N and *normal*(N), is 1-bounded.

Proof.
From S being an X-strategy of N follows that $N \oplus S$ is a closed net. From $\mathit{normal}(N)$ being interface equivalent to N follows that $\mathit{normal}(N) \oplus S$ is a closed net, too. Hence, the construction of a normal open net $\mathit{normal}(N) \oplus S$ from the open net $N \oplus S$ without adding the complementary places corresponds to applying the Petri transformation rule 'fusion of series places' [Mur89]. This rule is known to preserve liveness and boundedness; hence, $N \oplus S$ has b-limited communication if and only if $\mathit{normal}(N) \oplus S$ does. Adding now the complementary places will clearly not affect the set of reachable markings. (Note that we only need these places to ensure boundedness of $\mathit{normal}(N)$.) Adding the complementary places adapts the set of final markings of N; that is, in every final marking of $\mathit{normal}(N)$, all complementary places must contain b tokens. As by Definition 2.3.1 interface places do not contain a token in a final marking, no transition in the post-set of a complementary place can be enabled at a final marking (follows from the construction of $\mathit{normal}(N)$). Thus, normalizing N does not affect strict termination. It is easy to see that $N \oplus S$ has a deadlock (livelock) if and only if $\mathit{normal}(N) \oplus S$ has a deadlock (livelock). Thus, deadlock freedom and weak termination are preserved in both directions. Next, we have to show that $N \oplus S$ is quasi-live if and only if $\mathit{normal}(N) \oplus S$ is quasi-live. If $N \oplus S$ is quasi-live, then we conclude from the fact that any reachable marking of $N \oplus S$ is also reachable in $\mathit{normal}(N) \oplus S$ (if we restrict the markings of $\mathit{normal}(N) \oplus S$ to the places of $N \oplus S$) and the construction of $\mathit{normal}(N)$ that $\mathit{normal}(N) \oplus S$ is quasi-live, too. Let now $\mathit{normal}(N) \oplus S$ be quasi-live. We can show that (due to boundedness) for every reachable marking m of $\mathit{normal}(N) \oplus S$, there is a marking m' with $m \xrightarrow{*} m'$ and the restriction

of m' to the places of $N \oplus S$ is reachable in $N \oplus S$. Hence, $N \oplus S$ is quasi-live. Finally, we have to show that a set Y of nodes of N is covered in $N \oplus S$ if and only if it is covered in $\mathit{normal}(N) \oplus S$. This fact follows the same argumentation than for quasi-liveness. □

By commutativity of open-net composition (see Lemma 2.4.3) we directly conclude that we can construct a normal open net for both, N and S, without affecting the relationship between their sets of X-strategies.

Corollary 2.7.4 (correctness of construction).
Let X be an open-net property of an open net N. An open net S is an X-strategy of N iff the open net $\mathit{normal}(S)$ is an X-strategy of the open net $\mathit{normal}(N)$. ⌟

Due to Corollary 2.7.4 we may consider only open nets in normal form in the rest of this thesis.

2.7.2. Service automata and their relationship to open nets

In this section, we introduce service automata and a translation of any normal open net into a service automaton.

A state of a service automaton is comparable to a marking of the inner subnet of an open net N. Interface transitions of N are modeled as annotations to the transitions of the service automaton according to the respective transition label of N. Hence, the alphabet of a service automaton consists of an input and of an output alphabet modeling the input and the output channels, respectively. Unlike an open net, a service automaton has no explicit concept of message channels. The message channels are taken care of in the definition of composition: A state of a composed service automaton consists of a state of each participating service automaton and a state of the bag of currently pending messages.

First, we define the notion of an *automaton*, which is refined to a service automaton. For the labeling, we fix a *universe* $\mathcal{MC}$ of message channels, with $\tau \notin \mathcal{MC}$.

Definition 2.7.5 (automaton).
An *automaton* $A = (Q, MC^I, MC^O, \delta, q_0)$ is a finite LTS (Q, Σ, δ, q_0) where the alphabet $\Sigma = MC^I \cup MC^O$ is divided into an input alphabet $MC^I \subseteq \mathcal{MC}$ and an output alphabet $MC^O \subseteq \mathcal{MC}$ such that $MC^I \cap MC^O = \emptyset$. ⌟

An x-labeled transition $(q, x, q') \in \delta$ is a *sending transition* if $x \in MC^O$, a *receiving transition* if $x \in MC^I$, and an *internal transition* if $x = \tau$. A preceding question mark and exclamation mark emphasizes whether $x \in MC^I$ and $x \in MC^O$ in the graphical representation of an automaton, respectively.

To model service behavior, we have to extend automata with a set of final states. The resulting model, *service automata*, has been introduced in [MS05].

Definition 2.7.6 (service automaton).
A *service automaton* $A = (Q, MC^I, MC^O, \delta, q_0, \Omega)$ consists of an automaton $(Q, MC^I, MC^O, \delta, q_0)$ and a set $\Omega \subseteq Q$ of *final states.* ⌟

Service automata are related to classical I/O automata [Lyn96]. In contrast to I/O automata, service automata communicate asynchronously rather than synchronously, and they do not require the explicit modeling of the states of message channels. For a detailed comparison of service automata with other automata models, we refer to [Mas09].

Every normal open net N can be transformed easily into a service automaton $SA(N)$. To this end, we calculate the reachability graph of the inner subnet of N and annotate every edge with the respective transition label. The input and the output alphabet of $SA(N)$ is defined by the input and by the output places of N, respectively.

Definition 2.7.7 (corresponding service automaton).
The *corresponding service automaton* $SA(N) = (Q, MC^I, MC^O, \delta, q_0, \Omega)$ of an open net N is defined as

- $Q = \{m \mid m \in R_{inner(N)}(m_0)\}$;
- $MC^I = P^I \wedge MC^O = P^O$;
- $(m, l(t), m') \in \delta$ iff $m \xrightarrow{t} m'$, for $m, m' \in R_{inner(N)}(m_0) \wedge t \in T_{inner(N)}$;
- $q_0 = m_0$; and
- $\Omega = \{m \mid m \in R_{inner(N)}(m_0) \cap \Omega_{inner(N)}\}$. ⌟

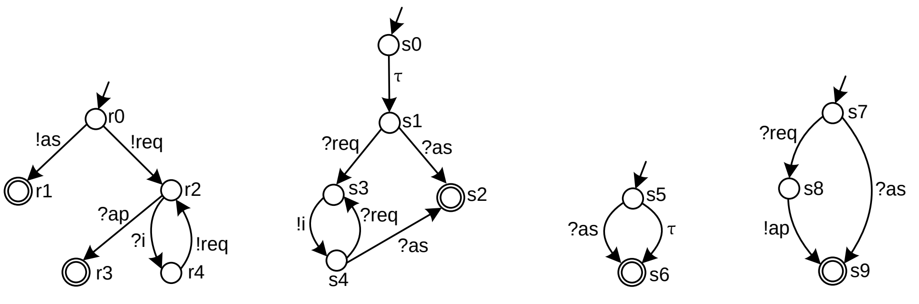
As an example, the corresponding service automata of the online bank (cf. Figure 2.3) and its three customers (cf. Figure 2.4) are illustrated in Figure 2.10. We use the standard graphical notations for automata and denote initial states by an inbound arrow and final states by double circles. In Figure 2.10(d), s7 is the initial state, s9 is the final state, and the states s7, s8, and s9 correspond to the markings [p12], [p13], and [p14] in Figure 2.4(c), respectively.

The translation of an open net into a service automaton according to Definition 2.7.7 guarantees that the reachability graph of *inner*(N) is isomorphic with $SA(N)$.

A service automaton A can, due to the transition labeling, be transformed easily into an open net $PN(A)$. The inner subnet of $PN(A)$ is then a state machine—that is, a Petri net where every transition has at most one state in its pre- and post-set.

Definition 2.7.8 (corresponding open net).
The *corresponding open net* $PN(A) = (P, T, F, P^I, P^O, m_0, \Omega_N)$ of a service automaton $A = (Q, MC^I, MC^O, \delta, q_0, \Omega)$ is defined as

- $P = Q \cup MC^I \cup MC^O$;



(a) SA(Bank) (b) SA(Cust1) (c) SA(Cust2) (d) SA(Cust3)

Figure 2.10.: Service automata of the open nets Bank, Cust1, Cust2, and Cust3.

- $T = \{(q, x, q') \mid (q, x, q') \in \delta\}$;
- $F = \quad \{(q, (q, x, q')), ((q, x, q'), q') \mid (q, x, q') \in \delta\}$
  $\cup \{(q, x, q'), x) \mid (q, x, q') \in \delta \wedge x \in MC^O\}$
  $\cup \{(x, (q, x, q')) \mid (q, x, q') \in \delta \wedge x \in MC^I\}$;
- $P^I = MC^I \wedge P^O = MC^O$;
- $m_0 = q_0$; and
- $\Omega_N = \{q \mid q \in \Omega\}$. ⌟

From the construction of $PN(A)$ in Definition 2.7.8, it is easy to see that there is an isomorphism between the states of A and the markings of the inner subnet of $PN(A)$.

Alternatively, it is also possible to translate a service automaton A into an open net $PN(A)$ by applying the theory of regions [BD98], for instance. That way, concurrency can be modeled explicitly in the resulting open net.

The composition of two service automata A and B is the service automaton of the composition $PN(A) \oplus PN(B)$ of their corresponding open nets $PN(A)$ and $PN(B)$. If the composition has an empty interface, then the corresponding service automaton is, in fact, an LTS.

Definition 2.7.9 (composition of service automata).
The *composition* $A \oplus B$ of service automata A and B is defined by $SA(PN(A) \oplus PN(B))$. ⌟

Let m be a reachable marking in $PN(A) \oplus PN(B)$. To model asynchronous communication, we model a state q_m of the composition $A \oplus B$ as a tripel (q_A, q_B, M). The states q_A and q_B correspond to a marking of *inner*$(PN(A))$ and of *inner*$(PN(B))$, respectively and M to a multiset (i. e., bag) of pending messages in m. Formally, $q_A = m|_{P_{inner(PN(A))}}$ (i. e., the restriction of m to

$\mathit{inner}(PN(A)))$, $q_B = m|_{P_{\mathit{inner}(PN(B))}}$ (i. e., the restriction of m to $\mathit{inner}(PN(B)))$, and $M(x) = m(x)$, for all places $x \in ((P^I_{PN(A)} \cup P^I_{PN(B)}) \cap (P^O_{PN(A)} \cup P^O_{PN(B)}))$ (i. e., the pending messages on the merged interface places of $PN(A)$ and $PN(B)$).

Sending a message x in $A \oplus B$ increases the multiplicity of the message x by 1 in the bag M; receiving a message x is only possible if the multiplicity of the involved message x is greater than 0 and, on occurrence, decreases this multiplicity by 1; an internal transition τ does not affect the bag M of pending messages.

As a consequence of Definitions 2.7.7–2.7.9, we can lift all definitions of open nets to service automata. As this is straight-forward, we do not present these definitions, but refer the interested reader to [Mas09].

The next theorem justifies that we can change arbitrarily between the two formalisms, open nets and service automata, without loosing information with respect to Strat_X.

Theorem 2.7.10 (forth and back translation preserves Strat_X).

- For any two open nets N_1, N_2 and any set $X \subseteq \mathit{Prop}$ of open-net properties holds: $N_1 \oplus N_2$ is an X-open net iff $\mathit{SA}(N_1) \oplus \mathit{SA}(N_2)$ satisfies X.
- For any two service automata A, B, open nets $PN(A), PN(B)$, and any set $X \subseteq \mathit{Prop}$ of open-net properties holds: $PN(A) \oplus PN(B)$ is an X-open net iff $A \oplus B$ satisfies X. ⌟

Correctness of this theorem is easily observed, as the reachable states of $\mathit{SA}(N_1) \oplus \mathit{SA}(N_2)$ and $\mathit{SA}(PN(\mathit{SA}(N_1)) \oplus PN(\mathit{SA}(N_2)))$ are isomorphic, and the reachable states of $A \oplus B$ and $\mathit{SA}(PN(A) \oplus PN(B))$ are isomorphic.

2.8. Discussion of the modeling restrictions

In the previous sections, we introduced open nets and, in addition, service automata as a formalism to model service behavior. The aim of this section is to evaluate the model of open nets and to justify design decisions we made so far. Afterwards, we will explain how services described by industrial service description languages can be transformed into our model open nets and how our analysis techniques can be applied in practice.

2.8.1. Justification of the design decisions in open nets

Four different aspects of service compatibility are distinguished in the literature [Pap07]: interface compatibility, behavioral compatibility, semantical compatibility, and QoS compatibility.

Open nets focus on the service behavior and are designed to analyze behavioral compatibility. Hence, open nets abstract from information related to the semantics and QoS of a service. In the model of open nets, we assume a unique label for each message channel. Composition of two open nets is defined by merging pairwise equally labeled input and output places. That way, interface compatibility of services is adequately modeled by open nets.

Extending open nets with information related to the semantics and QoS of a service seems not to be reasonable, because these information would complicate the analysis techniques that we will present in the following chapters. We think that for deciding semantical compatibility or QoS compatibility models different than open nets are required that are tailored towards the analysis of those compatibility notions.

As already mentioned in the introduction, this thesis focuses on the control-flow of services and abstracts from other also important aspects, such as data, time, and resources. The reason is mainly driven by analysis needs, because incorporating these aspects into the model would make the analysis much more complicated. In the following, we motivate why this is the case.

Data, time, and resources can be modeled with high-level Petri nets [Jen97], for instance. In case the data domain is finite, a high-level Petri net can be unfolded into an equivalent low-level Petri net with undistinguishable black tokens. However, unfolding typically results in huge Petri nets making the analysis hard or even impossible. Infinite data domains may be abstracted to a finite domain using techniques, such as abstract interpretation [CC77] and predicate abstraction [GS97]. If this is not possible, most analysis questions for such a Petri net are in general undecidable. Like for data, the explicit modeling of time makes the analysis more complex. However, an explicit modeling of time is often not necessary. Instead, a timed dependency between two activities can be modeled implicitly by a causality of the corresponding transitions.

If a service is executed, a running instance of this service is created. Several instances of a service may coexist. In this setting, a message being sent to the service has to be correlated to its corresponding instance. As another restriction of our model, we assume in this thesis that message correlation is taken care of elsewhere—for example, by using the correlation mechanisms in the context of WS-BPEL [Alv07]. This formalism could be modeled by using high-level Petri nets, for instance. Hence, we only consider a *single instance* of a service.

Summing up, from a practical point of view it would be beneficial to model all aspects of services. However, this would make the problems addressed in this thesis far more complex. Therefore, we decided to focus on the control-flow of services which can be adequately modeled with open nets.

Next, we show how services specified by industrial service description languages can be automatically translated into open nets.

2.8.2. Translating real-life services into open nets

In the forthcoming chapters, we will introduce analysis methods for services modeled as open nets. To justify the suitability of these methods, we have to apply them to real-life service models. In practice, services are usually not modeled by formalisms, such as Petri nets. Instead, a number of service description languages have been proposed by different industrial consortiums. Most prominent languages are WS-BPEL and BPMN. To obtain an open net from a service specified in any of such language, we need a formal semantics and a compiler to automatically translate services into an open net. In the following, we sketch some of the most prominent approaches.

For WS-BPEL, there exists a feature-complete open net semantics [Loh08] and a compiler, BPEL2oWFN, to translate a WS-BPEL process into an open net. With feature-complete we mean that the open-net semantics supports all concepts of WS-BPEL including the control flow, data flow, message flow, exception handling, and compensation handling. As mentioned in the previous subsection, we only translate a single instance of each WS-BPEL process, and therefore we do not need to correlate messages to process instances.

The compiler BPEL2oWFN is also capable of translating choreographies specified in the choreography description language BPEL4Chor [DKLW07] into a closed net [LKLR08]. That way, a choreography of WS-BPEL processes—for example, a service contract—can be automatically translated into a closed net.

There is also tool support to translate an open-net model of a service into a WS-BPEL (abstract) process [LK08]. Hence, a complete tool chain for translating WS-BPEL back and forth into open nets is available. So, all analysis methods for open nets can be used to analyze WS-BPEL processes as well. This is illustrated in Figure 1.4.

Besides this open-net semantics, there is another Petri-net semantics for WS-BPEL, which has been implemented in the tool BPEL2PNML [OVA$^+$07]. As its two main differences to BPEL2oWFN, this formal semantics is restricted to a subset of the constructs of WS-BPEL (i. e., those that have been part of BPEL4WS [And03], a predecessor version of WS-BPEL), and BPEL2PNML translates a BPEL process into a workflow net. That way, information about the interface is abstracted away. Workflow nets can be translated back into BPEL4WS using the tool WorkflowNet2BPEL4WS [AL08]. For a detailed comparison of the two Petri-net semantics, we refer to [LVOS08].

Furthermore, services specified in a service description languages other than WS-BPEL can be translated into Petri nets or other formalisms. For example, there exists tool support [DDO08] to translate BPMN into Petri nets.

If the service is described as a programming language like Java or C, a translation into a formal model becomes more complicated. However, there are also approaches like the one of [SCCS05] and of [SK07] that can translate such a service description into an automaton (or another formal model) using techniques that have proved themselves in the area of model checking [CGP00] and static program analysis [NNH05]. In principle, the resulting automaton can be easily modified to a service automaton.

Summing up, suitability of our model has been proved by a feature-complete open nets semantics for WS-BPEL and the choreography language BPEL4Chor. In addition, Petri net semantics for other service description languages, such as

BPMN, exist. In principle, it is possible to translate any service into an open net.

Part II.

Substitutability Criteria and Representations of Sets of Services

3. Substitutability Criteria

In this thesis, we consider two application scenarios of service substitutability in more detail: multiparty contracts (cf. Section 1.2.1) and service improvement (cf. Section 1.2.2). This chapter formally defines substitutability criteria that are applicable in these application scenarios.

In the literature, there exist various criteria when a service model can be substituted by another service model. These criteria depend on the (behavioral) properties that have to be preserved by the substitution. In this thesis, we consider five behavioral properties: deadlock freedom, weak termination, quasi-liveness, cover(Y), and strict termination (cf. Section 2.6.1). For any combination X of these properties, the set of X-strategies of N defines all open nets S such that the composition of N and S satisfies X.

We define the substitution of an open net N by an open net N' with respect to the set of X-strategies of N. We distinguish between two substitutability criteria: *X-conformance* and *X-preservation*. The criterion X-conformance ensures that N' preserves every X-strategy of N (Section 3.1), whereas the criterion X-preservation guarantees that N' preserves at least a certain subset of the X-strategies of N (Section 3.2).

Each substitutability criterion is defined as a preorder on open nets. In line with [Bau88, Vog92], these preorders are *external* preorders. An external preorder on N and on N' is not based on the respective internal characteristics of N and of N'. Rather, it relates these open nets with respect to their contexts (i. e., their X-strategies).

The X-conformance preorder is a classical refinement relation. This makes it worthwhile to identify the position of X-conformance in relation to known process equivalences and preorders. We show that {deadlock freedom}-conformance coincides with the stable failures preorder [Hoa85]. Furthermore, we show that fair testing [RV07] is the coarsest preorder known to us that implies {weak termination}-conformance.

The results of Section 3.1.1 and of Section 3.2 are a generalization of [ALM$^+$08, ALM$^+$09, SMB09]. The work on the relationship between {weak termination}-conformance and fair testing in Section 3.1.2 is based on [MSV09].

3.1. Substitutability under conformance

In the setting of multiparty contracts (cf. Section 1.2.1), each party of a contract has to implement its public view (the agreed specification of the service) such that the private view (the actual implementation of the public view) conforms to the overall specification of the contract.

In Section 3.1.1, we define conformance, a preorder on open nets, which is well-suited in the setting of contracts. In Section 3.1.2, we relate conformance to known preorders.

3.1.1. A notion of conformance

This section defines the notion of *conformance*. Conformance is a substitutability criterion that can be used to compare the public view and the private view on a party's share of a contract. The goal of the conformance notion is to preserve the set X of open-net properties of the overall contract N. For this reason, we use the term X-*conformance* in the following.

Let a contract N for parties $\mathcal{A} = \{A_1, \ldots, A_k\}$ be given, and let $N = N_1 \oplus \cdots \oplus N_k$ be an X-open net with an empty interface where N_i denotes the public view of a party A_i of $\mathcal{A}$, $i = 1, \ldots, k$. Each party A_i substitutes its public view N_i by the implemented private view N_i'. Let $N' = N_1' \oplus \cdots \oplus N_k'$ denote the overall contract as actually implemented. The notion of X-conformance should guarantee that if N is an X-open net and each private view N_i' X-*conforms* to its corresponding public view N_i, then N' is an X-open net as well.

Consider, for example, the weakly terminating contract for a credit request of Section 2.5. The contract is split into the three public views Client, Broker, and Credit. If each of these three public views is substituted by a private view such that the private view {weak termination}-conforms to the public view, then the composition of these private views should weakly terminate, too.

In general, each party's public view N_i is an X-strategy of the remaining part of the composition $N_1 \oplus \cdots \oplus N_{i-1} \oplus N_{i+1} \oplus \cdots \oplus N_k$. To guarantee that substituting the public view N_i by its private view N_i' does not affect the overall contract, we have to make sure that every X-strategy of N_i is also an X-strategy of N_i'.

Definition 3.1.1 ((X, X')-conformance).
Let N and N' be two open nets with equivalent interfaces, and let $X'' \subseteq \mathit{Prop} \setminus \{\mathrm{cover}(Y)\}$ be a set of open-net properties. Let either $X = X''$ and $X' = X''$ or $X = X'' \cup \mathrm{cover}(Y)$ and $X' = X'' \cup \mathrm{cover}(Y')$ with $Y \subseteq P_N \cup T_N$ and $Y' \subseteq P_{N'} \cup T_{N'}$. Open net N' *substitutes* N *under* (X, X')-*conformance* (N' (X, X')-*conforms to* N for short), denoted by $N' \sqsubseteq_{\mathit{conf},(X,X')} N$, iff $\mathit{Strat}_{X'}(N') \supseteq \mathit{Strat}_X(N)$. If $X = X'$ we write X instead of (X, X'). ⌟

Note that we need two parameters X and X' only in case open-net nodes have to be covered. For the sake of simplicity we will write X-conformance rather than (X, X')-conformance in the following and only give the precise notation if necessary. The reason is (as we will show further down) that we need a symmetric X-strategy notion in the setting of contracts.

X-conformance defines a refinement relation on open nets: If an open net N' X-conforms to an open net N, then *every* X-strategy of N is an X-strategy of N'

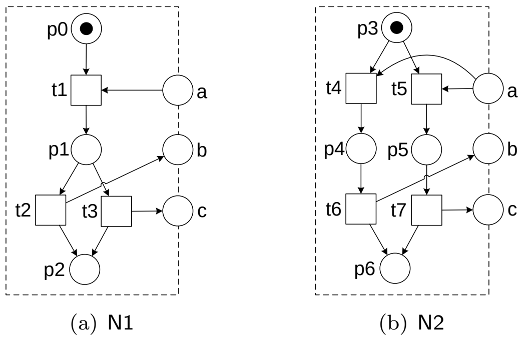


Figure 3.1.: Two open nets with $\mathit{Strat}_X(\mathsf{N1}) = \mathit{Strat}_X(\mathsf{N2})$, for $X = \{\text{weak termination}\}$.

as well. In addition, X-conformance allows N' to have more X-strategies. Therefore, X-conformance is well-suited for implementing public views of a multiparty contract.

Definition 3.1.1 defines X-conformance for arbitrary interface equivalent open nets N and N'. If N is a public view and N' is the corresponding private view, then interface equivalence is trivially fulfilled.

The notion of {deadlock freedom}-conformance has been introduced in [SMB09], and the notion of {weak termination}-conformance has been introduced in [ALM$^+$08, ALM$^+$09]. The latter notion is actually equivalent to the *conflict relation* in [MSR06] and the *subcontract relation* in [BZ07a].

As an illustration of the X-conformance relation, consider the two open nets in Figure 3.1 and assume $\Omega_{\mathsf{N1}} = \{[\mathsf{p2}]\}$ and $\Omega_{\mathsf{N2}} = \{[\mathsf{p6}]\}$. In this case, any open net that weakly terminates with N1 also weakly terminates with N2, and any open net that weakly terminates with N2 also weakly terminates with N1. In other words, N1 {weak termination}-conforms to N2 and vice versa. Both open nets are not branching bisimilar (cf. Definition 2.1.5), and hence branching bisimulation seems to be finer than X-conformance.

In the following, we consider some properties of the X-conformance relation. As X-conformance is defined as a subset relation on sets of X-strategies, it is easy to see that X-conformance is a *preorder*; that is, it is a reflexive and transitive relation.

Lemma 3.1.2 ((X, X')-conformance is a preorder).
The (X, X')-conformance relation, $\sqsubseteq_{\mathit{conf},(X,X')}$, is a preorder. ⌟

Besides the preorder property it is also important that the X-conformance relation between two open nets is preserved under composition; thus, if N' X-conforms to N, we want that $(N' \oplus R)$ X-conforms to $(N \oplus R)$, for any interface compatible open net R. A preorder that satisfies this condition is a *precongruence* with respect to the composition operator $\oplus$.

Lemma 3.1.3 ((X, X')-conformance is a precongruence w.r.t. $\oplus$).
The (X, X')-conformance relation is a precongruence with respect to the composition operator $\oplus$. ⌟

Proof.
Let N, N' be two open nets, and let X, X' be two sets of open-net properties defined as in Definition 3.1.1 such that $N' \sqsubseteq_{\mathit{conf},(X,X')} N$. So by Definition 3.1.1, we have $\mathit{Strat}_{X'}(N') \supseteq \mathit{Strat}_X(N)$. Let R be any interface compatible open net for N and N'. We have to show that $(N' \oplus R) \sqsubseteq_{\mathit{conf},(X,X')} (N \oplus R)$.

Let $S \in \mathit{Strat}_X(N \oplus R)$; that is, $N \oplus R \oplus S$ is an X-open net with empty interface. Hence, $(R \oplus S) \in \mathit{Strat}_X(N)$, because composition operator $\oplus$ is by Lemma 2.4.3 commutative and associative. Because of $N' \sqsubseteq_{\mathit{conf},(X,X')} N$ we have $(R \oplus S) \in \mathit{Strat}_{X'}(N')$, and hence $N' \oplus R \oplus S$ is an X'-open net with empty interface as well. By associativity of $\oplus$, we have $S \in \mathit{Strat}_{X'}(N' \oplus R)$, and hence we conclude $(N' \oplus R) \sqsubseteq_{\mathit{conf},(X,X')} (N \oplus R)$.

Note that if $\mathit{Strat}_X(N \oplus R) = \emptyset$; that is, $N \oplus R$ is not X-controllable, then $(N' \oplus R) \sqsubseteq_{\mathit{conf},(X,X')} (N \oplus R)$ holds trivially. □

The precongruence result enables us to refine an open-net composition in a modular way; that is, it guarantees compositionality. This is, in fact, crucial in the setting of multiparty contracts. As a direct consequence of this precongruence property, the following theorem shows that *each* party of a contract can substitute its public view by a private view *independently*. If each of the private views X-conforms to the corresponding public view, then the set X of open-net properties of the implemented contract is preserved.

Theorem 3.1.4 (implementation of a contract).
Let $X \subseteq \mathit{Prop} \setminus \{\mathrm{cover}(Y)\}$ be a set of open-net properties, and let N be a contract for parties $\{A_1, \ldots, A_k\}$ where N is an X-open net. If, for all $i = 1, \ldots, k$, N'_i X-conforms to N_i, then $N' = N'_1 \oplus \cdots \oplus N'_k$ is an X-open net. ⌟

In Theorem 3.1.4 we restricted ourselves to a set of open-net properties X that does not contain $\mathrm{cover}(Y)$; otherwise, we could not substitute a public view N_i by a private view N'_i if N_i contains nodes of N that have to be covered according to X. The reason is that the open-net property X of the contract N is global and hence requires a symmetric X-strategy notion.

The value of Theorem 3.1.4 is that it gives each party A_i of a contract a criterion—that is, X-conformance of the private view N'_i to its public view N_i—that can be locally verified for asserting a global property (i. e., the implemented contract N' is still an X-open net).

Restricting X-conformance to the same set of X-strategies yields an another substitutability criterion, X-conformance equivalence.

$$\begin{tikzcd}
 & & & \mathsf{S} \arrow[rrd] & & \\
 & & \mathsf{RS} \arrow[ru] \arrow[r] & \mathsf{F} \arrow[r] & \mathsf{CT} \arrow[r] & \mathsf{T} \\
\mathsf{B} \arrow[r] & \mathsf{BB} \arrow[ru] \arrow[r] & \mathsf{PF} \arrow[r] & \mathsf{FT} \arrow[ru] & &
\end{tikzcd}$$

Figure 3.2.: Some known preorders from the linear time – branching time spectrum.

Definition 3.1.5 ((X, X')-conformance equivalence).
Let N and N' be interface equivalent open nets, and let X, X' be sets of open-net properties as defined in Definition 3.1.1. Open net N' is (X, X')-*conformance equivalent* to N, denoted by $N' \equiv_{\mathit{conf},(X,X')} N$, iff $\mathit{Strat}_{X'}(N') = \mathit{Strat}_X(N)$. If $X = X'$ we write X instead of (X, X'). ⌟

X-conformance equivalence is defined as the identity on sets of X-strategies. Hence, it is an equivalence relation; that is, the relation $\equiv_{\mathit{conf},X}$ is reflexive, transitive, and symmetric.

Lemma 3.1.6 ((X, X')-conformance equivalence is an equivalence).
(X, X')-conformance equivalence, $\equiv_{\mathit{conf},(X,X')}$, is an equivalence relation. ⌟

It is easy to see that we can adapt Lemma 3.1.3 to X-conformance equivalence. This result enables us to prove a variant of Theorem 3.1.4 in case of X-conformance equivalence. In other words, X-conformance equivalence can also be used as a substitutability criterion in the setting of multiparty contracts.

3.1.2. Relationship between conformance and known preorders

X-conformance is a classical refinement relation and a precongruence with respect to composition. The linear time – branching time spectrum [Gla93, Gla01] lists many known preorders. Hence, we are interested in the relationship between X-conformance and these preorders.

In Figure 3.2, some of these preorders and the relations between them are depicted, featuring B (bisimulation), BB (branching bisimulation), S (simulation), RS (ready simulation), PF (possible futures), FT (fair testing [BRV95, NC95, RV07]), F (stable failures [BHR84, Hoa85, Ros98]), CT (completed trace), and T (trace) preorders. An arrow between two preorders denotes the inclusion relation; for example, B implies (is finer than) BB; if an arrow is absent (in the transitive closure), then the inclusion does not hold.

In case of $X = \{\text{deadlock freedom}\}$, we can prove that X-conformance and the stable failures preorder, denoted by $\sqsubseteq_f$, coincide.

Theorem 3.1.7 (X-conformance coincides with stable failures).
Let $X = \{\text{deadlock freedom}\}$. For any two interface equivalent open nets N and N' holds:

$$N' \sqsubseteq_{\mathit{conf},X} N \Leftrightarrow N' \sqsubseteq_{\mathit{f}} N \quad . \qquad \lrcorner$$

For a formal definition of the stable failures preorder and for a proof of this theorem, we refer the interested reader to Appendix A.2.

In case of $X = \{\text{weak termination}\}$, [MSR06, BZ07a] prove that fair testing, denoted by $\sqsubseteq_{\mathit{ft}}$, implies X-conformance (called conflict preorder in [MSR06] and subcontract preorder in [BZ07a]), but X-conformance does not imply fair testing.

Theorem 3.1.8 (X-conformance implies fair testing).
Let $X = \{\text{weak termination}\}$. For any two interface equivalent open nets N and N' holds:

$$N' \sqsubseteq_{\mathit{ft}} N \Rightarrow N' \sqsubseteq_{\mathit{conf},X} N \quad . \qquad \lrcorner$$

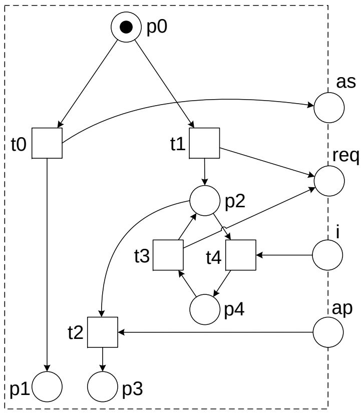
In Appendix A.1, we show that fair testing is the coarsest preorder known to us that implies X-conformance. Moreover, we analyze under which conditions X-conformance would imply fair testing. In this way, we identify the real differences between X-conformance and fair testing.

3.2. Substitutability under preservation

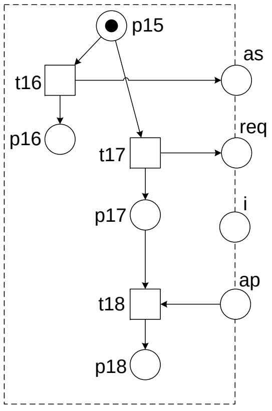
This section introduces a substitutability criterion that is suitable in the context of service improvement. In the setting of service improvement (cf. Section 1.2.2), an open net N shall be substituted by an improved version N' of N. Clearly, we could require that N' X-conforms to N. However, in the scenario of service improvement, X-conformance might be too restrictive; for example, if we want to remove some unprofitable functionality of N. Clearly, this would violate X-conformance.

To this end, we introduce with *X-preservation* another substitutability criterion. This criterion aims at preserving at least a fixed *subset* $\mathcal{S}$ of the X-strategies of N. Applications for X-preservation include: an upgraded service supports only behavior that is used by major clients, and all other clients have to adjust their services; an enterprise restricts itself to its core competencies, and thus all unprofitable X-strategies are rejected from its service; the behavior of a service is restricted to certain scenarios, such as payment by credit card. These considerations lead to the following definition of X-preservation.

Definition 3.2.1 ((X, X')-preservation).
Let N and N' be two open nets with equivalent interfaces, and let $X'' \subseteq \mathit{Prop} \setminus \{\text{cover}(Y)\}$ be a set of open-net properties. Let either $X = X''$ and $X' = X''$ or $X = X'' \cup \text{cover}(Y)$ and $X' = X'' \cup \text{cover}(Y')$ with $Y \subseteq P_N \cup T_N$ and $Y' \subseteq P_{N'} \cup T_{N'}$, and let $\mathcal{S} \subseteq \mathit{Strat}_X(N)$. Open net N' *substitutes* N *under* (X, X')-*preservation of* $\mathcal{S}$ (N' (X, X')-*preserves* $\mathcal{S}$ for short), denoted by $N' \equiv_{\mathit{pres},\mathcal{S},(X,X')} N$, iff $\mathit{Strat}_{X'}(N') \supseteq \mathcal{S}$. If $X = X'$ we write X instead of (X, X'). $\lrcorner$



(a) Bank



(b) Bank″

Figure 3.3.: The online bank Bank and an improved version Bank″.

Like for X-conformance, we will use the notion X-preservation the following and only give the precise notion if necessary.

In [SMB09], X-preservation has been introduced as *restriction* in case of $X = \{\text{deadlock freedom}\}$. According to Definition 3.2.1, at least every open net in $\mathcal{S}$ is an X-strategy of N', meaning, the substitution preserves at least the X-strategies in $\mathcal{S}$. Hence, X-preservation seems to be the right notion to achieve the previously mentioned goal.

Consider the running example Bank in Figure 3.3(a). Suppose we want to substitute Bank by an "improved" banking service that preserves all customers for Bank that do not send information messages. Then, Bank″ in Figure 3.3(b) with $\Omega_{\mathsf{Bank}''} = \{[\mathsf{p18}]\}$ is a valid candidate. We will come back to this in Chapter 6.2.

X-conformance requires that N' preserves all X-strategies of N, whereas X-preservation requires that N' preserves only a subset of N's X-strategies. Hence, there are fewer services N' that X-conform to N than services N' that satisfy X-preservation. Clearly, X-preservation coincides with X-conformance for $\mathcal{S} = \mathit{Strat}_X(N)$.

The X-preservation relation is an equivalence relation.

Lemma 3.2.2 ((X, X')-preservation is an equivalence).
The (X, X')-preservation relation, $\equiv_{\mathit{pres},\mathcal{S},(X,X')}$, is an equivalence relation. ⌟

Proof.
Let N_1, N_2, and N_3 be interface equivalent open nets, and let X_1, X_2, X_3 be any sets of open-net properties as defined in Definition 3.2.1.

Relation $\equiv_{\mathit{pres},\mathcal{S},X}$ is reflexive: For any $\mathcal{S} \subseteq \mathit{Strat}_{X_1}(N_1)$, we have $N_1 \equiv_{\mathit{pres},\mathcal{S},X_1} N_1$.

Relation $\equiv_{\boldsymbol{pres},\mathcal{S},X}$ is transitive: Suppose $N_3 \equiv_{\boldsymbol{pres},\mathcal{S},(X_2,X_3)} N_2$ and $N_2 \equiv_{\boldsymbol{pres},\mathcal{S},(X_1,X_2)} N_1$. By Definition 3.2.1, $N_2 \equiv_{\boldsymbol{pres},\mathcal{S},(X_1,X_2)} N_1$ implies $\mathit{Strat}_{X_2}(N_2) \supseteq \mathcal{S}$. Hence, because $N_3 \equiv_{\boldsymbol{pres},\mathcal{S}(X_2,X_3)} N_2$, also $\mathit{Strat}_{X_3}(N_3) \supseteq \mathcal{S}$. Hence, we conclude $N_3 \equiv_{\boldsymbol{pres},\mathcal{S},(X_1,X_3)} N_1$.

Finally, relation $\equiv_{\boldsymbol{pres},\mathcal{S},X}$ is symmetric: Let $N_2 \equiv_{\boldsymbol{pres},\mathcal{S},(X_1,X_2)} N_1$. This implies by Definition 3.2.1, $\mathcal{S} \subseteq \mathit{Strat}_{X_1}(N_1)$ and $\mathcal{S} \subseteq \mathit{Strat}_{X_2}(N_2)$. Thus, $N_1 \equiv_{\boldsymbol{pres},\mathcal{S},(X_2,X_1)} N_2$. $\square$

In contrast to the X-conformance relation, there is no meaningful congruence property with respect to composition for preservation. Let N and N' be two interface equivalent open nets such that $N' \equiv_{\boldsymbol{pres},\mathcal{S},X} N$. Let R be any interface compatible open net for N and for N'. Then $(N' \oplus R) \equiv_{\boldsymbol{pres},\mathcal{S},X} (N \oplus R)$ typically does not hold. The reason is that none of the open nets $S \in \mathcal{S}$ is an X-strategy of $N' \oplus R$ and of $N \oplus R$, because composing N and R as well as composing N' and R changes the interface of N and N', respectively. As a consequence, X-preservation misses an important property. Hence, from a theoretical point of view, X-preservation is less interesting than X-conformance. However, from a practical point view the X-preservation relation is very interesting in the setting of service substitutability.

4. A Finite Representation of Strategies

To decide whether an open net N' can substitute another open net N, we have to compare the two in general *infinite sets* of X-strategies of N' and of N. Therefore, we aim at constructing a *finite representation* of these sets that can be used to decide substitutability; see also Figure 1.4.

Given an open net N and a set X of open-net properties, we present an algorithm to *construct* a finite representation of the set $\mathit{Strat}_X(N)$ of all X-strategies of N. We also provide a method to check *containment* of an open net in such a representation.

For each set $\mathit{Strat}_X(N)$, there exists a particular service automaton $\mathit{MP}_X(N)$ that simulates the corresponding service automaton $\mathit{SA}(S)$ of every X-strategy S of N. Due to this property, $\mathit{MP}_X(N)$ is called the *most permissive X-strategy of N*. As a simulation relation of $\mathit{SA}(S)$ by $\mathit{MP}_X(N)$ is a necessary condition for S being an X-strategy of N, the representation of $\mathit{Strat}_X(N)$ is based on the most permissive X-strategy of N. To exclude open nets that are not an X-strategy of N, we have to add additional restrictions to $\mathit{MP}_X(N)$.

It turns out that the choice of the set X of open-net properties has a great impact on the set $\mathit{Strat}_X(N)$ of all X-strategies of N and therewith also a great impact on the restrictions we have to add to $\mathit{MP}_X(N)$. In this thesis, we focus on the 9 combinations of deadlock freedom, weak termination, quasi-liveness, cover(Y), and strict termination listed in Table 2.1. To illustrate the impact of X on $\mathit{Strat}_X(N)$, consider the open net Bank in Figure 2.3 and the three customers in Figure 2.4. Customers Cust1 and Cust3 are {deadlock freedom}-strategies of Bank whereas Cust2 is not. If we, however, consider {weak termination} as the set of open-net properties, only Cust3 is a {weak termination}-strategy of Bank. The corresponding representation of all X-strategies of Bank has to take this into account.

Interestingly, only 6 of the 9 sets of open-net properties listed in Table 2.1 affect the algorithmic solution of a finite representation, namely

- $X_1 = \{\text{deadlock freedom}\}$;
- $X_2 = X_1 \cup \{\text{quasi-liveness}\}$;
- $X_2(Y) = X_1 \cup \{\text{cover}(Y)\}$;
- $X_3 = \{\text{weak termination}\}$;
- $X_4 = X_3 \cup \{\text{quasi-liveness}\}$;

- $X_4(Y) = X_3 \cup \{\text{cover}(Y)\}$.

Unlike deadlock freedom, weak termination, quasi-liveness and cover(Y), strict termination can always be incorporated into the respective algorithmic solution and will therefore be neglected. As illustrated by the notations X_2 and $X_2(Y)$ as well as X_4 and $X_4(Y)$, quasi-liveness is a special case of cover(Y).

In each of the following four Sections 4.1–4.4 we present a finite representation, an *X-operating guideline*, for one set of open-net properties $X \in \{X_1, X_2(Y), X_3, X_4(Y)\}$. In Section 4.5, we show how the representations for X_3 and $X_4(Y)$ have to be adjusted if, in addition, strict termination is added to these sets of open-net properties.

In this chapter, we frequently compose an open net N and a service automaton A, where $N \oplus PN(A)$ is a closed net. To ease the representation, we do not explicitly transform the service automaton A into an open net $PN(A)$, but use $N \oplus A$ as a *shorthand notation* for the composition $N \oplus PN(A)$. Each state (m, q) in $N \oplus A$ consists of a marking m of N and a state q of A. State (m, q) is defined as a marking m^* of the composition $N \oplus PN(A)$. Marking $m = m^*|_{P_N}$ is the projection of m^* to the places of N, and state q corresponds to a marking $m^*|_{P_{inner(PN(A))}}$, which is the projection of m^* to the places of the inner subnet of $PN(A)$.

4.1. Representing strategies in case of deadlock freedom

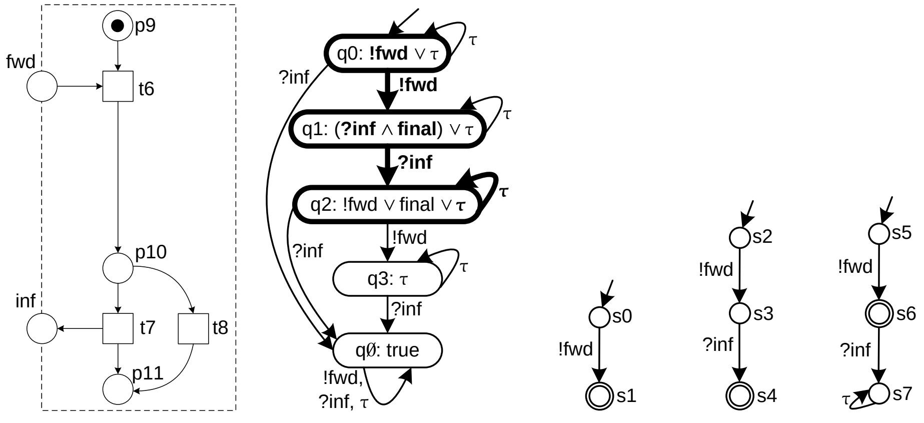
In this section, we recapitulate *operating guidelines* [MS05, LMW07b, Mas09]. An operating guideline of an open net N[1] is a finite representation of the (in general) infinite set of X_1-strategies of N. Technically, an X_1-operating guideline is an *annotated automaton*; that is, an automaton where each state is annotated with a Boolean formula.

We introduce annotated automata in Section 4.1.1. Subsequently, we define the construction of an X_1-operating guideline for an open net. For that purpose, we present an algorithm to construct the automaton (Section 4.1.2) and the annotation (Section 4.1.3) of an X_1-operating guideline.

4.1.1. Annotated automata to represent sets of open nets

This section introduces annotated automata as a compact representation of a set of open nets. Before we present the formal definitions, we introduce the general concepts of annotated automata with the help of the annotated automaton A^{ϕ} in Figure 4.1(b).

[1]We use the term X_1-operating guideline in this thesis.



(a) Credit (b) A^{ϕ} (c) SA_1 (d) SA_2 (e) SA_3

Figure 4.1.: The annotated automaton A^{ϕ} is the X_1-operating guideline of Credit. A^{ϕ} represents SA_3 (depicted in bold), but not SA_1 and SA_2 (assuming that the corresponding open net of each service automaton is a partner of Credit).

An illustrating example

The annotated automaton A^{ϕ} consists of an automaton A with five states. Each state is annotated with a Boolean formula, depicted inside of the state. As A^{ϕ} represents a set of open nets, we need a procedure to check whether an open net is represented by A^{ϕ}. We call this procedure *matching* and define $\mathit{Match}(\mathsf{A}^{\phi})$ as the set of all open nets that are represented by A^{ϕ}. Annotated automaton A^{ϕ} represents all X_1-strategies of Credit; hence, it is the X_1-operating guideline of Credit.

If an open net S is represented by A^{ϕ} (i. e., S *matches* with A^{ϕ}), then A simulates the corresponding service automaton $SA(S)$ of S; for example, A simulates the three service automata SA_1, SA_2, and SA_3 in Figure 4.1. The simulation relations are $\{(\mathsf{s0},\mathsf{q0}),(\mathsf{s1},\mathsf{q1})\}$, $\{(\mathsf{s2},\mathsf{q0}),(\mathsf{s3},\mathsf{q1}),(\mathsf{s4},\mathsf{q2})\}$, and $\{(\mathsf{s5},\mathsf{q0}),(\mathsf{s6},\mathsf{q1}),(\mathsf{s7},\mathsf{q2})\}$. Simulation is, however, only a sufficient criterion for matching; that is, A simulates service automata whose corresponding open nets are not represented by A^{ϕ}; for example, although the corresponding open nets of SA_1 nor SA_2 are no X_1-strategies of Credit, SA_1 and SA_2 are simulated by A. In the composition with SA_1, Credit can send message inf, which cannot be received by SA_1; in the composition with SA_2, Credit can terminate without sending message inf yielding a deadlock in the composition.

To exclude such open nets, we need to specify which restrictions of the structure of A correspond to behavior of open nets S that are contained in $\mathit{Match}(\mathsf{A}^{\phi})$. This can be achieved by specifying, for each state q of A, which outgoing transitions have to be present in a state of $SA(S)$ that is related to q by the simulation relation and whether the related state has to be a final state. To this end, every state q of A is annotated with a Boolean formula $\phi(q)$; for example, the annotation $\phi(\mathsf{q0})$ requires an open net either to send message fwd or to do a τ-step, and the annotation $\phi(\mathsf{q1})$ requires an open net either to be able to receive message inf *and* to be in a final state or to do a τ-step.

Observe that the due to the annotations the corresponding open net of SA_1 does not match with A^{ϕ}: In related states $(\mathsf{s1}, \mathsf{q1})$, the state $\mathsf{s1}$ can neither receive message inf nor execute an internal transition. Furthermore, the corresponding open net of SA_2 does also not match with A^{ϕ}, because in related states $(\mathsf{s3}, \mathsf{q1})$, the state $\mathsf{s3}$ is neither a final state nor does it enable an internal transition.

The service automaton SA_3 meets the restrictions of the annotations; for example, in related states $(\mathsf{s5}, \mathsf{q0})$, it can send message fwd, and in related states $(\mathsf{s6}, \mathsf{q1})$, the state $\mathsf{s6}$ is a final state and the service automaton can receive message inf. This is illustrated in bold in Figure 4.1(b). Hence, the corresponding open net of SA_3 matches with A^{ϕ}, and it is easy to see that its composition with Credit is deadlock-free.

Boolean formulae

As a Boolean formula is assigned to every state of an annotated automaton, we introduce Boolean formulae in the following. The definition will be very general, because annotated automata are not the only application of Boolean formulae in this thesis.

To define Boolean formulae, we need a propositional logic *without* negation. A *literal* of a Boolean formula is an element of an alphabet Σ. Let $\mathcal{BF}^{\Sigma}$ be the set of all such Boolean formulae with literals taken from Σ.

Definition 4.1.1 (Boolean formula).
The set $\mathcal{BF}^{\Sigma}$ of *Boolean formulae* with literals taken from an alphabet Σ is inductively defined as follows:

Basis : $x \in \Sigma$, *true*, *false* are Boolean formulae.
Step : If ψ, χ are Boolean formulae, then $(\psi \vee \chi)$ and $(\psi \wedge \chi)$ are Boolean formulae. ⌟

An *assignment* β is a mapping $\beta : \Sigma \to \{\mathit{true}, \mathit{false}\}$ assigning to each literal a truth value. An assignment β *satisfies* a Boolean formula ψ, denoted by $\beta \models \psi$, if ψ evaluates to *true* under β using standard propositional logic semantics. We define $\beta \models \mathit{true}$ and $\beta \not\models \mathit{false}$, for all β.

As usual, we denote the implication of two Boolean formulae ψ and χ (which is not a Boolean formula) by $\psi \Rightarrow \chi$; equivalence of ψ and χ is denoted by $\psi \equiv \chi$. If $\psi \equiv \mathit{true}$, then ψ is a tautology; if $\psi \equiv \mathit{false}$, then ψ is a contradiction.

Representing a set of open nets

To define annotated automata, we extend automata (cf. Definition 2.7.5) by an annotation function that assigns a Boolean formula, as defined in Definition 4.1.1, to each state of the automaton. In contrast to service automata, annotated automata do not have final states. The reason is that an annotated automaton describes many open nets, and each open net can have different final markings. Hence, an annotated automaton does not have final states, but information about final states is encoded in the annotations instead.

With $\mathcal{MC}^+$, we denote the set $\mathcal{MC} \cup \{\mathit{final}, \tau\}$ of message channels, extended by the literals τ and *final* representing an internal transition and a final state, respectively. For the Boolean formulae of annotated automata, the set $\mathcal{MC}^+$ serves as the alphabet Σ.

Furthermore, as X_1-operating guidelines will always have a deterministic transition relation, we restrict annotated automata to have a deterministic transition relation, too.

Definition 4.1.2 (annotated automaton).
An *annotated automaton* $A^\phi = (Q, MC^I, MC^O, \delta, q_0, \phi)$ consists of a deterministic automaton $A = (Q, MC^I, MC^O, \delta, q_0)$ and an *annotation function* $\phi : Q \to \mathcal{BF}^{\mathcal{MC}^+}$. ⌟

The idea of using annotated automata as a representation of a set of automata has been published first by Wombacher et al. [WFMN04]. As the main difference to our work, an annotated automaton in [WFMN04] represents the behavior of a set of synchronously communicating automata, whereas an annotated automaton in our case represents a set of asynchronously communicating open nets.

To represent a set of open nets, we take an annotated automaton A^ϕ and define when an open net S, described in terms of its corresponding service automaton $SA(S)$, matches with A^ϕ. Open net S matches with A^ϕ if

1. there exists a minimal simulation relation ϱ of $SA(S)$ by A; and
2. for every state q_S of $SA(S)$, such that $(q_S, q_A) \in \varrho$, the transitions leaving q_S and the fact whether q_S is a final state of $SA(S)$ constitute a satisfying assignment for $\phi(q_A)$.

Note that we require the existence of a *minimal* simulation relation of $SA(S)$ by A (see Definition 2.1.4). Because annotated automata are by definition deterministic, we can be sure that if there exists any simulation relation of $SA(S)$ by A, then there exists also a minimal one. We choose the minimal simulation relation, because it is well-defined and can be easily calculated. Furthermore, it guarantees that only states of $SA(S)$ that are reachable from the initial state of $SA(S)$ are related to states of A.

Definition 4.1.3 (service automaton constitutes assignment).
A service automaton $A = (Q, MC^I, MC^O, \delta, q_0, \Omega)$ *constitutes to each state* $q \in Q$ *an assignment* $\beta_A(q) : \mathcal{MC}^+ \to \{true, false\}$, defined by

$$\beta_A(q)(x) = \begin{cases} \mathit{true}, & \text{if } x \in \mathcal{MC}^+ \setminus \{\mathit{final}\} \text{ and there is a state } q' \text{ with} \\ & (q, x, q') \in \delta; \\ \mathit{true}, & \text{if } x = \mathit{final} \text{ and } q \in \Omega; \\ \mathit{false}, & \text{otherwise.} \end{cases}$$
⌟

The assignment $\beta_A(q)$ assigns *true* to a literal x if there is an x-labeled transition that leaves q or if $x = \mathit{final}$ and q is a final state of A. To all other literals *false* is assigned.

With the help of the assignment β, matching of an open net with an annotated automaton can be defined as follows.

Definition 4.1.4 (matching with an annotated automaton).
Let S be an open net, and let $A^\phi = (Q, MC^I, MC^O, \delta, q_0, \phi)$ be an annotated automaton such that $MC^I = MC^I_{SA(S)}$ and $MC^O = MC^O_{SA(S)}$. Open net S *matches with* A^ϕ iff

- there exists a minimal simulation relation ϱ of $SA(S)$ by A, where
- for each $(q_S, q_A) \in \varrho : \ \beta_{SA(S)}(q_S) \models \phi(q_A)$.

Let $Match(A^\phi)$ denote the set of all open nets that match with A^ϕ. ⌟

Consider again the annotated automaton A^ϕ in Figure 4.1(b). Its initial state is $\mathsf{q0}$, and the alphabet is given by $MC^I = \{\mathsf{?inf}\}$ and by $MC^O = \{\mathsf{!fwd}\}$. The service automaton SA_2 in Figure 4.1(d) is simulated by A^ϕ; however, state $\mathsf{s3}$ does not satisfy the formula $\phi(\mathsf{q1})$. SA_2 constitutes assignment $\beta_{SA_2}(\mathsf{s3})(\mathsf{?inf}) = \mathit{true}$ and *false* for all other literals. Hence, the corresponding open net of SA_2 does not match with A^ϕ. In contrast, the service automaton SA_3 is simulated by A^ϕ, and, in addition, each state of SA_3 satisfies the annotation ϕ of the respective state in A^ϕ. Hence, the corresponding open net of SA_3 matches with A^ϕ.

In spite of the restriction to deterministic structures and negation-free Boolean formulae in Definition 4.1.2, an annotated automaton is able to represent an open net whose corresponding service automaton contains τ-transitions. To this end, a disjunct τ has to be added to each Boolean formula, and each state of the annotated automaton must contain a τ-labeled self-loop. As an illustration, consider SA_3 and A^ϕ in Figure 4.1, with $(\mathsf{s7}, \mathsf{q2}) \in \varrho$. The state $\mathsf{s7}$ would violate the minimal simulation relation if the τ-loop was not present in the state $\mathsf{q2}$, and it would violate $\phi(\mathsf{q2})$ if the τ-disjunct was not present in $\phi(\mathsf{q2})$.

A normal form for annotated automata

The definition of an annotated automaton permits annotated automata that contain states or literals in the annotation of a state, which are not needed for the matching with any open net. This is a problem, because those states and annotations unnecessarily blow up the size of an annotated automaton and complicate the comparison whether two annotated automata match the same set of open nets. To approach this problem, Massuthe [Mas09] introduces a *normal form* for annotated automata. The normal form $normal(A^\phi)$ of any annotated automaton A^ϕ preserves the semantics of A^ϕ—that is, $Match(A^\phi) = Match(normal(A^\phi))$. In the following, we sketch the idea of this normalization.

Let B be the service automaton with the structure of A and the empty set of final states. An *annotation* $\phi(q)$ of a state q of A^ϕ is in normal form if, for each literal $x \in \phi(q)$, with $x \in \mathcal{MC} \cup \{\tau\}$, the service automaton B constitutes an assignment $\beta_B(q)(x) = true$. If $\beta_B(q)(x) = false$, then no service automaton of any open net that matches with A^ϕ can assign *true* to x. The reason is that the assignment β_B depends on the minimal simulation of B by A, and no service automaton can relate more states of A than B. Each such literal x is replaced by the formula *false* yielding the normalized annotation. As literal *final* can be set to *true* independently of B, it does not have to be considered. A *state* q of A^ϕ is in normal form if its annotation can be satisfied—that is, $\phi(q) \neq false$.

To normalize an annotated automaton A^ϕ, we first normalize the annotation of each state. In a second step, all states that are not in normal form are removed (together with their adjacent transitions). As the removal of states may again yield annotations that are not in normal form, this procedure has to be repeated until a fixpoint is reached. The resulting annotated automaton is then in normal form.

A slightly modified version $\mathsf{A'}^{\phi'}$ of the annotated automaton A^ϕ in Figure 4.1(b) is illustrated in Figure 4.2. Normalizing $\mathsf{A'}^{\phi'}$ yields in the first iteration $normal(\phi'(\mathsf{q4})) = false$, because there is no ?inf-labeled transition leaving q4 and hence $\beta_{B'}(\mathsf{q4})(\mathsf{?inf}) = false$. As a consequence, q4 is not in normal form, and hence it is removed. In the second iteration, the literal !fwd in the state q3 is replaced by the Boolean formula *false*. Hence, $normal(\phi'(\mathsf{q3})) = final \vee \tau$. The normalized annotated automaton is A^ϕ in Figure 4.1(b).

The following theorem justifies that the normalization procedure does not change the semantics of an annotated automaton. For a proof, we refer to [Mas09, Theorem 4.3.12, p.119].

Theorem 4.1.5 (expressiveness of normal annotated automata).
For each annotated automaton A^ϕ exists a normal form $A'^{\phi'} = normal(A^\phi)$ such that $Match(A^\phi) = Match(A'^{\phi'})$. ⌟

The next lemma proves that, for every nonempty, normalized annotated automaton A^ϕ, the service automaton A' that has the automaton of A and contains final states according to the annotation of A^ϕ matches with A^ϕ.

Lemma 4.1.6 (most permissive service automaton).
Let $A^\phi = (Q, MC^I, MC^O, \delta, q_0, \phi)$ be annotated automaton in normal form with $Q \neq \emptyset$, and let $A' = (Q, MC^I, MC^O, \delta, q_0, \Omega)$ be a service automaton, with $\Omega =$

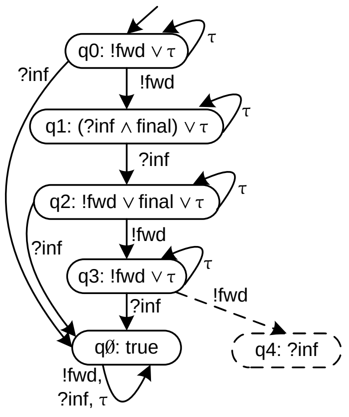


Figure 4.2.: An annotated automaton $\mathsf{A}'^{\phi'}$ that is not in normal form.

$\{q \in Q \mid \mathit{final}$ occurs in $\phi(q)\}$. Then, $PN(A')$ matches with A^{ϕ}. ⌟

Proof.
By construction of A', there is a minimal simulation relation of A' by A^{ϕ}. For each state $q \in Q$, $\beta_{A'}(q)$ assigns *true* to each literal in q (because A^{ϕ} is normalized by assumption). Therefore, $\beta_{A'}(q)$ satisfies $\phi(q)$. Hence, $PN(A')$ matches with A^{ϕ}. □

In the next subsection, we use annotated automata to represent all X_1-strategies of an open net.

4.1.2. Construction of the most permissive X_1-strategy

In this section, we present an algorithm to construct the most permissive X_1-strategy $MP_{X_1}(N)$ of N. More precisely, we do *not* construct an open net, but its corresponding service automaton. $MP_{X_1}(N)$ has an important property: It simulates the corresponding service automaton of every X_1-strategy of N. Moreover, every marking that N can reach in the composition with any X_1-strategy of N is reachable in the composition of N and $MP_{X_1}(N)$ as well. The underlying automaton of $MP_{X_1}(N)$ is an automaton A providing the structure of the operating guideline A^{ϕ} of N.

An illustrating example

The construction of the most permissive X_1-strategy $MP_{X_1}(\mathsf{Bank})$ of the open net Bank in Figure 4.3(a) is illustrated in Figure 4.3(b). The idea is to assign to each state q of $MP_{X_1}(N)$ the set of all markings that N might be in while $MP_{X_1}(N)$ is in q. The outgoing transitions of q are maximal in the sense that,

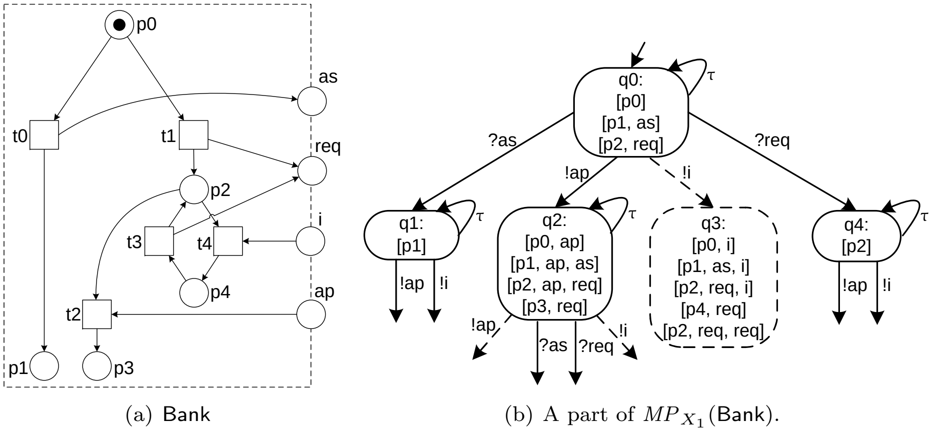


Figure 4.3.: Construction of the most permissive X_1-strategy MP_{X_1}(Bank) of Bank. The state q3 violates the message bound $b = 1$ and will be removed.

for each message that can be sent or received in one of the markings assigned to q, a respective receiving and sending transition with source q is added to $MP_{X_1}(N)$.

To construct MP_{X_1}(Bank), we add initially the initial state q0. If MP_{X_1}(Bank) is in state q0, then Bank is in its initial marking [p0] or in markings [p1, as] or [p2, req] that are reachable from [p0] (without any interaction with MP_{X_1}(Bank)). These three markings are assigned to the state q0; graphically, they are depicted inside this state.

Next, MP_{X_1}(Bank) has to trigger Bank. It can execute arbitrary many τ-transition; hence, we add a τ-loop to q0. It can also send any message of the input alphabet (i. e., ap and i); hence, we add states q2 and q3 and their respective transitions. For example, sending i results in adding to each marking in q0 a token on place i. This yields the first three markings in state q3. From marking [p2, req, i] are the markings [p4, req] and [p2, req, req] reachable in Bank (without any interaction with MP_{X_1}(Bank)). Hence, they are also assigned to q3.

Besides sending, MP_{X_1}(Bank) can also receive the messages as and req; for example, MP_{X_1}(Bank) can receive the message req (i. e., it consumes a token from the place req), because Bank can be in the marking [p2, req]. Hence, state q4 and transition (q0, ?req, q4) is added to MP_{X_1}(Bank). Only the marking [p2] (i. e., [p2, req] after removing the token from place req) is related to the state q2, because no other marking is reachable from [p2] in Bank. So MP_{X_1}(Bank) has to trigger again.

The service automaton MP_{X_1}(Bank), which is constructed this way, is in general *infinite*, because it can send any message in every state q. Consequently, it eventually violates the message bound b; for example, sending message i in state q0 yields the state q3. In this state, a marking [p2, req, req] is reachable that violates $b = 1$. To avoid the construction of an infinite service automaton, we only add a sending transition to MP_{X_1}(Bank) if none of the markings of Bank that are related to the resulting state of MP_{X_1}(Bank) violates the message bound. This guarantees MP_{X_1}(Bank) to be finite.

In the following, we formally define the construction of $MP_{X_1}(N)$.

Construction algorithm

To construct $MP_{X_1}(N)$, we have to identify those sets of markings of N that are assigned to a state q of $MP_{X_1}(N)$. We refer to these markings as the *knowledge* that $MP_{X_1}(N)$ has of N in state q. We first define knowledge of N for arbitrary service automata. Afterwards, we present how the knowledge that $MP_{X_1}(N)$ has of N and hence $MP_{X_1}(N)$ can be constructed.

Let N and S be open nets. The next definition establishes a connection between the markings of N and the markings of $\mathit{inner}(S)$ (i.e., states of $\mathit{SA}(S)$). If $\mathit{inner}(S)$ is in a marking m_S, then the knowledge $k_{\mathit{inner}(S),N}(m_S)$ that $\mathit{inner}(S)$ has of N is the set of markings of N that N might be in while S is in marking m_S. In the definition, we refer to states of $\mathit{SA}(S)$ rather than to markings of $\mathit{inner}(S)$ (see our notation before Section 4.1), because $MP_{X_1}(N)$ will be a service automaton.

Definition 4.1.7 (knowledge of the potential markings of N).
Let S and N be open nets. The *knowledge that $\mathit{SA}(S)$ has of N* is a mapping $k_{\mathit{SA}(S),N} : Q_{\mathit{SA}(S)} \to 2^{\mathcal{M}(N)}$, defined by $k_{\mathit{SA}(S),N}(q) = \{m \mid (m_0, q_0) \xrightarrow{*} (m, q) \text{ in } N \oplus \mathit{SA}(S)\}$, for $q \in Q_{\mathit{SA}(S)}$. ⌟

The knowledge $k_{\mathit{SA}(S),N}(q)$ contains all markings of N that are reachable in $N \oplus \mathit{SA}(S)$ when $\mathit{SA}(S)$ is in state q. As we restricted ourselves to b-limited communication, the set $\mathcal{M}(N)$ of all possible markings of N is *finite*, and no marking in $\mathcal{M}(N)$ violates the message bound b.

As an example, Figure 4.4 shows the customer SA(Cust1), where each state of SA(Cust1) is annotated with the knowledge of the online bank Bank in Figure 4.3(a). For instance, in state s0 is $k_{\mathit{SA}(\mathsf{Cust1}),\mathsf{Bank}}(\mathsf{s0}) = \{[\mathsf{p0}], [\mathsf{p1}, \mathsf{as}], [\mathsf{p2}, \mathsf{req}]\}$.

The knowledge that $\mathit{SA}(S)$ has of N contains *all* markings of N that are reachable in $N \oplus S$. So we can derive every state of Bank $\oplus$ Cust1 from Figure 4.4. However, open net Bank cannot be reconstructed from the knowledge. Markings of Bank that are not reachable in Bank $\oplus$ Cust1—for example, the marking [p3]—do not appear in the knowledge, and the knowledge does not give information whether a marking of Bank is a final marking.

For the construction of $MP_{X_1}(N)$, we need two operations on sets of markings of N. One operation, *closure*, is used to calculate the states and the other, *event*, to calculate the transition relation of $MP_{X_1}(N)$.

The first operation, closure, calculates the knowledge for each state of $MP_{X_1}(N)$. The closure adds to a marking m of N all markings m' that are

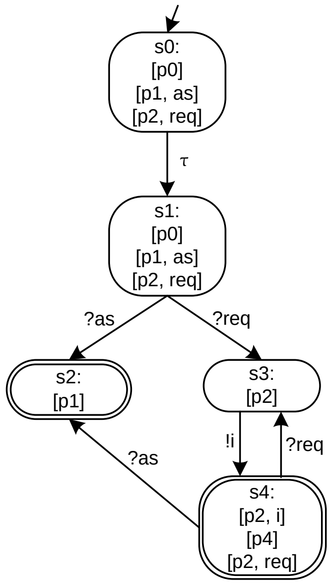


Figure 4.4.: Service automaton $SA(\mathsf{Cust1})$ annotated with the knowledge of Bank.

reachable from m in N.

Definition 4.1.8 (closure).
For a set $M \in 2^{\mathcal{M}(N)}$ of markings of an open net N, define $closure_N(M) = \{m' \mid m \in M \wedge m \xrightarrow{*} m'\}$. ⌟

The second operation for constructing $MP_{X_1}(N)$, event, corresponds to a transition in $MP_{X_1}(N)$. Given a set M of markings of N—that is, the knowledge that $MP_{X_1}(N)$ has of N in a state q—an event represents the *effect of a communication transition* of $MP_{X_1}(N)$ to M. We distinguish between *send events* and *receive events*. A send event models the effect that a message being sent by $MP_{X_1}(N)$ has on M. Accordingly, a receive event models the effect a message being received by $MP_{X_1}(N)$ has on M. As τ-events do not affect the knowledge of $MP_{X_1}(N)$, they shall be inserted as a τ-loop at each state in the definition of $MP_{X_1}(N)$.

Definition 4.1.9 (event).
Let N be an open net, let $M \in 2^{\mathcal{M}(N)}$ be a set of markings of N, and let $x \in P^I \cup P^O$. Let $[x]$ be a marking with $[x](x) = 1$ and $[x](y) = 0$, for all $y \neq x$. The *x-event of M in N* is defined as

$$event(M, x) = \begin{cases} \{m + [x] \mid m \in M\} & \text{if } x \in P^I; \\ \{m - [x] \mid m \in M, m(x) > 0\} & \text{if } x \in P^O. \end{cases}$$ ⌟

A send event x adds to each marking $m \in M$ a token on x. In contrast, a receive event x yields those markings m of M from which a token could be removed from x.

For the open net Bank in Figure 4.3(a) we have $\mathit{closure}_{\mathsf{Bank}}(\{[\mathsf{p0}]\}) = \{[\mathsf{p0}], [\mathsf{p1}, \mathsf{as}], [\mathsf{p2}, \mathsf{req}]\}$. For instance, the ?req-event is defined as $\mathit{event}(\mathit{closure}_{\mathsf{Bank}}(\{[\mathsf{p0}]\}), ?\mathsf{rej}) = \{[\mathsf{p2}]\}$, because only marking $[\mathsf{p2}, \mathsf{req}]$ contains a token on place req. As another example, the !ap-event is defined as $\mathit{event}(\mathit{closure}_{\mathsf{Bank}}(\{[\mathsf{p0}]\}), !\mathsf{ap}) = \{[\mathsf{p0}, \mathsf{ap}], [\mathsf{p1}, \mathsf{ap}, \mathsf{as}], [\mathsf{p2}, \mathsf{ap}, \mathsf{req}]\}$.

With the help of the two operations closure and event, we can construct the most permissive X_1-strategy $MP_{X_1}(N)$ of N. As we restricted ourselves to b-limited communication, the set $\mathcal{M}(N)$ of all possible markings of N is finite, and no marking in $\mathcal{M}(N)$ violates the message bound b. To this end, we require that the closure of the initial marking of N is an element of $\mathcal{M}(N)$ and hence does not violate b. Open nets that violate this requirement are not X_1-controllable. An example is the open net N1 in Figure 2.6(a), where we calculate $\mathit{closure}_{\mathsf{N1}}(m_0) = \{[\mathsf{p0}, \mathsf{p}], [\mathsf{p0}, \mathsf{p}, \mathsf{p}], \dots\}$.

Definition 4.1.10 (most permissive X_1-strategy).
Let $N = (P_N, T_N, F_N, P_N^I, P_N^O, m_{0N}, \Omega_N)$ be an open net where $\mathit{closure}_N(\{m_{0N}\}) \subseteq 2^{\mathcal{M}(N)}$. The *most permissive X_1-strategy* of N is defined as the service automaton $MP_{X_1}(N) = (Q, MC^I, MC^O, \delta, q_0, \Omega)$ with

- $Q = \{q_K \mid K \in 2^{\mathcal{M}(N)}\}$;
- $MC^I = P_N^O \wedge MC^O = P_N^I$;
- $\delta = \{(q_K, x, q_{K'}) \mid K \in 2^{\mathcal{M}(N)} \wedge x \in P_N^I \cup P_N^O \wedge K' = \mathit{closure}_N(\mathit{event}(K, x))\} \cup \{(q_K, \tau, q_K) \mid K \in 2^{\mathcal{M}(N)}\}$;
- $q_0 = q_{K_0}$ with $K_0 = \mathit{closure}_N(\{m_{0N}\})$;
- $\Omega = \{q_K \mid K \in 2^{\mathcal{M}(N)} \wedge K \cap \Omega_N \neq \emptyset\}$. ⌟

The construction of $MP_{X_1}(N)$ starts with the initial marking of N and computes its closure. The resulting set K_0 of markings yields the initial state q_0 of $MP_{X_1}(N)$. From the alphabet of $MP_{X_1}(N)$, we derive the set of all x-events of K_0. For each x, we add an outgoing transition to q_0. The corresponding successor state q of q_0, $q_0 \xrightarrow{x} q$, is constructed by computing the closure of the set $\mathit{event}(K_0, x)$ of markings. In addition, a τ-labeled self-loop is added to each state. That way, $MP_{X_1}(N)$ can be iteratively constructed. A state of $MP_{X_1}(A)$ is a final state if it contains a final marking of N.

The most permissive X_1-strategy $MP_{X_1}(\mathsf{Bank})$ of the online bank has 11 states. Ignore the Boolean formula in each state, then Figure 4.5 shows the underlying automaton of $MP_{X_1}(\mathsf{Bank})$. The final states of $MP_{X_1}(\mathsf{Bank})$ are q1 and q6.

During the construction of the most permissive X_1-strategy, states may be calculated that contain markings violating b-limited communication. These states are removed, because they violate the definition of a state in Definition 4.1.10. An example is the marking $[\mathsf{p2}, \mathsf{req}, \mathsf{req}]$ in state q3 in Figure 4.3(b).

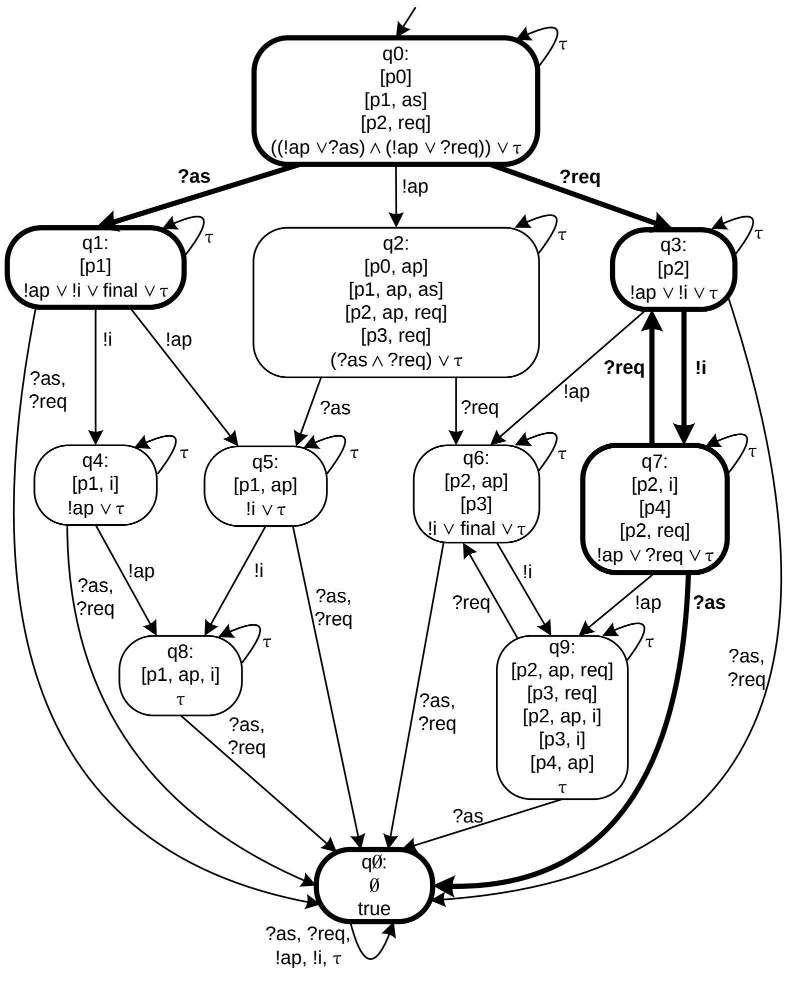


Figure 4.5.: X_1-operating guideline of Bank. Bold lines illustrate the minimal simulation relation with SA(Cust1).

For each open net N that contains at least one output place, the algorithm of Definition 4.1.10 constructs a particular state, the *empty state* $q_\emptyset$. This state is added if a receive event x for a set M of markings is calculated, and *no* marking of M marks the place x. In this case, the x-event yields the empty set of markings (cf. Definition 4.1.9). Applying any x-event to the empty state always leads to the empty state again.

Proposition 4.1.11 (empty state).
Let $MP_{X_1}(N)$ be the most permissive X_1-strategy of some open net N. The

empty state $q_\emptyset$ of $MP_{X_1}(N)$ is δ-reachable in $MP_{X_1}(A)$ iff N has at least one output place. Furthermore, for each transition $(q_\emptyset, x, q) \in \delta : \ q = q_\emptyset$. ⌟

Every incoming transition (except for the self-loops) of the empty state of $MP_{X_1}(\mathsf{Bank})$ in Figure 4.5 is a receiving transition of $MP_{X_1}(\mathsf{Bank})$. None of these receiving transitions can appear in the composition of $MP_{X_1}(\mathsf{Bank})$ and Bank, because there is no corresponding sending transition in Bank, and hence the receiving transition in $MP_{X_1}(\mathsf{Bank})$ will never be enabled. Nevertheless, any such receiving transition in $MP_{X_1}(\mathsf{Bank})$ is valid behavior and does not harm Bank.

Properties of the most permissive X_1-strategy

The aim for constructing $MP_{X_1}(N)$ of an open net N was to get a finite, deterministic service automaton that simulates the corresponding service automaton of every X_1-strategy of N. The next lemma proves that $MP_{X_1}(N)$ has indeed these properties.

Lemma 4.1.12 (properties of $MP_{X_1}(N)$).
Let N be an open net, and let $MP_{X_1}(N)$ be its most permissive X_1-strategy. The following properties hold:

- For each partner S, there is a minimal simulation relation of $SA(S)$ by $MP_{X_1}(N)$ [Mas09, Corollary 5.4.4, p.167].
- $MP_{X_1}(N)$ is deterministic [Mas09, Corollary 5.2.12, p.154].
- $MP_{X_1}(N)$ is finite (i. e., has finitely many states and a finite interface) and, for each state q_K of $MP_{X_1}(N)$, K is finite [Mas09, Lemma 5.4.3, p.166]. ⌟

Finiteness of $MP_{X_1}(N)$ for a bounded open net N with b-limited communication guarantees that $MP_{X_1}(N)$ can be computed.

The minimal simulation relation of $SA(\mathsf{Cust1})$ in Figure 4.4 by $MP_{X_1}(\mathsf{Bank})$ in Figure 4.5 is $\varrho = \{(\mathsf{s0},\mathsf{q0}),(\mathsf{s1},\mathsf{q0}),(\mathsf{s2},\mathsf{q1}),(\mathsf{s3},\mathsf{q3}),(\mathsf{s4},\mathsf{q7}),(\mathsf{s2},\mathsf{q}\emptyset)\}$. Notice the importance of the empty state. Without this state there would be no simulation relation, because in related states $(\mathsf{s3},\mathsf{q3})$, transition ?as could not be mimicked in $MP_{X_1}(\mathsf{Bank})$.

For any partner S, we can even prove a stronger property than the existence of a minimal simulation relation ϱ of $SA(S)$ by $MP_{X_1}(N)$. Relation ϱ actually establishes a relation between the knowledge values of the involved states of $SA(S)$ and of $MP_{X_1}(N)$. Hence, the knowledge that $MP_{X_1}(N)$ has of N can be used to deduce the knowledge that $SA(S)$ has of N.

Lemma 4.1.13 (knowledge of S is union of knowledge of $MP_{X_1}(N)$).
Let $MP_{X_1}(N)$ be the most permissive X_1-strategy of an open net N, let S be a partner of N, and let $\varrho \subseteq Q_{SA(S)} \times Q_{MP}$ be the minimal simulation relation of

$SA(S)$ by $MP_{X_1}(N)$. For each state q_S of $SA(S)$ holds:

$$k_{SA(S),N}(q_S) = \bigcup_{(q_S,q)\in\varrho} k_{MP,N}(q) \quad .$$

⌟

For a proof of Lemma 4.1.13 see [Mas09, Lemma 5.2.14, p.156].

Relation ϱ relates a state q_S of $SA(S)$ to a set of states of $MP_{X_1}(N)$. The knowledge that $SA(S)$ has of N in q_S is equal to the union of the knowledge values of all states q of $MP_{X_1}(N)$ with $(q_S, q) \in \varrho$.

For an illustration of Lemma 4.1.13, consider again the minimal simulation relation of $SA(\mathsf{Cust1})$ by $MP_{X_1}(\mathsf{Bank})$. From this relation, we conclude by Lemma 4.1.13:

$$\begin{array}{lcl}
k_{SA(\mathsf{Cust1}),\mathsf{Bank}}(\mathsf{s0}) & = & k_{MP,\mathsf{Bank}}(\mathsf{q0}) \\
k_{SA(\mathsf{Cust1}),\mathsf{Bank}}(\mathsf{s1}) & = & k_{MP,\mathsf{Bank}}(\mathsf{q0}) \\
k_{SA(\mathsf{Cust1}),\mathsf{Bank}}(\mathsf{s2}) & = & k_{MP,\mathsf{Bank}}(\mathsf{q1}) \cup k_{MP,\mathsf{Bank}}(\mathsf{q}\emptyset) \\
k_{SA(\mathsf{Cust1}),\mathsf{Bank}}(\mathsf{s3}) & = & k_{MP,\mathsf{Bank}}(\mathsf{q3}) \\
k_{SA(\mathsf{Cust1}),\mathsf{Bank}}(\mathsf{s4}) & = & k_{MP,\mathsf{Bank}}(\mathsf{q7})
\end{array}$$

The state s2 of $SA(\mathsf{Cust1})$ is an example, where the knowledge is equal to a union of knowledge values.

4.1.3. X_1-operating guidelines

The most permissive X_1-strategy $MP_{X_1}(N)$ of an open net N provides only the structure A of an annotated automaton A^ϕ describing *all* X_1-strategies of N. In this section, we present an algorithm to construct the second ingredient of A^ϕ, the *annotation function*, which assigns to each state of $MP_{X_1}(N)$ a Boolean formula. Recall that these Boolean formulae are needed to exclude all open nets that are not an X_1-strategy of N, but are simulated by $MP_{X_1}(N)$.

We construct the annotation function in two steps. First, we show how deadlock freedom of a composition $N \oplus S$ can be decided based on $MP_{X_1}(N)$ and its knowledge of N. In a second step, we show how this criterion can be encoded into a Boolean formula.

A characterization of deadlock freedom

Based on the knowledge that $SA(S)$ has of N and the information about final markings of N, deadlock freedom of $N \oplus S$ can be characterized from the point of view of $SA(S)$. This result has been proved in [Mas09, Corollary 5.1.6, p.146].

Lemma 4.1.14 (characterization of deadlock freedom).
The composition $N \oplus S$ of two open nets N and S is deadlock-free iff, for all states q of $SA(S)$ and each marking m of N with $m \in k_{SA(S),N}(q)$, at least one of the following three conditions holds:

(i) $m \xrightarrow{t}$, for some $t \in T_N$; or

(ii) $q \xrightarrow{x}$, for some $x \in (P_S^I \cup P_S^O \cup \{\tau\}) \wedge m(x) > 0$ if $x \in P_S^I$; or

(iii) (m, q) is a final state of $N \oplus SA(S)$, i. e., $m \in \Omega_N \wedge q \in \Omega_{SA(S)}$. ⌟

Lemma 4.1.14 enables us to decide the absence of deadlocks in the composition $N \oplus S$ by checking only the three conditions, for each marking of N in the knowledge of each state q of $SA(S)$. The conditions require that (i) a transition in N is enabled at a marking m; (ii) state q has an outgoing transition, and in case it is a receiving transition, then the corresponding interface place in N is marked; or (iii) the composition is in a final state. If one of these three conditions holds for a marking $m \in k_{SA(S),N}(q)$ of N, state (m, q) is not a deadlock in $N \oplus SA(S)$. The other way around, if none of these three conditions holds, marking (m, q) is a deadlock in $N \oplus SA(S)$.

Consider the marking [p1] in the state s2 of SA(Cust1) in Figure 4.4. In this example, the third condition holds; that is, [p1] is a final marking of Bank, and s2 is a final state of SA(Cust1). Consequently, ([p1], s2) is not a deadlock. As all other markings satisfy at least one of the three conditions, we conclude that Bank $\oplus$ Cust1 is deadlock-free.

As a consequence of Lemma 4.1.13, we do not have to compute the knowledge of $SA(S)$. Instead, we can use $MP_{X_1}(N)$ to decide whether or not a marking in $N \oplus S$ is deadlock-free. This can be done by first checking the existence of a minimal simulation relation ϱ of S by $MP_{X_1}(N)$. If ϱ does not exist, S is not an X_1-strategy of N. Otherwise, by checking the knowledge of all states of $MP_{X_1}(N)$ that are in ϱ, we can decide whether $N \oplus S$ is deadlock-free.

An annotation function to encode deadlock freedom

In the sequel, we present a way to make the procedure of deciding whether $N \oplus S$ is deadlock-free more efficient. To this end, we directly encode the three conditions of Lemma 4.1.14 into an annotation function, which assigns a Boolean formula to each state of $MP_{X_1}(N)$. Besides the underlying automaton of $MP_{X_1}(N)$, the annotation function will be the second ingredient of the X_1-operating guideline of N.

For each state q_K of $MP_{X_1}(N)$ that occurs in ϱ, we have to check each marking $m \in K$. As the three conditions are a disjunction, the formula we are looking for is a conjunction over all markings $m \in K$, and each such conjunct is a disjunction encoding the three conditions. The encoding works as follows: Condition (i) is encoded by *true* if m satisfies condition (i); else by *false*. Condition (ii) holds if the corresponding state q of $SA(S)$, $(q, q_K) \in \varrho$, enables any internal transition, any sending transition, or any receiving transition (where the respective interface place is marked at m). This can be encoded by a disjunction over the respective literals. Finally, condition (iii) is encoded by *final* if m is a final marking and by *false* else. Because the conjunct for m trivially holds if m satisfies condition (i), it is sufficient to restrict ourselves to markings $m \in K$ that do not enable any transition of N (i. e., $m \in K \wedge \forall t \in T_N : m \stackrel{t}{\not\rightarrow}$). Thus, we do not have to encode condition (i).

Definition 4.1.15 (annotation function).
For an open net $N = (P_N, T_N, F_N, P^I_N, P^O_N, m_{0N}, \Omega_N)$, let $MP_{X_1}(N) = (Q, MC^I, MC^O, \delta, q_0, \Omega)$ be its most permissive X_1-strategy. Define the *annotation function of* $MP_{X_1}(N)$ as a mapping $\psi_{MP} : Q \rightarrow \mathcal{BF}^{\mathcal{MC}^+}$ where, for each state q_K of $MP_{X_1}(N)$

$$\psi_{MP}(q_K) = \bigwedge_{m \in K \wedge \forall t \in T_N : m \stackrel{t}{\not\rightarrow}} \Big(\psi_{ii}(m) \vee \psi_{iii}(m)\Big) \quad \text{with}$$

- $\psi_{ii}(m) = \tau \vee \left(\bigvee_{x \in MC^O} x\right) \vee \left(\bigvee_{x \in MC^I, m(x) > 0} x\right)$; and
- $\psi_{iii}(m) = \begin{cases} \mathit{final}, & \text{if } m \in \Omega_N; \\ \mathit{false}, & \text{otherwise.} \end{cases}$ ⌟

Let q_S be a state in $SA(S)$, which is related to a state q_K of $MP_{X_1}(N)$ by the minimal simulation relation ϱ of $SA(S)$ by $MP_{X_1}(N)$. The formula $\psi_{ii}(m)$, encoding the second condition of Lemma 4.1.14, evaluates to *true* if q_S enables any transition and thus assigns *true* to any of the literals of $MC^I \cup MC^O$ of $MP_{X_1}(N)$ or to the literal τ.

The formula $\psi_{iii}(m)$ encodes that state (m, q_S) is a final state in $N \oplus SA(S)$. In this case, m is a final marking of N. Then, the formula $\psi_{iii}(m)$ contains a literal *final* and by Definition 4.1.3, $SA(S)$ assigns *true* to *final* if q_S is a final state of $SA(S)$.

For the empty state $q_\emptyset$ of $MP_{X_1}(N)$, the knowledge is the empty set. Hence, the Boolean formula of $MP_{X_1}(N)$ at state $q_\emptyset$ is always equal to the empty conjunction (which is defined to be *true*).

As an example, for $MP_{X_1}(\mathsf{Bank})$ in Figure 4.5, we calculate the following annotations for the states q0–q3:

$$\begin{array}{lcl}
\psi_{MP}(\mathsf{q0}) & = & (\tau \vee \mathsf{!ap} \vee \mathsf{!i} \vee \mathsf{?as} \vee \mathit{false}) \wedge (\tau \vee \mathsf{!ap} \vee \mathsf{!i} \vee \mathsf{?req} \vee \mathit{false}) \\
\psi_{MP}(\mathsf{q1}) & = & \tau \vee \mathsf{!ap} \vee \mathsf{!i} \vee \mathit{final} \\
\psi_{MP}(\mathsf{q2}) & = & (\tau \vee \mathsf{!ap} \vee \mathsf{!i} \vee \mathsf{?as} \vee \mathit{false}) \wedge (\tau \vee \mathsf{!ap} \vee \mathsf{!i} \vee \mathsf{?req} \vee \mathit{false}) \\
\psi_{MP}(\mathsf{q3}) & = & \tau \vee \mathsf{!ap} \vee \mathsf{!i} \vee \mathit{false}
\end{array}$$

The automaton of the most permissive X_1-strategy $MP_{X_1}(N)$ of N and the annotation function ψ_{MP} define the X_1-operating guideline of N.

Definition 4.1.16 (X_1-operating guideline).
Let $MP_{X_1}(N) = (Q, MC^I, MC^O, \delta, q_0, \Omega)$ be the most permissive X_1-strategy of an open net N, and let ψ_{MP} be the annotation function of $MP_{X_1}(N)$. Let A be

an automaton, which is isomorphic to $(Q, MC^I, MC^O, \delta, q_0)$ under isomorphism h, and let $\phi : Q_A \rightarrow \mathcal{BF}^{\mathcal{MC}^+}$ be an annotation function of A, defined as $\phi(q_A) = \psi_{MP}(h(q_A))$, for all $q_A \in Q_A$. The X_1*-operating guideline* $OG_{X_1}(N)$ *of* N is the annotated automaton A^ϕ. ⌟

The next theorem, which is also the main theorem of this section, shows that an X_1-operating guideline of N characterizes all X_1-strategies of N. For the proof, we refer to [Mas09, Corollary 5.4.10, p.175].

Theorem 4.1.17 (X_1-operating guideline represents all X_1-strategies). For any open net with X_1-operating guideline $OG_{X_1}(N)$ holds:

$$\mathit{Match}(OG_{X_1}(N)) = \mathit{Strat}_{X_1}(N) \quad .$$ ⌟

Theorem 4.1.17 justifies that, given an open net N, we have an algorithm for constructing the X_1-operating guideline of N.

Constructing the most permissive X_1-strategy and the canonical Boolean annotation according to Definitions 4.1.10 and 4.1.15, respectively, the resulting X_1-operating guideline is not necessarily in normal form. As an X_1-operating guideline is a special annotated automaton, we can apply the normalization procedure for annotated automata as introduced in Section 4.1.1. By Theorem 4.1.5, the normalization does not affect any X_1-strategy of N. Hence, it is sufficient to consider only normalized X_1-operating guidelines in the following. Note that the normalization only normalizes the annotations. It never removes any state of an X_1-operating guideline, because each conjunct of every annotation contains a literal τ, and each state of the X_1-operating guideline has a τ-labeled self-loop.

The X_1-operating guidelines of the online bank is illustrated in Figure 4.5. The annotations of the states q8 and q9 are normalized to τ.

Another important fact is that once $OG_{X_1}(N)$ is calculated, we can remove the knowledge from each state of $OG_{X_1}(N)$. To this end, Definition 4.1.16 defines a new automaton A that is isomorphic to the structure of the most permissive X_1-strategy of N. The knowledge is needed for the construction of $OG_{X_1}(N)$, rather than for matching any open net with $OG_{X_1}(N)$. As an advantage, we have to store only the much more compact annotated automaton without its knowledge of N. Additionally, trade secrets of N are preserved if we publish $OG_{X_1}(N)$, because no other party can observe internal states of N by looking at the X_1-operating guideline.

Finally, we conclude from Lemma 4.1.6 the open net $PN(MP_{X_1}(N))$ matches with the X_1-operating guideline of N.

4.1.4. Experimental results and discussion

Based on a prototypical implementation of the algorithm for constructing the X_1-operating guideline of a service, we experiment with a number of real-life service models. Furthermore, we discuss some complexity issues and some related work. As we *did not contribute* in the development of X_1-operating guidelines, this section is not devoted to evaluate X_1-operating guidelines. Instead, we aim at providing information that enables us to compare X_1-operating guidelines with the three other finite representations of X-strategies we will present later in this chapter.

Experimental results

The construction algorithm for X_1-operating guidelines has been implemented in the service analysis tool Fiona[2] [MW08, LMSW08]. Among others, Fiona can be used to read an open net N, to calculate the X_1-operating guideline of N, and to check whether an open net matches with an X_1-operating guideline.

Deadlock freedom is a less restrictive correctness criterion, because every open net is X_1-controllable; for example, an open net that executes an infinite τ-loop is an X_1-strategy of every open net. As an X_1-strategy may put on each input place of N b messages and then execute an infinite τ-loop, the most permissive X_1-strategy tends to become great in size. For that reason, Fiona implements *responsiveness* instead of deadlock freedom. Responsiveness is a more restrictive correctness criterion than deadlock freedom. A composition $N \oplus S$ is responsive if and only if it is deadlock-free, and each livelock contains at least one marking that enables a prior interface transition of N or of S. Formally, for all reachable markings m of $N \oplus S$, m is not a deadlock and, for all livelocks $\{m_1, \ldots, m_k\}$ of $N \oplus S$, there exists a marking m_i with $i = 1, \ldots, k$ and a transition t of $N \oplus S$ such that $m_i \xrightarrow{t}$ and t is an interface transition of N or of S. That means, responsiveness guarantees the absence of deadlocks, and in addition it guarantees the absence of livelocks where N and S do not interact with each other. For instance, the two previously mentioned X_1-strategies are not responsive and hence excluded by Fiona. For the online bank Bank, Fiona calculates the annotated automaton in Figure 4.6 as an X_1-operating guideline in case of responsiveness. However, although Fiona implements responsiveness, our theory is based on deadlock freedom.

Table 4.1 provides the results of our experiment including 16 service models which were specified as open nets. The first five examples are realistic service models taken from the WS-BPEL specification [Alv07] (‘Loan Approval’, ‘Purchase Order’ and ‘Travel Service 1’), and from [AFFK05] (‘Olive Oil Ordering’). ‘Travel Service 2’ is a modification of ‘Travel Service 1’. As these examples were specified in the service description language WS-BPEL [Alv07], we had to translate them into open nets using the compiler BPEL2oWFN [Loh08].

The ‘Beverage Machine’ is taken from [LMW07b], and the online shops are taken from [LMSW06]. ‘Philosophers’ are an open net model of three and five dining philosophers. ‘SMTP Protocol’ models the SMTP protocol. The last five examples are real-life service models provided by a consultant company.

[2]available at `http://www.service-technology.org/fiona`

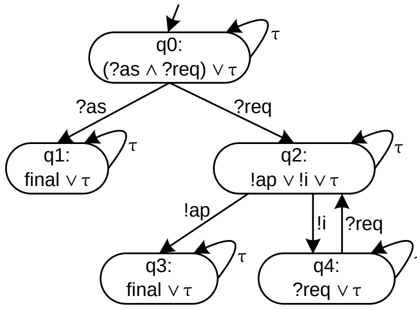


Figure 4.6.: X_1-operating guideline of Bank in case of responsiveness. The empty state is ommitted.

For each open N, we computed the X_1-operating guideline $OG_{X_1}(N)$, for $b = 1$, using Fiona. Table 4.1 provides information about the size of the open net N, the size of its behavior, the state space of the most permissive X_1-strategy $MP_{X_1}(N)$ of N, the state space of the most permissive X_1-strategy $MP^R_{X_1}(N)$ of N in case of responsiveness, and the time for computing $MP^R_{X_1}(N)$. More precisely, columns 2–4 refer to the number of places ($|P|$), transitions ($|T|$), and interface places ($|P^{Int}| = |P^I \cup P^O|$) in N, respectively; columns 5 and 6 refer to the number of states ($|Q|$) and edges ($|\delta|$) in the corresponding service automaton $SA(N)$, respectively; columns 7 and 8 show whether N has concurrency ($||$) and contains a cycle ($\circlearrowleft$), respectively; columns 9 and 10 refer to the number of states ($|Q|$) and edges ($|\delta|$) of $MP_{X_1}(N)$, respectively; the size of $MP^R_{X_1}(N)$ is shown in columns 11 and 12. Finally, the most right column denotes the time for computing the X_1-operating guideline of N.

As an example, the corresponding open net of 'Online Shop 1' has 61 places, 58 transitions, and 7 interface places. 'Online Shop 1' has 205 states and 463 edges. It contains concurrency, but it is acyclic. The most permissive X_1-strategy of 'Online Shop 1' has 117 states and 312 transitions; in case of responsiveness, it has only 13 states and 50 transitions. The time for computing the X_1-operating guideline is 2 seconds.

Based on the experimental results, we make the following observations: The X_1-operating guidelines of these service models were calculated in reasonable time; most of them within a few seconds. However, calculating the X_1-operating guideline of the 'SMPT Protocol' took about 47 minutes. A second observation is that the most permissive X_1-strategy $MP_{X_1}(N)$ contains in general far more states than the most permissive X_1-strategy $MP^R_{X_1}(N)$ in case of responsiveness. In case of 'Process 2', MP_{X_1} is almost 100 times greater in size than $MP^R_{X_1}$, for instance. We discuss these observation in the following.

Table 4.1.: Calculation of $OG_{X_1}(N)$ with Fiona. All experiments were obtained on an UltraSPARC III processor with 900MHz and 4 GB RAM running Solaris 10.

Service	N			$SA(N)$		Props		$MP_{X_1}(N)$		$MP^R_{X_1}(N)$		time
	$\|P\|$	$\|T\|$	$\|P^{Int}\|$	$\|Q\|$	$\|\delta\|$	$\|\|$	$\circlearrowleft$	$\|Q\|$	$\|\delta\|$	$\|Q\|$	$\|\delta\|$	mm:ss
Loan Approval	38	25	6	26	33	✓	-	47	102	8	30	00:00
Purchase Order	22	7	10	12	15	✓	-	1,025	5,163	169	1,182	00:02
Olive Oil Ordering	12	6	6	6	6	-	-	47	137	17	67	00:00
Travel Service 1	15	5	8	7	7	✓	-	257	1,035	57	300	00:00
Travel Service 2	22	9	12	10	11	✓	-	4,097	22,494	289	2,252	00:10
Online Shop 1	61	58	7	205	463	✓	-	117	312	13	50	00:02
Online Shop 2	72	69	8	308	744	✓	-	113	315	8	35	00:04
Beverage Machine	12	8	7	5	8	-	✓	21	76	12	54	00:00
Philosophers #3	22	10	6	46	70	✓	✓	225	600	67	243	00:01
Philosophers #5	36	16	10	574	1,476	✓	✓	11,165	42,310	1,433	8,390	01:25
SMPT Protocol	123	133	12	8,345	34,941	-	✓	7,585	37,288	1,217	8,408	47:25
Registration	127	88	6	1,057	2,927	✓	-	33	75	8	30	00:10
Process 1	39	27	20	97,511	468,072	✓	-	995	14,143	897	13,548	00:07
Process 2	29	18	13	244	758	✓	-	1,685	16,675	1,608	16,445	00:07
Process 3	26	15	13	992	3,744	✓	-	21,729	130,751	237	1,437	07:57
Process 4	29	14	19	160	494	✓	-	41,185	402,317	6,821	154,713	06:24

Discussion

We start with a short complexity analysis for X_1-operating guidelines. Similar considerations for acyclic open nets can be found in [MW07]. Let $|TS|$ denote the size of a transition system TS, where $|TS|$ is defined as $|Q_{TS}| + |\delta_{TS}|$. When referring to the size of an open net N, we always mean the size of its corresponding service automaton $SA(N)$.

To compute an X_1-operating guideline $OG_{X_1}(N)$ of N, the most permissive X_1-strategy $MP_{X_1}(N)$ of N has to be computed. The number of states of $MP_{X_1}(N)$ is proportional to the number $2^{\mathcal{M}(N)}$ of knowledge sets of N (see Definition 4.1.10). We calculate this size in two steps. First, consider the inner subnet of N. The number of knowledge sets for *inner*(N) is proportional to 2^x with x is the number of the reachable markings of *inner*(N). Second, consider the interface of N, and let $k = |P^I \cup P^O|$ denote the number of interface places of N. For a given communication bound b, we have $(b+1)^k$ markings. The overall space complexity of $MP_{X_1}(N)$ is proportional to the product of the knowledge sets of *inner*(N) and the interface of N. Hence, we have $\mathcal{O}(|MP_{X_1}(N)|) = 2^x \cdot (b+1)^k$, which is also the time complexity for computing $MP_{X_1}(N)$.

Computing $OG_{X_1}(N)$ is proportional in time to the product of the number of states of N and its most permissive X_1-strategy $MP_{X_1}(N)$; that is, we have $\mathcal{O}(|N| \cdot |MP_{X_1}(N)|)$. In addition, we have to compute the annotation function. To this end, we have to consider each state q of $MP_{X_1}(N)$ and traverse through its knowledge $k_{MP,N}(q)$, building the conjunction of at most k literals. This takes $\mathcal{O}(|N| \cdot |MP_{X_1}(N)| \cdot k)$. As we assume the number k of interface places to be such small that it is dominated by the rest of product, we conclude that the overall time and space complexity of $OG_{X_1}(N)$ is proportional to $\mathcal{O}(|N| \cdot |MP_{X_1}(N)|)$.

Matching an open net S with $OG_{X_1}(N)$ means to find a minimal simulation relation ϱ of $SA(S)$ by $OG_{X_1}(N)$. This is proportional in time to the product of the single state spaces of $OG_{X_1}(N)$ and S. So we have $\mathcal{O}(|OG_{X_1}(N)| \cdot |S|)$. For each pair $(q_{SA(S)}, q_{OG}) \in \varrho$ of related states, it has to be checked whether $q_{SA(S)}$ satisfies the annotation $\phi(q_{OG})$. An annotation has at most k many different variables, and hence there are at most 2^k different assignments. So independent of the size of the formula, the check takes at most time $\mathcal{O}(2^k)$. Thus, we have the overall complexity of $\mathcal{O}(|OG_{X_1}(N)| \cdot |S| \cdot 2^k)$. As we assume the number of interface places of N to be such small that even 2^k is dominated by the number of states of a composition [MW07], we conclude that matching takes $\mathcal{O}(|OG_{X_1}(N)| \cdot |S|)$.

Although the time and space complexity of $OG_{X_1}(N)$ is exponential, our experimental results in Table 4.1 and other experimental results (e. g., in [LMW07b]) show that the calculation of $OG_{X_1}(N)$ is feasible in practical applications both for time and space. One reason is that the computation of the X_1-operating guideline gives room for optimizations.

Regarding the time complexity, Table 4.1 illustrates that the number of states of $MP_{X_1}(N)$ is typically much greater than the number of states of $MP^R_{X_1}(N)$. This is one of the reasons why responsiveness rather than deadlock freedom is implemented in Fiona. To compute the X_1-operating guideline for responsiveness, Fiona calculates first $MP_{X_1}(N)$. Clearly, this causes a lot of overhead. Therefore, a number of optimizations exist. One idea is to analyze the reachability graph of the inner subnet of N to derive information about the interaction behavior of

N with any X_1-strategy. That way, information about which messages are never sent or received or message bounds can be computed [SW09b]. This information can then be used during the computation of the X_1-operating guideline of N.

The automaton-based representation of an X_1-operating guideline causes overhead—for example, the diamond structures that represent concurrency in the most permissive X_1-strategy. Some effort has been investigated to reduce the (average) space complexity of X_1-operating guidelines. Kaschner et al. [KMW07] apply BDD-based symbolic representations to reduce the size of an X_1-operating guideline. Gierds [Gie08] presents an algorithm that replaces diamond structures in X_1-operating guidelines by a more compact structure. Recently an approach has been published [LW09] that transforms the underlying automaton of the X_1-operating guideline into a Petri net by applying the theory of regions. Investigating regularities in the annotations, the annotations can be represented in a compact manner. Experimental results in [LW09] show that the resulting representation is very space-efficient. In fact, the worst-case space complexity reduces from $\mathcal{O}(|N| \cdot |MP_{X_1}(N)|)$ to the size of $MP_{X_1}(N)$.

Besides responsiveness, there also exists variants of X_1-operating guidelines that depend on the notion of a final state used for open nets, for instance. For an overview, we refer to [Mas09].

4.2. Representing strategies in case of deadlock freedom and cover

In the previous section, we recapitulated X_1-operating guidelines as a finite representation of all open nets S for a given open net N such that the composition $N \oplus S$ is deadlock-free. We denoted such an open net S as an X_1-strategy of N. The aim of this section is to introduce a finite representation of X_1-strategies S' of N such that a given set Y of nodes of N is covered in the composition $N \oplus S'$. We refer to S' as an $X_2(Y)$-strategy of N.

The information for deciding whether or not S' is an $X_2(Y)$-strategy of N can actually be derived from the knowledge that $OG_{X_1}(N)$ has of N (Section 4.2.1). This information can be encoded into a global constraint. Basically, the global constraint specifies which states of $OG_{X_1}(N)$ must be contained in the minimal simulation relation with the corresponding service automaton $SA(S')$ of S'. The X_1-operating guideline of N with the global constraint defines the $X_2(Y)$-operating guideline of N, which represents all $X_2(Y)$-strategies of N (Section 4.2.2). The $X_2(Y)$-operating guideline can be adapted such that it represents all X_2-strategies of N; that is, all open nets such that the composition with N is quasi-live (Section 4.2.3). Based on a prototypical implementation, we present experimental results and compare X_1-operating guidelines and $X_2(Y)$-operating guidelines in Section 4.2.4.

The results of this section have been published in [SW08, SW09a].

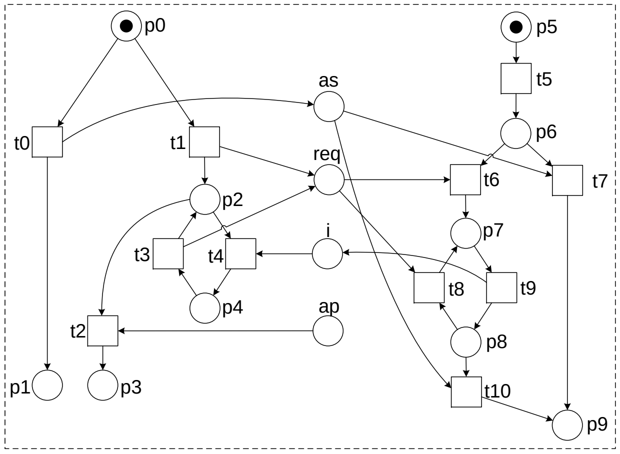


Figure 4.7.: The composition of the online bank Bank and the customer Cust1.

4.2.1. Deciding coverability of open-net nodes

The motivation for covering open-net nodes is to figure out if some functionality of a service—that is, some communication pattern, such as a credit approval—can in principle be used by other services. On the level of open nets, we have to check whether there exists an open net S for an open net N such that $N \oplus S$ is deadlock-free and, in addition, some nodes Y of N—for example, transition credit approval—are covered in $N \oplus S$.

As an example, the composition of the online bank Bank and the customer Cust1 is illustrated in Figure 4.7 again. Cust1 is an $X_2(Y)$-strategy of Bank for $Y = \{\mathsf{p1}\}$, because marking $[\mathsf{p1}, \mathsf{p5}, \mathsf{as}]$ is reachable in the composition. For $Y = \{\mathsf{t2}\}$, Cust1 is not an $X_2(Y)$-strategy of Bank, because this transition cannot be enabled in the composition.

Every $X_2(Y)$-strategy of an open net N is an X_1-strategy of N as well, and covering nodes Y of N restricts the set of X_1-strategies of N. Thus, we conclude $\mathit{Strat}_{X_2(Y)}(N) \subseteq \mathit{Strat}_{X_1}(N)$.

To decide when an X_1-strategy of an open net N is an $X_2(Y)$-strategy of N, we need a criterion to distinguish between an X_1-strategy of N and an $X_2(Y)$-strategy of N. The first observation is that an interface place p of N is covered in $N \oplus S$ if and only if a corresponding interface transition of S can be enabled in $N \oplus S$. Suppose $Y = \{\mathsf{ap}\}$ is the set of nodes of Bank to be covered. In this case, Cust1 cannot cover Y, because place ap cannot be marked in the composition Bank $\oplus$ Cust1. This can also be observed at the X_1-operating guideline $OG_{X_1}(\mathsf{Bank})$ in Figure 4.5, because no !ap-labeled transition is present in the minimal simulation relation of $SA(\mathsf{Cust1})$ by $OG_{X_1}(\mathsf{Bank})$. However, as we want to cover arbitrary nodes of an open net—that is, also internal places and transitions—the information that we can extract from the labels of an X_1-operating guideline is not sufficient.

According to Definition 2.6.4, a transition t (place p) is covered if it can be fired (marked) in the composition $N \oplus S$. This is equivalent to the existence of a reachable marking in $N \oplus S$ that enables t (marks p). The information about which markings of N are reachable in the composition $N \oplus S$ are, in fact, given by the knowledge that S has of N. In addition, Lemma 4.1.13 provides a relation of the knowledge that S has of N and the knowledge that the X_1-operating guideline of N has of N. So, we can decide whether a node of N is covered in the composition of N and an X_1-strategy S of N by exploring the knowledge values of each state of the X_1-operating guideline of N that is contained in the minimal simulation relation of S by $OG_{X_1}(N)$.

The following lemma uses these facts about X_1-operating guidelines to decide on the basis of an X_1-operating guideline whether a node of N is covered in the composition of N and an X_1-strategy S of N.

Lemma 4.2.1 (place/transition coverability).
Let an open net S be an X_1-strategy of an open net N, and let ϱ be the minimal simulation relation of $SA(S)$ by $OG_{X_1}(N)$. A transition $t \in T_N$ (a place $p \in P_N$) is covered in $N \oplus S$ iff there is a state $q \in Q$ of $OG_{X_1}(N)$, a state q_S of $SA(S)$, and a marking $m_N \in k_{OG(N),N}(q)$ with $(q_S, q) \in \varrho$, and t is enabled at m_N ($m_N(p) \geq 1$). ⌟

Proof.
We present the proof only for the case of a covered transition $t \in T_N$. The case of a covered place $p \in P_N$ is analogous.

($\Rightarrow$): Let N, $OG_{X_1}(N)$, and $S \in Strat_{X_1}(N)$ be given. Let transition t be covered in $N \oplus SA(S)$. According to Definition 2.6.4, there is a run $(m_{0N}, q_{0S}) \xrightarrow{t_1} \ldots \xrightarrow{t_n} (m_N, q_S) \xrightarrow{t} (m'_N, q'_S)$ in $N \oplus SA(S)$. As t is a transition of N, t is enabled at m_N. By Definition 4.1.7, we have $m_N \in k_{SA(S),N}(q_S)$. As $S \in Strat_{X_1}(N)$, there must be (by Lemma 4.1.13) a state q in $OG_{X_1}(N)$ with $m_N \in k_{OG(N),N}(q)$ and $(q_S, q) \in \varrho$ (the minimal simulation relation of $SA(S)$ by $OG_{X_1}(N)$).

($\Leftarrow$): Let N be an open net, and let $OG_{X_1}(N) = (Q, MC^I, MC^O, \delta, q_0, \phi)$. Let S be an X_1-strategy of N. Because S is an X_1-strategy of N, there is a minimal simulation relation ϱ of the states in $SA(S)$ by the states in Q. Let q_S, q, and m_N be as assumed. Thus, $(q_S, q) \in \varrho$, $m_N \in k_{OG(N),N}(q)$, and t is enabled at m_N. From Lemma 4.1.13 follows $m_N \in k_{SA(S),N}(q_S)$. Consequently, there is a run $(m_{0N}, q_{0S}) \xrightarrow{*} (m_N, q_S)$ in $N \oplus SA(S)$. As t is enabled at m_N, t is also enabled at (m_N, q_S). Hence, the transition t is covered in $N \oplus S$. □

The value of Lemma 4.2.1 is that it gives us a criterion to check whether an open-net node is covered or not. A transition t of N is covered by an X_1-strategy of N if there is a state q, and the knowledge in q contains a marking m of N such that t is enabled at m. Analogously, a place p of N is covered by an X_1-strategy of N if there is a state q in $OG_{X_1}(N)$, and the knowledge in q contains a marking of N where p is marked. That way, it is easily possible to annotate each state q of $OG_{X_1}(N)$ with all places and transitions which are covered in q. This can be done *during* the calculation of the X_1-operating guideline.

Observe the importance of using the minimal simulation relation of $SA(S)$ by $OG_{X_1}(N)$ for the matching; otherwise, a state q_S of $SA(S)$ that is not reachable in the composition $N \oplus SA(S)$ could be related to a state q of $OG_{X_1}(N)$ and thus violate the result of Lemma 4.2.1.

Based on the knowledge of Bank in the X_1-operating guideline $OG_{X_1}(\mathsf{Bank})$ in Figure 4.5, we can derive the following sets of nodes of Bank that are covered in states q0–q3 of $OG_{X_1}(\mathsf{Bank})$:

$$\begin{aligned}
&\mathsf{q0}: && \{\mathsf{p0}, \mathsf{p1}, \mathsf{p2}, \mathsf{as}, \mathsf{req}, \mathsf{t0}, \mathsf{t1}\};\\
&\mathsf{q1}: && \{\mathsf{p1}\};\\
&\mathsf{q2}: && \{\mathsf{p0}, \mathsf{p1}, \mathsf{p2}, \mathsf{p3}, \mathsf{ap}, \mathsf{as}, \mathsf{req}, \mathsf{t0}, \mathsf{t1}, \mathsf{t2}\};\\
&\mathsf{q3}: && \{\mathsf{p2}\}.
\end{aligned}$$

In the following, we show how all $X_2(Y)$-strategies of an open net can be represented.

4.2.2. Extending X_1-operating guidelines with a global constraint

In this section, we extend X_1-operating guidelines by a global Boolean formula to represent all $X_2(Y)$-strategies of an open net N. We further present an algorithm for deciding whether an open net S matches with such an extended X_1-operating guideline.

Consider the open net Bank in Figure 4.3(a), and suppose we want to cover $Y = \{\mathsf{t2}, \mathsf{t3}\}$. Transition t2 is enabled when p2 and ap are marked. Hence, we look for knowledge values of $OG_{X_1}(\mathsf{Bank})$ (cf. Figure 4.5) that mark these two places. This is the case for the following three states of $OG_{X_1}(\mathsf{Bank})$: $[\mathsf{p2}, \mathsf{ap}, \mathsf{req}] \in k_{OG(\mathsf{Bank}),\mathsf{Bank}}(\mathsf{q2})$, $[\mathsf{p2}, \mathsf{ap}] \in k_{OG(\mathsf{Bank}),\mathsf{Bank}}(\mathsf{q6})$, and $[\mathsf{p2}, \mathsf{ap}, \mathsf{req}], [\mathsf{p2}, \mathsf{ap}, \mathsf{i}] \in k_{OG(\mathsf{Bank}),\mathsf{Bank}}(\mathsf{q9})$. Transition t3 is enabled when p4 is marked. This is the case for the following two states of $OG_{X_1}(\mathsf{Bank})$: $[\mathsf{p4}] \in k_{OG(\mathsf{Bank}),\mathsf{Bank}}(\mathsf{q7})$, and $[\mathsf{p4}, \mathsf{ap}] \in k_{OG(\mathsf{Bank}),\mathsf{Bank}}(\mathsf{q9})$. According to Lemma 4.2.1, Y is covered in the composition of an X_1-strategy S of Bank and Bank if $SA(S)$ has at least a state q_{t2} that is related to states q2, q6, or q9 in the minimal simulation relation ϱ of $SA(S)$ by $OG_{X_1}(\mathsf{Bank})$ (for covering t2), and it has a state q_{t3} that is related to q7 or q9 in ϱ (for covering t3).

This example illustrates that it is in general not possible to express the constraints for covering open-net nodes in the shape of local annotations in each state of the X_1-operating guideline. Consequently, the present concept of an annotated automaton fails at representing all $X_2(Y)$-strategies of N. To overcome this problem, we propose another representation that takes the non-locality of covering open-net nodes into account. To this end, we slightly refine the concept of an X_1-operating guideline.

Consider again the example. The X_1-operating guideline $OG_{X_1}(\mathsf{Bank})$ in Figure 4.5 represents all X_1-strategies of Bank, and every $X_2(Y)$-strategy of Bank is also an X_1-strategy of Bank. Hence, we have to restrict $OG_{X_1}(\mathsf{Bank})$ to $X_2(Y)$-strategies. This can be achieved by a *global constraint* specifying that, for every open-net node $y \in Y$ of Bank, at least one state q in $OG_{X_1}(\mathsf{Bank})$ with $m \in k_{OG(\mathsf{Bank}),\mathsf{Bank}}(q)$ must be present in the minimal simulation relation of an $X_2(Y)$-strategy by $OG_{X_1}(\mathsf{Bank})$ such that m marks/enables y. We can express this constraint as a Boolean formula χ.

We formalize annotated automata, extended with a global constraint, and define the matching relation between an open net and such an extended annotated automaton.

Definition 4.2.2 (extended annotated automaton).
Let $A^\phi = (Q, MC^I, MC^O, \delta, q_0, \phi)$ be an annotated automaton, and let $\chi \in \mathcal{BF}^Q$ be a Boolean formula with literals taken from the set Q. Then, $A^{\phi,\chi} = (A^\phi, \chi)$ is an *extended annotated automaton*, and χ is its *global constraint*. ⌟

As an example for a global constraint to $OG_{X_1}(\mathsf{Bank})$, consider $\chi \equiv (\mathsf{q2} \vee \mathsf{q6} \vee \mathsf{q9}) \wedge (\mathsf{q7} \vee \mathsf{q9})$. This formula is satisfied if and only if *true* is assigned to sufficiently many states to cover the set $Y = \{\mathsf{t2}, \mathsf{t3}\}$.

Extending an annotated automaton A^ϕ with a global constraint χ makes it necessary to define matching of an open net S with $A^{\phi,\chi}$. Open net S matches with an extended annotated automaton $A^{\phi,\chi}$ if it matches with A^ϕ and, in addition, satisfies χ.

Definition 4.2.3 (matching with extended annotated automaton).
Let $A^{\phi,\chi}$ be an extended annotated automaton. An open net S *matches* with $A^{\phi,\chi}$ iff

- S matches with A^ϕ using relation ϱ; and
- χ evaluates to *true* in the assignment $\gamma_{SA(S)} : Q_A \to \{\mathit{true}, \mathit{false}\}$, where $\gamma_{SA(S)}(q) = \mathit{true}$, for $q \in Q_A$ iff there is a state q_S in $SA(S)$ such that $(q_S, q) \in \varrho$. ⌟

Finally, we construct an extended annotated automaton for an open net N as a finite representation of the set $\mathit{Strat}_{X_2(Y)}(N)$ of all $X_2(Y)$-strategies of N, the $X_2(Y)$-operating guideline $OG_{X_2(Y)}(N)$ of N.

Definition 4.2.4 ($X_2(Y)$-operating guideline).
Let N be an open net and $OG_{X_1}(N)$ the X_1-operating guideline of N. Let $Y \subseteq P_N \cup T_N$. For a place $p \in P_N$ of N and a state $q \in Q$ of $OG_{X_1}(N)$, let $p \sim q$ iff there is an $m \in k_{OG(N),N}(q)$, where $m(p) > 0$. For a transition $t \in T_N$ of N and a state

$q \in Q$ of $OG_{X_1}(N)$, let $t \sim q$ iff there is an $m \in k_{OG(N),N}(q)$, where t is enabled. Then, the extended annotated automaton $OG_{X_2(Y)}(N) = (OG_{X_1}(N), \chi)$ with

$$\chi = \bigwedge_{y:y\in Y} \bigvee_{q:y\sim q} q$$

defines the $X_2(Y)$-*operating guideline of* N. ⌟

As a consequence of Lemma 4.2.1, we obtain the main result of this section; that is, $OG_{X_2(Y)}(N)$ represents all $X_2(Y)$-strategies of N.

Theorem 4.2.5 ($OG_{X_2(Y)}(N)$ represents all $X_2(Y)$-strategies of N). An open net S is an $X_2(Y)$-strategy of an open net N iff S matches with $OG_{X_2(Y)}(N)$. ⌟

Proof.
($\Rightarrow$): Let S be an $X_2(Y)$-strategy of N, and let $OG_{X_2(Y)}(N) = (OG_{X_1}(N), \chi)$ be the $X_2(Y)$-operating guideline of N. We show that S matches with $OG_{X_2(Y)}(N)$.

From $S \in \mathit{Strat}_{X_2(Y)}(N)$ and $\mathit{Strat}_{X_2(Y)}(N) \subseteq \mathit{Strat}_{X_1}(N)$, we conclude $S \in \mathit{Strat}_{X_1}(N)$. Thus, S matches with $OG_{X_1}(N)$ satisfying the first item of Definition 4.2.3. Furthermore, every $y \in Y$ of N is (by definition $X_2(Y)$-strategy) covered in $N \oplus S$. From Lemma 4.2.1 follows that, for all $y \in Y$, there is a state $q \in Q$ of $OG_{X_1}(N)$, a state q_S in $SA(S)$ of S, and a marking $m_N \in k_{OG(N),N}(q)$ with $(q_S, q) \in \varrho$, and y is marked in/enabled at m_N. Thus, for each disjunction of χ, $\gamma_{SA(S)}$ assigns *true* to at least one state $q \in Q$ of $OG_{X_2(Y)}(N)$. Consequently, $SA(S)$ satisfies χ and the second item of Definition 4.2.3 holds. Hence, we conclude from Definition 4.2.3 that S matches with $OG_{X_2(Y)}(N)$.

($\Leftarrow$): Let S match with $OG_{X_2(Y)}(N)$. We show that S is an $X_2(Y)$-strategy of N.

Because S matches with $OG_{X_2(Y)}(N)$, we know by Definition 4.2.3 that S matches with $OG_{X_1}(N)$ as well. Thus, S is an X_1-strategy of N. Furthermore, $SA(S)$ satisfies χ (follows also from Definition 4.2.3). So we conclude, for each disjunction of χ, $\gamma_{SA(S)}$ assigns *true* to at least one state $q \in Q$ of $OG_{X_2(Y)}(N)$. Hence, for all nodes $y \in Y$ of N, there is a state q with $y \sim q$. By Lemma 4.2.1, we conclude that all $y \in Y$ are covered in $N \oplus S$. Therefore S is an $X_2(Y)$-strategy of N. □

Theorem 4.2.5 proves that an $X_2(Y)$-operating guideline is indeed a finite representation of all $X_2(Y)$-strategies of an open net, and hence it justifies Definition 4.2.4.

The representation of all $X_2(Y)$-strategies of Bank with $Y = \{\mathsf{t2}, \mathsf{t3}\}$ is the $X_2(Y)$-operating guideline $OG_{X_2(Y)}(\mathsf{Bank}) = (OG_{X_1}(\mathsf{Bank}), \chi)$ with $\chi \equiv (\mathsf{q2} \vee \mathsf{q6} \vee \mathsf{q9}) \wedge (\mathsf{q7} \vee \mathsf{q9})$ as previously mentioned. If we consider the open net $\mathsf{Cust1}$ (which is an X_1-strategies of Bank) again, then we get that $\mathsf{Cust1}$ does not match with $OG_{X_2(Y)}(\mathsf{Bank})$, because it does not satisfy the global constraint. More precisely, there is no state in $SA(\mathsf{Cust1})$ that is related to any of the states q2, q6, or q9 as it can be observed from Figure 4.5. Hence, Cust1 is not an $X_2(Y)$-strategy of Bank.

4.2.3. Representing strategies in case of deadlock freedom and quasi-liveness

In the following, we show how the notion of an extended annotated automaton can be used to represent all open nets S of an open net N such that the composition $N \oplus S$ is deadlock-free and quasi-live. Quasi-liveness ensures that every transition of N *and* every transition of S is covered in $N \oplus S$. We refer to such an open net S as an X_2-strategy of N.

An X_2-strategy S of N is an X_1-strategy of N (ensuring deadlock freedom in the composition). Moreover, it is an $X_2(Y)$-strategy of N, for $Y = T_N$ (ensuring that each transition of N is covered in the composition). From the set of $X_2(Y)$-strategies of N we have to rule out all open nets that have a transition that is not covered in the composition. This can be achieved by adjusting the matching of S with the $X_2(Y)$-operating guideline of N: For every state q_S of $SA(S)$, there must exist a state q of the $X_2(Y)$-operating guideline of N in the minimal simulation relation and q is not the empty state.

We continue with the definition of the X_2-operating guideline of an open net N and present afterwards matching with such an annotated automaton.

Definition 4.2.6 (X_2-operating guideline).
Let N be an open net with $Y = T_N$, and let $OG_{X_1}(N)$ be the X_1-operating guideline of N. The extended annotated automaton $OG_{X_{2(Y)}}(N)$ defines the *X_2-operating guideline $OG_{X_2}(N)$ of N.* ⌟

Definition 4.2.7 (matching with X_2-operating guideline).
For an open net N let $OG_{X_2}(N) = A^{\phi,\chi}$ be the X_2-operating guideline of N. An open net S *matches* with $OG_{X_2}(N)$ iff

- S matches with $A^{\phi,\chi}$ using relation ϱ; and
- for all states q_S of $SA(S)$, there is a $q \in Q_A$ with $(q_S, q) \in \varrho$ and q is not the empty state $q_\emptyset$. ⌟

Existence of q_S in the minimal simulation relation ϱ ensures that this state is reachable from the initial state. Further, existence of a state $q \neq q_\emptyset$ with $(q_S, q) \in \varrho$ guarantees reachability of q_S in the composition with N.

From Theorem 4.2.5 we conclude that the X_2-operating guideline of N represents all X_2-strategies of N.

Corollary 4.2.8 ($OG_{X_2}(N)$ represents all X_2-strategies of N).
An open net S is an X_2-strategy of an open net N iff S matches with $OG_{X_2}(N)$.⌟

The representation of all X_2-strategies of Bank is the X_2-operating guideline $OG_{X_2}(\mathsf{Bank}) = (OG_{X_1}(\mathsf{Bank}), \chi)$ with

$$\chi \equiv (\mathsf{q0} \vee \mathsf{q2}) \wedge (\mathsf{q0} \vee \mathsf{q2}) \wedge (\mathsf{q2} \vee \mathsf{q6} \vee \mathsf{q9}) \wedge (\mathsf{q7} \vee \mathsf{q9}) \wedge (\mathsf{q7} \vee \mathsf{q9}) \quad ,$$

which is equivalent to

$$\chi \equiv (\mathsf{q0} \vee \mathsf{q2}) \wedge (\mathsf{q2} \vee \mathsf{q6} \vee \mathsf{q9}) \wedge (\mathsf{q7} \vee \mathsf{q9}) \quad .$$

If we consider the open net Cust1 (which is an X_1-strategies of Bank) again, then we get that Cust1 does not match with $OG_{X_2}(\mathsf{Bank})$, because it does not satisfy the global constraint. More precisely, there is no state in $SA(\mathsf{Cust1})$ that is related to any of the states q2, q6, or q9 as it can be observed from Figure 4.5. Hence, Cust1 is not an X_2-strategy of Bank.

4.2.4. Experimental results and discussion

We use a prototypical implementation of the algorithm for constructing the $X_2(Y)$-operating guideline of an open net N to experiment with a number of real-life service models. Furthermore, we compare X_1-operating guidelines and $X_2(Y)$-operating guidelines and discuss some related work.

Experimental results

The results presented in this section have been implemented in the service analysis tool Fiona. Thus, Fiona can also be used to calculate the $X_2(Y)$-operating guideline (or the X_2-operating guideline) of an open net and to check whether an open net matches with an $X_2(Y)$-operating guideline.

For the example services from Section 4.1.4, we calculated the X_2-operating guidelines. Table 4.2 provides the results of this experiment. As Table 4.1 in Section 4.1.4, it shows the size of the open nets (i. e., number of places, transitions, and interface places), the size of its corresponding service automaton (i. e., number of states and transitions), structural properties of the open nets (i. e., concurrency, cycles), the size of the X_2-operating guidelines for responsiveness (i. e., number of states and transitions), the size of the global constraint (number of disjunctions—the global constraint is a conjunction of disjunctions—and number of literals appearing in the constraint), and the time to calculate the X_1-operating guideline and the X_2-operating guideline. In comparison to Table 4.1 in Section 4.1.4, the information about the global constraint has been added, and the information about the size of the most permissive X_1-strategy is not shown again.

The corresponding open net of ‘Online Shop 1’ has 61 places, 58 transitions, and 7 interface places. The state space of its inner subnet has 205 states and 463 edges. ‘Online Shop 1’ contains concurrency, but it is acyclic. The X_2-operating guideline has 13 states and 50 transitions, and the global constraint consists of 3 disjunctions and 3 literals. The time for computing the X_1-operating guideline is 2 seconds. Computing the X_2-operating guideline takes also 2 seconds.

Table 4.2.: Calculation of $OG_{X_2}(N)$ with Fiona. All experiments were obtained on an UltraSPARC III processor with 900MHz and 4 GB RAM running Solaris 10.

	N			$SA(N)$		Props		$MP^R_{X_1}(N)$		Constraint		t_{X_1}	t_{X_2}
Service	$\|P\|$	$\|T\|$	$\|P^{Int}\|$	$\|Q\|$	$\|\delta\|$	$\|\|$	$\circlearrowleft$	$\|Q\|$	$\|\delta\|$	$\|\vee\|$	$\|\chi\|$	mm:ss	mm:ss
Loan Approval	38	25	6	26	33	✓	-	8	30	3	3	00:00	00:01
Purchase Order	22	7	10	12	15	✓	-	169	1,182	5	135	00:02	00:02
Olive Oil Ordering	12	6	6	6	6	-	-	17	67	3	9	00:00	00:00
Travel Service 1	15	5	8	7	7	✓	-	57	300	5	68	00:00	00:00
Travel Service 2	22	9	12	10	11	✓	-	289	2,252	11	753	00:10	00:10
Online Shop 1	61	58	7	205	463	✓	-	13	50	3	3	00:02	00:02
Online Shop 2	72	69	8	308	744	✓	-	8	35	2	2	00:04	00:04
Beverage Machine	12	8	7	5	8	-	✓	12	54	3	7	00:00	00:00
Philosophers #3	22	10	6	46	70	✓	✓	68	243	4	95	00:01	00:01
Philosophers #5	36	16	10	574	1,476	✓	✓	1,433	8,390	6	2,805	01:25	01:57
SMPT Protocol	123	133	12	8,345	34,941	-	✓	1,217	8,408	8	1,583	47:25	47:30
Registration	127	88	6	1,057	2,927	✓	-	8	30	3	3	00:10	00:10
Process 1	39	27	20	97,511	468,072	✓	-	897	13,548	5	1,613	00:07	00:09
Process 2	29	18	13	244	758	✓	-	1,608	16,445	7	3,163	00:07	00:12
Process 3	26	15	13	992	3,744	✓	-	237	1,437	7	197	07:57	07:56
Process 4	29	14	19	160	494	✓	-	6,821	154,713	7	8,297	06:24	06:24

The experimental results illustrate that the X_2-operating guidelines of these services were calculated in almost the same time than the corresponding X_1-operating guidelines (cf. Table 4.1). Furthermore, the size of the global constraints is typically still tractable.

Discussion

In the following, we compare X_1-operating guidelines and $X_2(Y)$-operating guidelines. We further discuss some complexity issues.

Let N be an open net, and let $Y \subseteq P_N \cup T_N$ be the set of nodes of N to be covered. As usual, the X_1-operating guideline of N is denoted by $OG_{X_1}(N)$ and the $X_2(Y)$-operating guideline with a global constraint χ by $OG_{X_2(Y)}(N)$.

Comparing an X_1-operating guideline and an $X_2(Y)$-operating guideline of N, we identify that both representations have the same underlying automaton—the automaton of the most permissive X_1-strategy of N. The reason is that each $X_2(Y)$-strategy of N is also an X_1-strategy of N. Furthermore, if the most permissive X_1-strategy of N is not an $X_2(Y)$-strategy of N, then the set of $X_2(Y)$-strategies is empty.

The X_1-operating guideline $OG_{X_1}(N)$ can be computed in time $\mathcal{O}(|N| \cdot |MP_{X_1}(N)|)$. For $OG_{X_2(Y)}(N)$ the time complexity does not change. All information necessary for annotating the states of $OG_{X_1}(N)$ with the nodes of N and setting up the global constraint have to be computed for $OG_{X_1}(N)$ anyway. To increase efficiency, it is sufficient to annotate each state $q \in Q$ of $OG_{X_1}(N)$ only with nodes of the set Y.

The space complexity of $OG_{X_1}(N)$ is $\mathcal{O}(|N| \cdot |OG_{X_1}(N)|)$. If we compute $OG_{X_2(Y)}(N)$, then this complexity increases because of χ. The global constraint is a conjunction of at most $|Y|$ disjunctions, where each disjunction may consist of at most $|Q|$ literals. Hence, the size of the global constraint is at most $\mathcal{O}(|Y| \cdot |Q|)$, and we have an overall space complexity of $\mathcal{O}(|N| \cdot |OG_{X_1}(N)| + |Y| \cdot |Q|)$. The experimental results suggests that the size of χ will be much smaller in practice.

Matching an open net S with $OG_{X_1}(N)$ has a time complexity $\mathcal{O}(|OG_{X_1}(N)| \cdot |S|)$. If we check matching of S with $OG_{X_2(Y)}(N)$, we additionally have to check whether the global constraint is satisfied by the assignment $\gamma_{SA(S)}$. This can be done in linear time with respect to the size of the constraint χ. So the overall complexity is $\mathcal{O}(|OG_{X_1}(N)| \cdot |S| + |Y| \cdot |Q|)$.

As the space complexity and the matching complexity for the $X_2(Y)$-operating guidelines only marginally increase in comparison with X_1-operating guidelines and as a consequence of our case study, we conclude that this novel notion is as well-suited as X_1-operating guidelines.

It is possible to reduce the size of the global constraint by taking into account the structure of the $X_2(Y)$-operating guideline $OG_{X_2(Y)}(N)$. For example, if an open net cannot assign *true* to a state q of $OG_{X_2(Y)}(N)$ in the matching, it cannot assign true to any other state q' of $OG_{X_2(Y)}(N)$ that is reachable only from q.

Recently, an approach has been published by Kaschner and Wolf [KW09] that uses the notion of an extended annotated automaton to represent the *complement* of all X_1-strategies for a given open net N. The authors present an algorithm to construct, given the X_1-operating guideline of N, an extended annotated automaton that represents all open nets S that are *not* an X_1-strategy of N. As a substantial difference to our notion of an extended annotated automaton, [KW09] use a propositional logic with *negation* to define Boolean formulae. The additional application of extended annotated automata justifies their usefulness as a representation of sets of open nets.

The notion of *relaxed soundness* [DA04] for workflow nets ensures that for each transition t of a workflow net, there is a run that enables t and can be carried forward to the final state. Our approach enables us to compute all X_2-strategies S of an open net N such that the composition $N \oplus S$ is deadlock-free, and every transition of the composition can be enabled in at least a single run. So on the one hand, our approach is stricter than relaxed soundness, as $N \oplus S$ cannot deadlock. Deadlocks are not excluded by the notion of relaxed soundness. On the other hand, relaxed soundness guarantees that, for each transition t, there exists a run to a final state. In our approach, this is not necessarily the case, because deadlock freedom does not exclude livelocks; thus, we cannot ensure that a run covering t reaches a final state.

4.3. Representing strategies in case of weak termination

In this section, we introduce for any open net N a finite representation of all open nets S such that the composition of N and S weakly terminates. We refer to S as an X_3-strategy of N. Strengthen the termination criterion from deadlock freedom to weak termination requires a representation that is different from (extended) annotated automata. In the next subsection, we motivate the need for another finite representation and outline this section.

The results of this section have been published in [WSOD09].

4.3.1. Motivation for another representation

In this subsection, we show that, in general, (extended) annotated automata cannot represent all X_3-strategies of an open net. Afterwards, we explain our new approach and outline the remainder of this section.

Limitations of extended annotated automata

In order that the X_1-operating guideline $OG_{X_1}(\mathsf{Bank})$ in Figure 4.5 can serve as a finite representation of all X_3-strategies of Bank, it must exclude all open nets S that may cause a livelock in the composition with Bank. We distinguish livelocks that are *local* to Bank or to S (i. e., the inner subnet of Bank or of S contains a livelock) and livelocks that are caused by the *interaction* of Bank and S (i. e., the livelock occurs only in the composition $\mathsf{Bank} \oplus S$).

Livelocks that are local to one of the open nets do not cause a problem, as they can be detected by standard state-space exploration algorithms [CGP00].

In contrast, livelocks that are caused by the interaction of two open nets pose a challenge.

As an illustration, consider the composition Bank $\oplus$ Cust1 in Figure 4.7. Although $inner$(Bank) and $inner$(Cust1) do not contain a livelock, the composition Bank $\oplus$ SA(Cust1) may livelock. After Bank has sent a request req and Cust1 has received this message, the composition enters a cycle which cannot be left. The reason can be observed on the X_1-operating guideline OG_{X_1}(Bank) in Figure 4.5. The cycle corresponds to states q3 and q7 in OG_{X_1}(Bank). It cannot be left by SA(Cust1), because transition (q3, !ap, q6) of OG_{X_1}(Bank) is not present in the minimal simulation relation of SA(Cust1) by OG_{X_1}(Bank). That means, Cust1 can send only i, but not ap.

To identify that Cust1 is not an X_3-strategy of Bank, we need to encode that a service automaton can always leave this cycle. This example illustrates that it is in general not possible to express such a constraint for leaving a cycle in the shape of local annotations in each state of an X_1-operating guideline. The notion of a global constraint, as introduced in Section 4.2.2, is also not suitable: With the help of a global constraint, we can specify that a final state *can* be reached. To exclude livelocks, we need, however, to specify that a final state can *always* be reached. Hence, we need another representation.

Illustration of the novel representation

We have motivated that it is, in general, not possible to represent all X_3-strategies of an open net N as an (extended) annotated automaton in a straightforward manner. So, we need another representation. As in case of X_1-operating guidelines, the representation of all X_3-strategies of N should hide the internals of N, because it might be necessary to publish this representation. Hence, open net N or a reduced version of N is not a well-suited candidate. For this reason, we follow a different approach.

The approach starts with the construction of the most permissive X_1-strategy $MP_{X_1}(N)$ of N. From this service automaton, we construct the most permissive X_3-strategy $MP_{X_3}(N)$ of N (Section 4.3.2) having the following two properties: The composition $N \oplus MP_{X_3}(N)$ contains all markings of N that are reachable with any X_3-strategy S of N, and there is a minimal simulation relation of any X_3-strategy of N by $MP_{X_3}(N)$. We make use of these facts and partition the LTS of the composition $N \oplus MP_{X_3}(N)$ into *fragments* (Section 4.3.3). A fragment is that part of the composition that takes place between two subsequent transitions of $MP_{X_3}(N)$. As an example, Figure 4.8 shows the LTS of the composition Bank $\oplus$ MP_{X_3}(Bank) and its partitioning into fragments.

Given an open net S and the set of fragments of N, we glue fragments to the actual LTS of $N \oplus S$. Here, the minimal simulation relation of $SA(S)$ by $MP_{X_3}(N)$ determines the way in which fragments are glued. The LTS can then be analyzed for weak termination using state-of-the-art model-checking tools. That way, we can decide whether S is an X_3-strategy of N.

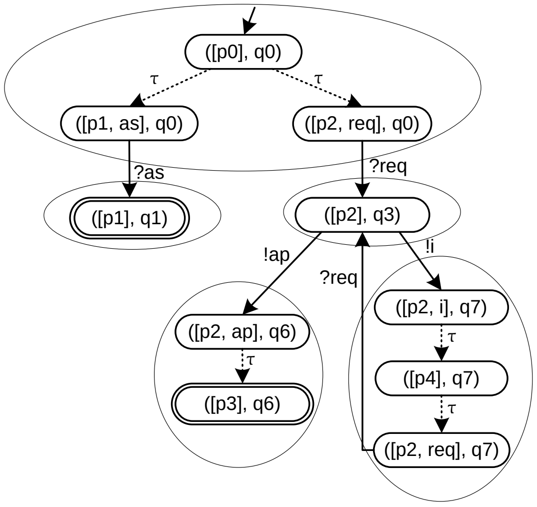


Figure 4.8.: An LTS of Bank $\oplus$ MP_{X_3}(Bank). Dotted arcs denote transitions of Bank; solid arcs denote transitions of MP_{X_3}(Bank). Each ellipse denotes a fragment of Bank.

To speed up the approach and to hide the internals of N, we apply reduction rules to condense the state spaces of the fragments a posteriori. As an advantage, the outcome of this reduction is then available in *every* LTS that uses the reduced fragments (Section 4.3.4). We also present a procedure to detect already *during* the construction of the LTS of $N \oplus S$ if this LTS may deadlock (Section 4.3.5). In such a case, the construction of the LTS is stopped immediately.

Finally, Section 4.3.6 presents some experimental results based on a prototypical implementation. We also highlight some complexity results and discuss the achieved results.

4.3.2. Construction of the most permissive X_3-strategy

An algorithm for constructing the most permissive X_3-strategy $MP_{X_3}(N)$ of an open net N has been developed in [Wol09]. As in [Wol09] final states are restricted to states where an open net can only receive a message, we have to slightly adapt this work.

We start with the construction of the most permissive X_1-strategy $MP_{X_1}(N)$ of N according to Definition 4.1.10. Service automaton $MP_{X_1}(N)$ has to be adjusted such that $N \oplus MP_{X_1}(N)$ is weakly terminating. Weak termination (cf. Definition 2.6.2) requires that, from every state, it is always possible to reach a final state. Hence, we remove all states q of $MP_{X_1}(N)$, where a state (m, q) in $N \oplus MP_{X_1}(N)$ exists such that no final marking is reachable from (m, q).

Definition 4.3.1 (transformation of $MP_{X_1}(N)$ to an X_3-strategy).
Let $MP_{X_1}(N)$ be the most permissive X_1-strategy of an open net N. Let $(MP_{X_1}(N))^i$, $i = 0, 1, \ldots$ be a sequence of service automata inductively defined as follows.

Basis : $(MP_{X_1}(N))^0 = (Q^0, MC^I, MC^O, \delta^0, q_0, \Omega^0)$
Step : $(MP_{X_1}(N))^{i+1} = (Q^{i+1}, MC^I, MC^O, \delta^{i+1}, q_0, \Omega^{i+1})$ with
– $Q^{i+1} = Q^i \setminus \{q \in Q^i \mid \exists\ (m, q) \in N \oplus (MP_{X_1}(N))^i :$
$\forall\ (m', q'), q' \in \Omega^i, m' \in \Omega_N : (m, q) \overset{*}{\not\rightarrow} (m', q')\}$
– $\delta^{i+1} = \delta^i \cap (Q^{i+1} \times \mathcal{MC}^+ \times Q^{i+1})$
– $\Omega^{i+1} = \Omega^i \cap Q^{i+1}$.

Let j be the smallest number with $(MP_{X_1}(N))^j = (MP_{X_1}(N))^{j+1}$. If $q_0 \in Q^j$, then $(MP_{X_1}(N))^j$ is the *most permissive X_3-strategy $MP_{X_3}(N)$* of N; else $MP_{X_3}(N)$ is not defined. ⌟

The algorithm iteratively removes all states of the most permissive X_1-strategy of N from which a final marking in the composition with N cannot be reached. Accordingly, the transition relation and the final states have to be adjusted to the new state set.

Consider the most permissive X_1-strategy of Bank in Figure 4.5. From markings [p1, i] and [p1, ap] in states q4 and q5, respectively, a final state is not reachable. Hence, we remove both states and therefore also its successor state q8. As from marking [p1, ap, as] in state q2 and marking [p3, i] in state q9 a final state cannot be reached either, both states are removed. Figure 4.9 illustrates the most permissive X_3-strategy of Bank.

The most permissive X_1-strategy is finite by Lemma 4.1.12. Hence, the construction algorithm of Definition 4.3.1 always terminates. Correctness of this construction can be observed easily, as Definition 4.3.1 directly implements the definition of weak termination. So if the resulting most permissive X_3-strategy of N contains at least one state, we conclude that N is X_3-controllable (cf. Definition 2.6.8). This is justified by the following theorem, which is the main result of this subsection.

Theorem 4.3.2 (transformation preserves all X_3-strategies).
Let N be an open net, and let $MP_{X_3}(N)$ be its most permissive X_3-strategy being constructed according to Definition 4.3.1. Open net N is X_3-controllable iff $Q_{MP} \neq \emptyset$. ⌟

From Theorem 4.3.2 and Lemma 4.1.12, we conclude that $MP_{X_3}(N)$ simulates each X_3-strategy of N.

Corollary 4.3.3 ($MP_{X_3}(N)$ simulates all X_3-strategies of N).
Let N be an X_3-controllable open net. The most permissive X_3-strategy of N simulates every X_3-strategy S of N. ⌟

With the help of $MP_{X_3}(N)$, we can now define a finite representation of all X_3-strategies of an open net.

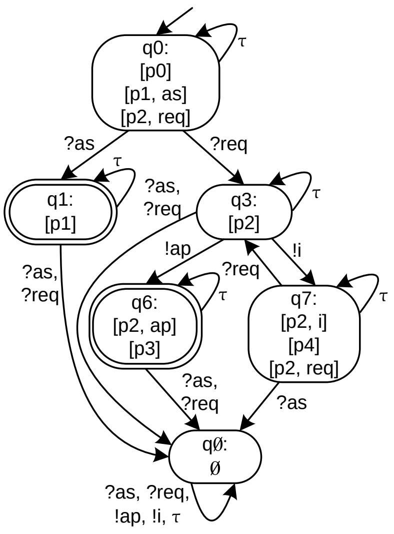


Figure 4.9.: The most permissive X_3-strategy $MP_{X_3}(\mathsf{Bank})$ of Bank (annotated with the knowledge of N).

4.3.3. A finite generator set for constructing composed systems

We aim at computing a finite set of state-space fragments such that, for every X_3-strategy S of N, the LTS of the composition $N \oplus S$ can be constructed from these fragments. The core idea is to make use of the fact that the most permissive X_3-strategy $MP_{X_3}(N)$ of N simulates every X_3-strategy of N (cf. Corollary 4.3.3). So if $SA(S)$ is simulated by $MP_{X_3}(N)$, its composition $N \oplus SA(S)$ must follow those patterns which are already present in $N \oplus MP_{X_3}(N)$. To this end, we decompose the state space of $N \oplus MP_{X_3}(N)$ into fragments (cf. Figure 4.8). These fragments can then be glued to the actual composition $N \oplus SA(S)$, where the simulation relation of $SA(S)$ by $MP_{X_3}(N)$ determines the way in which fragments are glued.

We formally define fragments based on LTSs and show how to compute fragments for an open net N. Afterwards, we present how an LTS of $N \oplus S$ can be constructed, given the fragments of N and an open net S.

Fragments and connections

Given N and $MP_{X_3}(N)$, the state space of the composition $N \oplus MP_{X_3}(N)$ yields an LTS TS that consists of transitions of N and $MP_{X_3}(N)$. We use this property to derive a canonical decomposition of TS. Suppose we remove all transitions of $MP_{X_3}(N)$. This yields a set of unconnected subgraphs of TS. Each subgraph consists of states and transitions, where each transition corresponds to a transition of N. Such a subgraph is a *fragment*, and a transition of $MP_{X_3}(N)$ is a *connection* and connects two states of different fragments. Figure 4.8 illustrates this idea.

The set of fragments and connections can be seen as a decomposition of TS into the reachable states of $MP_{X_3}(N)$ in TS. Each fragment corresponds to a state q of $MP_{X_3}(N)$, and the states of this fragment are the Cartesian product of q with the knowledge (cf. Definition 4.1.7) that $MP_{X_3}(N)$ has of N in state q. Besides the knowledge (i. e., a set of markings of N) in state q, a fragment also contains the transition relation of the markings inside the knowledge. This is, in fact, the main difference to X_1-operating guidelines. We assign to each state of an X_1-operating guideline the knowledge of N, rather than the transition relation of the markings within the knowledge.

According to our approach, we want to compose state spaces from fragments. To this end, we formally define fragments and connections in terms of LTSs.

Definition 4.3.4 (state-space fragment).
A (state-space) *fragment* $F = (V, \Sigma, E, \Omega)$ is an LTS without initial state. ⌟

To distinguish between service automata and fragments, we denote the set of states of a fragment by V (instead of Q) and the transition relation by E (instead of δ). 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 LTS of the composition $N \oplus S$ from fragments of N. The initial state will be the initial state of $N \oplus MP_{X_3}(N)$.

When composing a state space from fragments, it may happen that we need several copies of one and the same fragment. The reason is that different states of $SA(S)$ may be related to the same state of $MP_{X_3}(N)$ and hence to the same fragment. For instance, this happens if $SA(S)$ executes an internal transition, because then the knowledge of N does not change. To this end, we introduce fragment *instances*. Let some fixed set FI denote the name space of all fragment instances.

Definition 4.3.5 (fragment instance).
Let $n \in FI$. An *instance* $F(n)$ of a fragment F is built by renaming the constituents as follows: $v \mapsto (v, n)$, $e = (v_1, x, v_2) \mapsto ((v_1, n), x, (v_2, n))$, for all $v \in V_F$, $e \in E_F$. ⌟

Fragment instances are fragments again. A fragment instance contains the label of the fragment and the label of the fragment instance. Later on, the label of a fragment instance refers to a state of $SA(S)$.

For gluing fragments, more precisely, for connecting states of different fragment (instances), we use the concept of *connections*.

Definition 4.3.6 (connection, connection instance).
A *connection* C between fragments $F_1 = (V_1, \Sigma, E_1, \Omega_1)$ and $F_2 = (V_2, \Sigma, E_2, \Omega_2)$ is defined as $C \subseteq V_1 \times (\Sigma \cup \{\tau\}) \times V_2$. 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 $m, n \in FI$. ⌟

If C is a connection between fragments F_1 and F_2, then $C(m, n)$ is a connection between fragment instances $F_1(m)$ and $F_2(n)$.

Given a set of fragments and a set of connections, we can build an LTS by connecting states of different fragments according to the connections.

Definition 4.3.7 (transition system of fragments).
A set $F_1, \dots, F_n$ of fragments with alphabet Σ and a set $C_1, \dots, C_m$ of connections define the *labeled transition system* $TS = (V, \Sigma, E)$ where

- $V = \bigcup_{k=1}^{n} V_{F_k}$; and
- $E = \bigcup_{i=1}^{n} E_{F_i} \cup \bigcup_{j=1}^{m} C_j$. ⌟

We do not define initial and final states of such an LTS, but we define these states later when we consider fragments for a given open net.

Next, we show how to define the set of fragments and connections for a given open net N.

Fragments and connections for a given open net

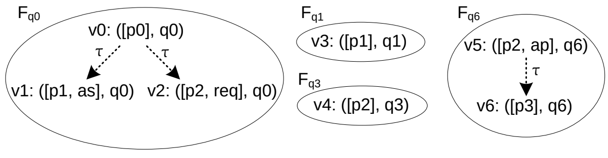
Given an open net N and its most permissive X_3-strategy $MP_{X_3}(N)$, we have all the ingredients to construct the sets $\mathcal{F}(N)$ of fragments and $\mathcal{C}(N)$ of connections for N. Due to finiteness of N and $MP_{X_3}(N)$, these sets can be indeed computed. With the sets $\mathcal{F}(N)$ and $\mathcal{C}(N)$ we provide those ingredients from which we can construct composed systems involving N.

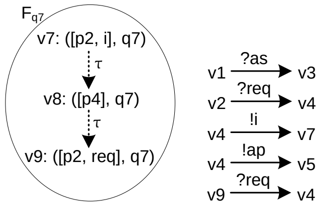
First, we construct the reachability graph of $N \oplus MP_{X_3}(N)$. Thereby, each transition of $MP_{X_3}(N)$ keeps its label. As an example, the reachability graph of $N \oplus MP_{X_3}(\mathsf{Bank})$ is illustrated in Figure 4.8. The reachability graph of $N \oplus MP_{X_3}(N)$ can be decomposed into the fragments and connections of N.

Definition 4.3.8 ($\mathcal{F}(N)$, $\mathcal{C}(N)$).
For an open net N, let $MP_{X_3}(N) = (Q, MC^I, MC^O, \delta, q_0, \Omega)$ be the most permissive X_3-strategy of N, and let $R = (Q_R, \Sigma_R, \delta_R, q_{0R})$ be the reachability graph of $N \oplus MP_{X_3}(N)$. Define $\mathcal{F}(N) = \{F_q \mid q \in Q\}$ and $\mathcal{C}(N) = \{C_{(q,x,q')} \mid (q, x, q') \in \delta\}$, where

- $V_{F_q} = \{(m, q) \mid (m, q) \in Q_R\}$;
- $E_{F_q} = \delta_R \cap (V_{F_q} \times \{\tau\} \times V_{F_q})$;
- $\Sigma_{F_q} = MC^I \cup MC^O$;
- $\Omega_{F_q} = \{(m, q) \in V_{F_q} \mid m \in \Omega_N \wedge q \in \Omega\}$; and
- $C_{(q,x,q')} = \delta_R \cap (V_{F_q} \times \{x\} \times V_{F_{q'}}) \wedge x \in MC^I \cup MC^O \cup \{\tau\}$. ⌟

For each state $q \in Q$ in the most permissive X_3-strategy $MP_{X_3}(N)$, there is a fragment F_q. Fragment F_q describes the behavior of N while $MP_{X_3}(N)$ is in state q. This yields the set V_{F_q} of states and E_{F_q} of transitions of F_q. Thereby,





(a) $\mathcal{F}$(Bank) (b) $\mathcal{C}$(Bank)

Figure 4.10.: (a) The sets of all fragments (except for the empty state $q_\emptyset$) and (b) all connections (except for the τ-loops) of the online bank Bank.

each state of fragment F_q consists of markings of N in q. Each transition of N is internal in the composition, and hence it is labeled with τ. The alphabet of a each fragment is equal to the alphabet of $MP_{X_3}(N)$. Final states of a fragment are those states, where both, N and $MP_{X_3}(N)$, are in a final state. A connection $C_{(q,x,q')}$ describes a set of transitions in the composed system that originates from a single transition (q, x, q') in $MP_{X_3}(N)$. Note that $\mathcal{F}(N)$ contains a fragment $F_{q\emptyset}$. However, $F_{q\emptyset}$ is unconnected and does not contain any state, because state $q_\emptyset$ is not reachable in R.

The fragments and the connections of the online bank Bank are shown in Figure 4.10. For each of the five reachable states q0, q1, q3, q6, and q7 of the most permissive X_3-strategy MP_{X_3}(Bank) (cf. Figure 4.9), there is one fragment in Figure 4.10(a). For instance, the fragment for the initial state q0 is defined as $\mathsf{F_{q0}} = (\{\mathsf{v0}, \mathsf{v1}, \mathsf{v2}\}, \{\mathsf{?as}, \mathsf{?req}, \mathsf{!ap}, \mathsf{!i}, \tau\}, \{(\mathsf{v0}, \tau, \mathsf{v1}), (\mathsf{v0}, \tau, \mathsf{v2})\}, \emptyset)$. Connection $\mathsf{C_{(v1,?as,v3)}}$ corresponds to a transition $\mathsf{v1} \xrightarrow{\mathsf{?as}} \mathsf{v3}$. Thereby v0 relabels the state ([p0], q0); v1 relabels the state ([p1, as], q0), etc.

Next, we show how we can construct an LTS of $N \oplus S$ given an open net S and the set of fragments $\mathcal{F}(N)$ and connections $\mathcal{C}(N)$ of N.

Composing a labeled transition system from fragments

Throughout this section, fix an open net N. Let $MP_{X_3}(N)$ be the most permissive X_3-strategy of N that has been used for constructing $\mathcal{F}(N)$ and $\mathcal{C}(N)$.

By Corollary 4.3.3, a minimal simulation relation ϱ of a service automaton $SA(S)$ by $MP_{X_3}(N)$ is a necessary condition for an open net S being an X_3-strategy of N. That is, if $MP_{X_3}(N)$ does not simulate $SA(S)$, $N \oplus S$ is not weakly terminating, and hence there is no use in constructing an LTS TS from $SA(S)$, $\mathcal{F}(N)$, and $\mathcal{C}(N)$ that reflects the LTS of $N \oplus S$. Thus, we may assume existence of ϱ when constructing TS, because once ϱ is violated, we can immediately stop the construction of TS. The existence of ϱ is actually checked during the construction of TS by relating states of $SA(S)$ to fragments of $\mathcal{F}(N)$.

We build the LTS TS by gluing fragments and connections of N. For constructing TS, we have to define its set $\mathcal{F}(TS)$ of fragments and $\mathcal{C}(TS)$ of connections. We add, for each $(q_S, q) \in \varrho$, a fragment F_q to the set $\mathcal{F}(TS)$. More precisely, as q_S might not be the only state of $SA(S)$ that is related to q, we add a fragment instance $F_q(q_S)$. Accordingly, the set $\mathcal{C}(TS)$ is determined by the connections of N and the fragments F_q of $\mathcal{F}(TS)$ where state q is used in ϱ.

Definition 4.3.9 (construction of *TS*).
Let ϱ be a minimal simulation relation of $SA(S)$ by $MP_{X_3}(N)$. Compose the LTS TS from the following fragments and connections:

- $\mathcal{F}(TS) = \{F_q(q_S) \mid (q_S, q) \in \varrho\}$;
- $\mathcal{C}(TS) = \{C_{(q,x,q')}(q_S, q'_S) \mid \exists x : (q_S, q), (q'_S, q') \in \varrho \wedge (q_S, x, q'_S) \in \delta_{SA(S)} \wedge (q, x, q') \in \delta_{MP}\}$.

Let the initial state of TS be the initial state of $N \oplus MP_{X_3}(N)$ in $F_{q_0}(q_{0S})$. Let the set of final states of TS consist of all final states occurring in those fragments $F_q(q_S)$ where $q_S \in \Omega_{SA(S)}$. ⌟

Fragment instance $F_{q_0}(q_{0S})$ is definitely contained in $\mathcal{F}(TS)$ and contains an instance of the initial state of $N \oplus MP_{X_3}(N)$; otherwise, there would be no minimal simulation relation of $SA(S)$ by $MP_{X_3}(N)$. We use this particular state as the initial state of TS.

In our example, we construct the state space of Bank⊕Cust1 from the fragments of Bank (cf. Figure 4.10) and SA(Cust1) (cf. Figure 4.4). We have $(\mathsf{s0}, \mathsf{q0})$, $(\mathsf{s1}, \mathsf{q0}) \in \varrho$. Hence, we add two instances of $\mathsf{F_{q0}}$, $\mathsf{F_{q0}(s0)}$ and $\mathsf{F_{q0}(s1)}$, to $\mathcal{F}(TS)$. As transition $(\mathsf{s0}, \tau, \mathsf{s1}) \in \delta_{SA(\mathsf{Cust1})}$, we add connection $\mathsf{C}_{(\mathsf{q0},\tau,\mathsf{q0})}(\mathsf{q0}, \mathsf{q1})$ to $\mathcal{C}(TS)$. Next, we add fragment $\mathsf{F_{q1}(s2)}$ to $\mathcal{F}(TS)$, because $(\mathsf{s2}, \mathsf{q1}) \in \varrho$. From transition $(\mathsf{s1}, ?\mathsf{as}, \mathsf{s2}) \in \delta_{SA(\mathsf{Cust1})}$ and $(\mathsf{q0}, ?\mathsf{as}, \mathsf{q1}) \in \delta_{MP}$, we conclude that connection $\mathsf{C}_{(\mathsf{q0},?\mathsf{as},\mathsf{q1})}(\mathsf{s1}, \mathsf{s2})$ has to be added to $\mathcal{C}(TS)$, and so on. Figure 4.11 shows the resulting transition system TS. Note that $(\mathsf{s4}, \mathsf{q7}) \in \varrho$ and presence of transition $(\mathsf{s4}, ?\mathsf{as}, \mathsf{s2}) \in \delta_{SA(\mathsf{Cust1})}$ (cf. Figure 4.4) yields $(\mathsf{s2}, \mathsf{q}\emptyset) \in \varrho$. However, as $\mathsf{q}\emptyset$ is not reachable in MP_{X_3}(Bank), there is no ?as-labeled connection leaving $\mathsf{F_{q7}(s4)}$. TS contains a livelock, because $\mathsf{F_{q3}(s3)}$ and $\mathsf{F_{q7}(s4)}$ do not have a final state. Thus, Cust1 is not an X_3-strategy of Bank.

As the main result of this subsection, we prove that the constructed LTS TS indeed reflects the state space of $N \oplus S$; that is, TS and the LTS of $N \oplus S$ are bisimilar. Intuitively, a stronger result than a bisimulation—for example, an isomorphism—cannot be achieved due unfoldings of cycles.

Theorem 4.3.10 (*TS* and $N \oplus S$ are bisimilar).
Let S be an X_3-strategy of N, and let TS be as previously defined. There is a bisimulation that respects final states between TS and the LTS of $N \oplus S$. ⌟

Proof.
We prove this theorem on the level of service automata. Let N and S be service automata, let ϱ be the minimal simulation relation of S by $MP_{X_3}(N)$ that is

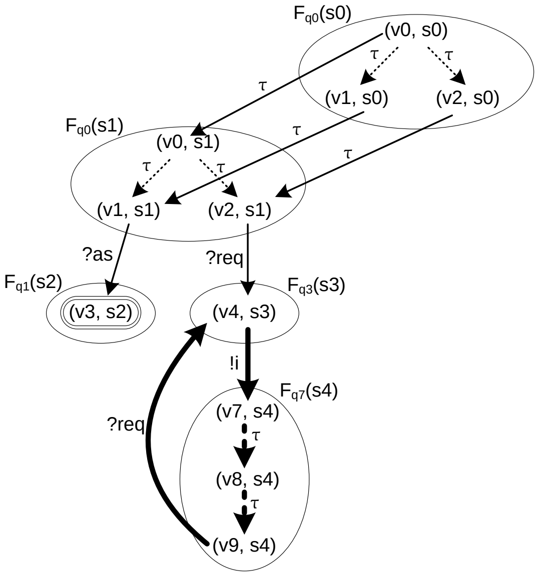


Figure 4.11.: Constructing the state space of Bank $\oplus$ Cust1 from the fragments and connections of Bank. Bank $\oplus$ Cust1 has a livelock (depicted in bold); the only final state is $(\mathsf{v3}, \mathsf{s2})$.

used for constructing TS. Let further be the labels of transitions of N and of S be preserved in the composition $N \oplus S$, and let also the labels of transitions of N be present in the fragments of N. We claim that the following relation $\varrho^* \subseteq Q_{N\oplus S} \times Q_{TS}$ is the required bisimulation:

$$\begin{aligned}\varrho^* = \{\langle (q_N, q_S, M), ((q_N, q_{MP}, M), q_S)\rangle \mid\ & (q_N, q_S, M) \in Q_{N\oplus S} \\ & \wedge (q_S, q_{MP}) \in \varrho \\ & \wedge ((q_N, q_{MP}, M), q_S) \in F_{q_{MP}}(q_S)\}\end{aligned}$$

By construction of TS, $F_{q_{MP}}(q_S) \in \mathcal{F}(TS)$, so ϱ^* is well defined.

The initial state of $N \oplus S$ is $({q_0}_N, {q_0}_S, [\,])$, the initial state of TS is $(({q_0}_N, {q_0}_{MP}, [\,]), {q_0}_S)$; so ϱ^* relates the initial states of the considered systems.

Let $\langle (q_N, q_S, M), ((q_N, q_{MP}, M), q_S)\rangle \in \varrho^*$. A transition in $N \oplus S$ originates either from a transition in N or from a transition in S. A transition of N leads to some state (q'_N, q_S, M') in $N \oplus S$. Obviously, the same transition is possible in state (q_N, q_{MP}, M) of the composition $N \oplus MP_{X_3}(N)$ as well, leading to (q'_N, q_{MP}, M') there. Both states and the corresponding transition between them are thus part of fragment $F_{q_{MP}}$ which proves that a transition from $((q_N, q_{MP}, M), q_S)$ to $((q'_N, q_{MP}, M'), q_S)$ with the same label is present in TS. Analogously, a transition internal to a fragment (which again stems from a transition in N) used in TS can be mimicked in $N \oplus S$.

Consider now a transition from state (q_N, q_S, M) to state (q_N, q'_S, M') of $N \oplus S$ that originates from a transition (q_S, x, q'_S) in S. As ϱ is a minimal simulation, there is a transition (q_{MP}, x, q'_{MP}) with $(q'_S, q'_{MP}) \in \varrho$, for some q'_{MP}. (This transition can be either internal (i. e., τ) or visible. However, as $MP_{X_3}(N)$ is deterministic, every successor state in ϱ is uniquely determined, and we do not have to distinguish between internal or visible transitions in the following.) Both transitions change message bag M in the same way, because they carry the same label. Fragment $F_{q'_{MP}}$ and connection instance $C_{(q_{MP},x,q'_{MP})}(q_S, q'_S)$ have been included in the construction of TS. This connection instance contains the required transition from $((q_N, q_{MP}, M), q_S)$ with x to $((q_N, q'_{MP}, M'), q'_S)$ in TS. The other way around, consider a transition in TS from $((q_N, q_{MP}, M), q_S)$ with x to some state $((q_N, q'_{MP}, M'), q'_S)$ that is part of an included connection instance $C_{(q_{MP},x,q'_{MP})}(q_S, q'_S)$. By construction of TS, we have $(q_S, x, q'_S) \in \delta_S$. Because enabledness of a transition in $N \oplus S$ and its effect on M just depend on M and x, transition $((q_N, q_S, M), x, (q_N, q'_S, M'))$ must be present in $N \oplus S$.

Let (q_N, q_S, M) be final in $N \oplus S$—that is, $q_N \in \Omega_N$, $q_S \in \Omega_S$, and $M = [\,]$; thus, for any q_{MP}, $((q_N, q_{MP}, M), q_S)$ is final in $F_{q_{MP}}$ and, by construction of TS, $((q_N, q_{MP}, M), q_S)$ is final in TS. The other way around, a state in TS is final according to Definition 4.3.9 if q_S is final and the state of the fragment instantiated with q_S is final. By construction of fragments, this implies $q_N \in \Omega_N$ and $M = [\,]$, so (q_N, q_S, M) is final in $N \oplus S$. $\square$

It is well known that bisimilar LTSs are undistinguishable for any formula of the temporal logic CTL that uses atomic propositions, which are preserved by the considered bisimulation relation [CGP00]. As weak termination can be expressed as *AG EF final* in this logic, Theorem 4.3.10 implies the following corollary.

Corollary 4.3.11 (*TS* strongly preserves weak termination).
The LTS of $N \oplus S$ weakly terminates iff TS weakly terminates. ⌟

The value of Corollary 4.3.11 is that it enables us to verify TS in place of the LTS of $N \oplus S$, because the bisimulation relation between both LTSs guarantees strong preservation of weak termination. With *strong preservation* we mean if weak termination does not hold for TS, it also does not hold for the LTS of $N \oplus S$. Consequently, our approach of constructing TS by gluing fragments of N is fully justified.

Based on Corollary 4.3.11, we can now define *matching* of an open net S with the set of fragments and connections of an open net N.

Definition 4.3.12 (matching with fragments).
For an open net N, let $\mathcal{F}(N)$ and $\mathcal{C}(N)$ denote the set of its fragments and connections. An open net S *matches* with $\mathcal{F}(N)$ and $\mathcal{C}(N)$ iff S is a partner of N, and TS, constructed from $\mathcal{F}(N)$ and $\mathcal{C}(N)$, weakly terminates. ⌟

The LTS of Bank $\oplus$ SA(Cust1) in Figure 4.11, constructed from fragments and connections of Bank, does not weakly terminate. Hence, Cust1 does not match with the fragments and connections of Bank.

The sets $\mathcal{F}(N)$ of fragments and $\mathcal{C}(N)$ of connections of an open net N, as defined in Definition 4.3.8, are a finite representation of all X_3-strategies of N. More precisely, $\mathcal{F}(N)$ and $\mathcal{C}(N)$ serve as a *finite generator set* for constructing the state space of $N \oplus S$, for any partner S of N. Matching—that is, checking whether an open net S is an X_3-strategy of N—means to construct the LTS TS of $N \oplus S$ from the sets $\mathcal{F}(N)$ and $\mathcal{C}(N)$ according to Definition 4.3.9 and then model check TS for weak termination.

The following two subsections are devoted to speed up the matching procedure.

4.3.4. Reducing fragments

In this section, we aim at reducing the size of the calculated fragments and thus the number of connections. That way, we achieve *three advantages*. First, reducing fragments actually speeds up the matching procedure. On the one hand, fewer states and connections reduce the effort when constructing TS from fragments and connections, and on the other hand, the smaller LTS TS reduces the model-checking effort in general. Second, we have to store only fragments of smaller size and third, abstracting parts from the fragments helps to hide trade secrets of a service, modeled as an open net N.

The reduction we are using is different from usual state-space reduction known from explicit state-space verification [Val98, CGP00]. Whereas traditional state-space reduction is 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, such as the symmetry method [CFJ93, ES93] or partial-order reduction [Val91, God91, Pel93], are completely out of scope in our setting. The problem is that application of these techniques must be restricted to the interior of a fragment, because changing number or connection of fragments would cause problems in the procedure of composing fragments to an LTS TS as described in the previous section. Consequently, we focus on an *a posteriori state-space reduction* that can be applied immediately after having constructed the fragments. The outcome of the reduction is then available in *every* LTS that uses the reduced fragments.

We continue by first restricting the states of all fragments to strongly connected components and then by defining abstraction rules that minimize the number of states of the fragments.

Strongly connected component-based fragment reduction

Deadlocks and livelocks in an LTS are terminal strongly connected components (TSCCs), which do not contain a final state (cf. Definition 2.1.2). If this TSCC is a singleton state without a self-loop, we have a deadlock; otherwise, we have a livelock.

In this light, replacing all strongly connected states in a fragment by single states is an obvious reduction. It may turn a livelock into a deadlock, but the reduced LTS is weakly terminating if and only if the original LTS is.

Definition 4.3.13 (SCC reduction).
Let $\mathcal{F}$ be a set of fragments, and let $\mathcal{C}$ be a set of connections. The reduced sets $\mathcal{F}'$ and $\mathcal{C}'$ are defined as follows. For each fragment $F = (V, \Sigma, E, \Omega)$, let $\equiv_F$ be the equivalence on V where $q \equiv_F q'$ iff q and q' are mutually reachable using transitions in E. Let $F' = (V', \Sigma, E', \Omega')$ 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 between fragments F_1 and F_2, let $([q]_{\equiv_{F_1}}, x, [q']_{\equiv_{F_2}}) \in C'$ iff there exist $q_1 \in [q]_{\equiv_{F_1}}$ and $q_1' \in [q']_{\equiv_{F_2}}$, with $(q_1, x, q_1') \in C$. Let $\mathcal{C}' = \{C' \mid C \in \mathcal{C}\}$. ⌟

For each SCC, we only store one representative. Each transition from (to) a state of an SCC is replaced by a transition from (to) the representative of this SCC.

As all replacements are executed simultaneously and consistently, it is easy to see that the reduction is well defined. Furthermore, all SCCs of single fragments are strongly connected in every LTS composed of the fragments. Mutual reachability of states in different SCCs is left invariant. It is thus easy to see that the following lemma holds.

Lemma 4.3.14 (SCC reduction strongly preserves weak termination).
An LTS, composed using fragments in $\mathcal{F}$ and connections in $\mathcal{C}$, is weakly terminating iff the corresponding reduced LTS is where, for every fragment $F \in \mathcal{F}$, F' is used instead, and every connection $C \in \mathcal{C}$ is replaced by the corresponding C'. ⌟

SCC reduction on fragments alone is applicable if the given open net N has cycles in its reachability graph. As none of the fragments of Figure 4.10(a) has an internal cycle, SCC reduction does not effect these fragments.

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 three applicable rules that have been adapted from the deadlock-preserving reduction rules by Juan et al. [JTM98].

Most of the rules in [JTM98] indeed do preserve livelocks. Thus, the adaptation was quite straight forward. The actual challenge in adapting the rules was rather to assure that their application is justified in *any* LTS that can be constructed using the considered fragments and connections.

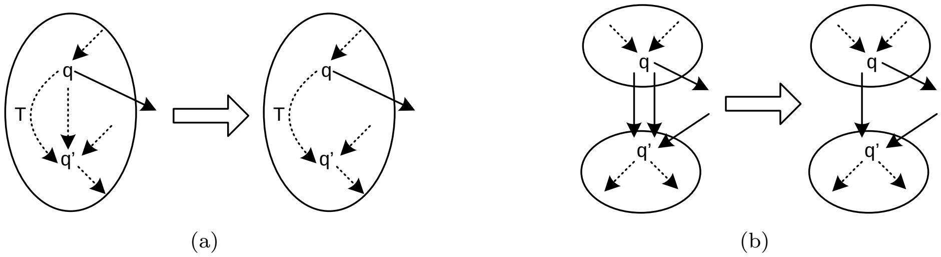


Figure 4.12.: Illustration of the transitive reduction rule: (a) fragment internal application and (b) removing a redundant connection.

For simplifying presentation, we introduce some notation. For a state q, let $F_q = (V_q, \Sigma, E_q, \Omega_q)$ be the fragment where $q \in V_q$. For a pair (q, q'), let $E_{(q,q')} = E_q$ if $F_q = F_{q'}$ and $(q, q') \in E_q$, and let $E_{(q,q')} = C_{(F_q, F_{q'})}$, otherwise. With this notation, we are able to refer to the fragment some state actually 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 via 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 ${}^\bullet q = \{q' \mid (q', q)$ appears in any fragment or connection$\}$ and $q^\bullet = \{q' \mid (q, q')$ appears in any fragment or connection$\}$ denote the set of predecessor and successor states of q, respectively.

We can now present the actual transformation rules.

Rule Transitive Reduction The transitive reduction rule [AGU72] aims at removing a transition (q, q') if and only if there is another transition sequence T from q to q'.

Constraint: There is a transition (q, q') where $q \xrightarrow{(E_q \cup E_{q'} \cup E_{(q,q')}) \setminus \{(q,q')\}} *q'$.

Application: $E_{(q,q')} := E_{(q,q')} \setminus \{(q, q')\}$.

Figure 4.12 illustrates the transitive reduction rule. It shows that the rule can be internally applied to a fragment (see Figure 4.12(a)) but also to remove redundant connections (see Figure 4.12(b)). The following lemma proves that we can apply this reduction rule in our setting.

Lemma 4.3.15 (justification of rule Transitive Reduction).
Rule Transitive Reduction strongly preserves weak termination. ⌟

Proof.
Let TS be constructed of fragments, and assume that it contains an instance of (q, q'). If this transition is internal to a fragment, then the whole sequence required in the constraint is internal to the same fragment. So q' is still reachable

from q. If (q, q') belongs to a connection C between fragments F_1 and F_2, then fitting instances of both fragments and the connection C are present in TS, and the constraint again assures reachability of q' from q via a member of the same connection. Thus, mutual reachability of states is left invariant, which is sufficient for asserting preservation of weak termination. □

As none of the fragments in Figure 4.10(a) has states that are transitively reachable, transitive reduction does not effect these fragments.

Rule Sequence The intuition behind this rule is to reduce sequences of states and synchronization states to minimize all paths to deadlocks or final states. A state q can be removed if

(1) all predecessor states of q are in the same fragment as q; or

(2) all successor states of q are in the same fragment as q.

Constraint: There is a state q such that there is a transition $(q, q^*) \in E_q$—that is, q is not a deadlock in its own fragment and q is not the initial state of $\mathcal{F}(TS)$—and

(1) ${}^\bullet q \times \{q\} \subseteq E_q$; or

(2) $\{q\} \times q^\bullet \subseteq E_q$.

Application: (i) Add a transition from every predecessor of state q to each of its successor states. Then remove all transitions from (ii) and to (iii) state q, and finally (iv) remove state q. Formally,

(i) for every $q' \in {}^\bullet q$ and every $q'' \in q^\bullet$, $E^* := E^* \cup \{(q', q'')\}$, where $E^* = E_{(q',q)}$ if $E_{(q',q)} \neq E_q$, and $E^* = E_{(q,q'')}$, otherwise;

(ii) for all $q' \in {}^\bullet q$, $E_{(q',q)} := E_{(q',q)} \setminus \{(q', q)\}$;

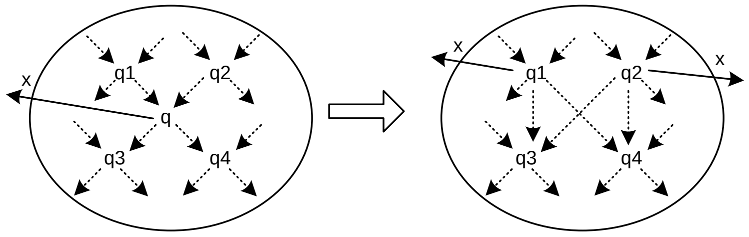
(iii) for all $q' \in q^\bullet$, $E_{(q,q')} := E_{(q,q')} \setminus \{(q, q')\}$; and

(iv) $V_q := V_q \setminus \{q\}$.

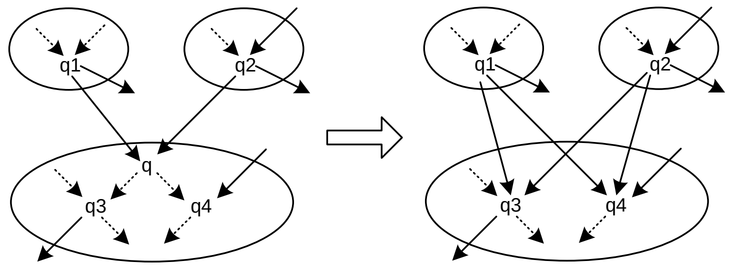
An example for each of the two forms of this rule is shown in Figure 4.13. State q is always the state that is removed. The following lemma proves that we can apply this reduction in our setting.

Lemma 4.3.16 (justification of rule Sequence).
Rule Sequence strongly preserves weak termination. ⌟

Proof.
Let TS be an LTS built from fragments, and let F_q be the fragment to be changed by the reduction rule. We prove both forms of this rule. First, we show for an arbitrary LTS that every deadlock or final state being reachable before the reduction is also reachable after the reduction, and no new deadlocks are introduced by



(a)



(b)

Figure 4.13.: Illustration of the sequence rule: (a) all predecessors of q are in the same fragment as q, and (b) all successors of q are in the same fragment as q.

the reduction. Afterwards, we show that these properties hold for any LTS that can be constructed from fragments.

Consider a path to a final state or to a deadlock that passes through q. This path passes through some $q' \in {}^{\bullet}q$ and some $q'' \in q^{\bullet}$. Applying the reduction rule yields a bypass transition (q', q''), for all q', q'', and the only removed state is q. Clearly, reachability remains invariant by adding bypass transition (q', q''). So, reachability of a final state or a deadlock remains invariant, for all states that reach q, and also no additional deadlock has been introduced by the reduction. State q is by assumption not a deadlock and the only state that is removed. Thus, deadlock states remain invariant by the reduction. Now assume, q is a final state. By assumption, q has at least one successor q^* in its own fragment. According to Definition 2.6.2 (weak termination), a final state must be reachable from every state of the LTS and therefore also from q^*. As reachability is not affected by the reduction, reachability of a final state from q is the same as reachability of a final state from the (still present) q^*.

It remains to show that these properties hold for any LTS TS. Clearly, instances of manipulated connections are only present in TS with instances of F_q. Consider again the path through some $q' \in {}^{\bullet}q$ and some $q'' \in q^{\bullet}$. By our constraint, at least one of (q', q) and (q, q'') is internal to F_q. If one of these transitions is outside the fragment F_q, the bypass transition (q', q'') is member of the same connection. Thus, an instance of (q', q'') is present in TS if and only if transitions (q', q) and (q, q'') are present in the unreduced TS. So reachability of deadlocks and final states remains invariant in any LTS. Suppose q is a final state. As the constraints assert that q has at least one successor in its own fragment, it has a successor in TS. Hence, reachability of a final state remains invariant for all states that reach q in any LTS. □

Applying this rule to the fragments of our example in Figure 4.10(a) results in removing the state v5 from fragment F_{q6} and the states v7 and v8 from fragment F_{q7}. Accordingly, we have to change connections $\mathsf{C}_{(\mathsf{v4,!i,v7})}$ and $\mathsf{C}_{(\mathsf{v4,!ap,v5})}$ to $\mathsf{C}_{(\mathsf{v4,!i,v9})}$ and $\mathsf{C}_{(\mathsf{v4,!ap,v6})}$. The example shows that this rule is a rather powerful reduction technique.

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 $F_q = F_{q'}$, $q \in \Omega_q$ iff $q' \in \Omega_q$, and, for all fragment-internal transitions and connections E^*, $(q, q'') \in E^*$ iff $(q', q'') \in E^*$.

Application: (1) Redirect every transition from a predecessor of state q to q', (2) remove all transitions from q to its successors, and finally (3) remove q. Formally,

(1) for every $q^* \in {}^\bullet q$, $E_{(q^*,q)} := (E_{(q^*,q)} \setminus \{(q^*, q)\}) \cup \{(q^*, q')\}$;

(2) for every $q'' \in q^\bullet$, $E_{(q,q'')} := E_{(q,q'')} \setminus \{(q, q'')\}$; and

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

Figure 4.14 illustrates this reduction rule. In Figure 4.14(a), the removal of state q is local to the fragment of q. The example in Figure 4.14(b) shows that also surrounding fragments may be involved, and hence also connections have to be redirected or removed. The following lemma justifies the applicability of this reduction in our setting.

Lemma 4.3.17 (justification of rule Equivalent States).
Rule Equivalent States strongly preserves weak termination. ⌟

Proof.
Consider an instance of F_q. This fragment contains both states q and q' in its unreduced version. We prove for an arbitrary LTS that every deadlock or final state being reachable in the unreduced LTS is also reachable in the reduced LTS, and no new deadlocks are introduced by the reduction. Afterwards, we show that these properties hold for any LTS that can be constructed from fragments.

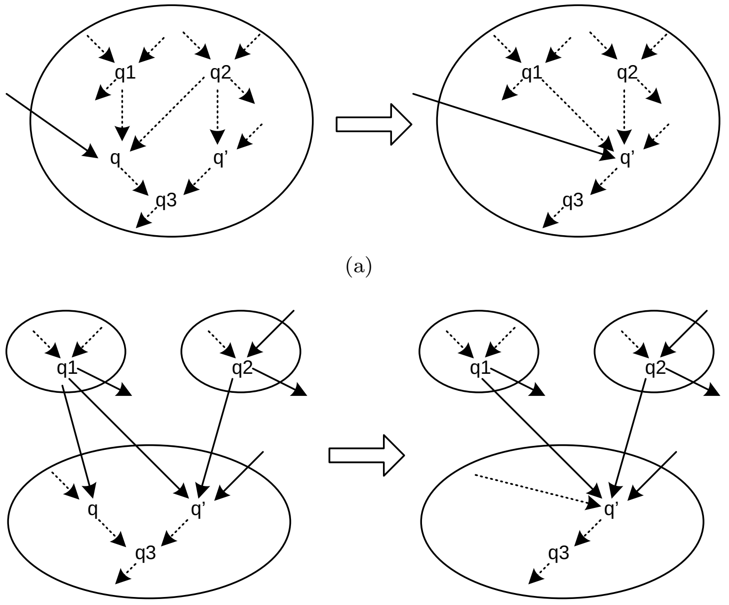


Figure 4.14.: Illustration of equivalent states rule: (a) fragment internal application and (b) application that involves several fragments.

The constraints assure that q and q' have the same successors. Thus, redirection of all transitions (q^*, q) to (q^*, q') guarantees that reachability remains invariant in the reduced LTS, for all states that reach q. If q is a deadlock or a final state, so is q' (follows from the constraint). Hence, reachability of a deadlock or a final state remains invariant in the reduced LTS, for all states that reach q. If q is not a deadlock, then q' is also not a deadlock. As reachability remains invariant in the reduced LTS, we conclude that the reduction does not add additional deadlocks.

We continue by proving that these properties hold for any LTS. The redirected transitions appear in the same fragment or connection as the original one. So replacing F_q by the reduced fragment is independent of the actual construction of any LTS. 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, q^*) is guaranteed to be independent from the actual construction of any LTS. Suppose now there is a transition (q, q^*), and q^* is not a state of F_q. The constraint assures the existence of transition (q', q^*), which is also a member of the same connection. Hence, removing transition (q, q^*) affects only one connection, which is because of the presence of q' in the reduced fragment still present in the reduced LTS. Hence, substituting F_q by the reduced fragment is independent of the actual construction of any LTS. □

As none of the fragments of Figure 4.10(a) has equivalent states, the application of this reduction rule does not effect these fragments.

From Lemma 4.3.15, from Lemma 4.3.16, and from Lemma 4.3.17 we can conclude that we can apply the three reduction rules in any order, and the minimized fragments strongly preserve weak termination.

Corollary 4.3.18 (justification of rule-based reduction).
Let TS' result from applying any sequence of reduction rules as described above to TS. Then, TS weakly terminates iff TS' weakly terminates. ⌟

The abstraction rules presented in this section are related to work on minimization of LTSs (see [Val95, FGK$^+$96, PV99], for instance). These papers present congruences with respect to composition, which never equate an LTS with a property, such as deadlock freedom or livelock freedom, with one, which does not satisfy that property. That way, one can minimize an open net N to an open net N' such that N' preserves deadlocks and livelocks. However, as S is unknown at the minimization phase, only N can be minimized, whereas in our approach, we construct an overapproximation $N \oplus MP_{X_3}(N)$ of any state space $N \oplus S$ and minimize $N \oplus MP_{X_3}(N)$ instead of only minimizing N.

The condensation rules presented in this subsection reduce TS and therefore in particular speed up the second step of the matching procedure—that is, model checking TS for weak termination. Remember that the more complicated matching procedure (in comparison to X_1-strategies) became necessary, because we want to exclude livelocks. As a drawback, the approach will also calculate the LTS of $N \oplus S$ if S causes a deadlock in the composition with N. For checking deadlock freedom, a more efficient procedure has been introduced for X_1-operating guidelines in Section 4.1.3, namely, encoding deadlock freedom into an annotation. In the next subsection, we will adapt this procedure to the setting of fragments.

4.3.5. An annotation function for fragments to encode deadlock freedom

The aim of this section is to detect already during the construction of the LTS of $N \oplus S$ whether $N \oplus S$ may deadlock. A fragment F corresponds to a state q_{MP} of the most permissive X_3-strategy $MP_{X_3}(N)$ of N. Each state $q = (m, q_{MP})$ of F consists of a marking of N (i. e., a knowledge value of N) and the state q_{MP} of $MP_{X_3}(N)$. That way, we can directly apply the three conditions of Lemma 4.1.14 to fragments. As a main difference to the annotation function for X_1-operating guidelines (cp. Definition 4.1.15), the Boolean formula, annotated to a fragment F, is not a conjunction over the set of all markings in F that do not enable any transition, but a conjunction over the set of all *TSCCs* in F—that is, all states q that have at most successors in a different fragment than F.

Definition 4.3.19 (annotation function for fragments).
Let N be an open net, and let $MP_{X_3}(N) = (Q, MC^I, MC^O, \delta, q_0, \Omega)$ be the most permissive X_3-strategy of N that has been used for constructing the set $\mathcal{F}(N)$

of all (reduced) fragments and the set $\mathcal{C}(N)$ of all connections of N. Define the *annotation function of* $(\mathcal{F}(N), \mathcal{C}(N))$ as $\psi_{\mathcal{F}(N)} : \mathcal{F}(N) \to \mathcal{BF}^{\mathcal{MC}^+}$ where, for each fragment $F = (V_F, \Sigma_F, E_F, \Omega_F) \in \mathcal{F}(N)$

$$\psi_{\mathcal{F}(N)}(F) = \bigwedge_{v=(m,q)\in V_F \wedge v \text{ is a TSCC}} \Big(\psi_{ii}(v) \vee \psi_{iii}(v)\Big) \quad \text{with}$$

- $\psi_{ii}(v) = \tau \vee \Big(\bigvee_{x\in MC^O} x\Big) \vee \Big(\bigvee_{x\in MC^I, m(x)>0} x\Big)$;
- $\psi_{iii}(v) = \begin{cases} \mathit{final}, & \text{if } v \in \Omega_F; \\ \mathit{false}, & \text{otherwise.} \end{cases}$

⌟

Let the corresponding service automaton $SA(S)$ of an open net S be in state q_S such that q_S is related to a state q of $MP_{X_3}(N)$ by the minimal simulation relation. Let v be a TSCC of F_q (the corresponding fragment of q). Formula $\psi_{ii}(v)$ encodes the second condition of Lemma 4.1.14. It evaluates to *true* if q_S enables any connection having v as its source. So it assigns *true* to any of the literals $x \in MC^I \cup MC^O$ of $MP_{X_3}(N)$ that represent a connection, or it assigns *true* to the literal τ.

Formula $\psi_{iii}(v)$ encodes that the TSCC v is a final state of fragment F_q. Then, the formula $\psi_{iii}(v)$ contains literal *final* and by Definition 4.1.3, $SA(S)$ assigns *true* to *final* if and only if q_S is a final state of $SA(S)$.

As for annotated automata, the resulting Boolean formulae may not be in normal form. However, it is easy to adapt the normalization procedure for annotated automata of Section 4.1.1. For the fragments in Figure 4.10(a), we calculate the following (normalized) annotations:

$\psi_{\mathcal{F}(\mathsf{Bank})}(\mathsf{F_{q0}}) = \tau \vee (?\mathsf{as} \wedge ?\mathsf{req})$
$\psi_{\mathcal{F}(\mathsf{Bank})}(\mathsf{F_{q1}}) = \tau \vee \mathit{final}$
$\psi_{\mathcal{F}(\mathsf{Bank})}(\mathsf{F_{q3}}) = \tau \vee !\mathsf{ap} \vee !\mathsf{i}$
$\psi_{\mathcal{F}(\mathsf{Bank})}(\mathsf{F_{q6}}) = \tau \vee \mathit{final}$
$\psi_{\mathcal{F}(\mathsf{Bank})}(\mathsf{F_{q7}}) = \tau \vee ?\mathsf{req}$

The set of fragments and connections of an open net N and the canonical Boolean annotation form the X_3*-operating guideline of* N.

Definition 4.3.20 (X_3-operating guideline).
For an open net N, let $\mathcal{F}(N)$ be its set of fragments, $\mathcal{C}(N)$ its set of connections, and $\phi = \psi_{\mathcal{F}(N)}$ its annotation function. The X_3*-operating guideline of* N is defined by $OG_{X_3}(N) = (\mathcal{F}(N), \mathcal{C}(N), \phi)$. ⌟

Based on the X_3-operating guideline, as a representation of all X_3-strategies of an open net N, we can give an improved definition of matching that takes into account the presence of the annotation function.

Definition 4.3.21 (matching with X_3-operating guideline).
For an open net N, let $OG_{X_3}(N) = (\mathcal{F}(N), \mathcal{C}(N), \phi)$ be its X_3-operating guideline. An open net S *matches* with $OG_{X_3}(N)$ iff

- S matches with $(\mathcal{F}(N), \mathcal{C}(N))$ using minimal simulation relation $\varrho \subseteq Q_{SA(S)} \times Q_{MP}$, with Q_{MP} is the set of states of $MP_{X_3}(N)$); and
- for each $(q_S, q) \in \varrho : \ \beta_{SA(S)}(q_S) \models \phi(F_q)$, where $F_q \in \mathcal{F}(N)$. ⌟

An open net S matches with $OG_{X_3}(N)$ if (1) S and N are partners, (2) the minimal simulation relation that is used for gluing fragments of N to the LTS TS of $N \oplus S$, constitutes for each pair (q_S, q) of states a satisfying assignment for $\phi(F_q)$, and (3) TS weakly terminates.

In our example, the customer Cust1 in Figure 4.11 actually satisfies conditions (1) and (2). However, as already shown, the LTS of Bank$\oplus SA($Cust1$)$ constructed from fragments and connections of Bank violates condition (3), as it does not weakly terminate. Hence, Cust1 does not match with $OG_{X_3}($Bank$)$.

The next theorem is the main theorem of this subsection. It proves that the X_3-operating guideline of an open net N represents all X_3-strategies of N.

Theorem 4.3.22 ($OG_{X_3}(N)$ represents all X_3-strategies of N).
An open net S is an X_3-strategy of an open net N iff S matches with $OG_{X_3}(N)$. ⌟

Proof.
Let $OG_{X_3}(N) = (\mathcal{F}(N), \mathcal{C}(N), \phi)$. From Theorem 4.3.10, we conclude that S matches with $(\mathcal{F}(N), \mathcal{C}(N))$ if and only if the LTS of $N \oplus S$ weakly terminates. The annotation function for fragments is defined analogously to the annotation function for X_1-operating guidelines. So we can apply Theorem 4.1.17 and conclude that ϕ is satisfied by $SA(S)$ if and only $N \oplus S$ is deadlock-free. Thus, ϕ does not exclude any X_3-strategy, and hence the theorem holds. □

Theorem 4.3.22 justifies that, given any finite-state open net N, we have an algorithm for constructing $OG_{X_3}(N)$ and a procedure to decide whether an open net S matches with $OG_{X_3}(N)$.

4.3.6. Experimental results and discussion

We use a prototypical implementation of the algorithm for constructing the sets of all fragments and connections for an open net N to experiment with a number of real-life service models. Furthermore, we compare X_1-operating guidelines and X_3-operating guidelines—in particular, some complexity issues—and discuss related work.

Experimental results

The results presented in this section have also been prototypically implemented in the service analysis tool Fiona. So, given an open net N, Fiona can calculate the most permissive X_3-strategy $MP_{X_3}(N)$ of N together with the (minimized) fragments and connections of N. Furthermore, Fiona can also check whether an open net S matches with the fragments of N by constructing the state space for $N \oplus S$.

In this experiment, we use the same services as in the experiments in Sections 4.1.4 and 4.2.4.

For each open N, Table 4.3 provides information about the size of N, the size of its corresponding service automaton, structural properties of N, the state space of the most permissive X_3-strategy $MP_{X_3}(N)$ of N, the size of the fragments, and the time for calculating $MP_{X_3}(N)$ and computing the reduced fragments. More precisely, columns 2–4 refer to the number of places, transitions, and interface places of N; columns 5 and 6 refer to the number of states and transitions of $SA(N)$; columns 7 and 8 provide information whether N has concurrency and cycles, respectively; columns 9 and 10 refer to the number of states and transitions of the most permissive X_3-strategy of N[3]; $|V|$ and $|V_{red}|$ in columns 11 and 12 refer to the number of states of *all* fragments before and after applying the reduction rules; $\frac{|V|}{Q}$ and $\frac{|V_{red}|}{Q}$ in columns 13 and 14 show the average number of states *per* fragment before and after applying the reduction rules. Finally, t_{MP} denotes the time for computing the most permissive X_3-strategy of N, and t_{red} shows the time for reducing the fragments.

As an example, the most permissive X_3-strategy of ‘Online Shop 1’ has 12 states. Thus, Fiona computed 12 fragments. The sum of all states of these 12 fragments is $|V| = 137$. Applying the proposed abstraction rules results in $|V_{red}| = 15$ states. So we have in average 11.4 states per fragment before and 1.3 states per fragment after the reduction. Hence, for matching an open net S with the reduced fragments of N only $\frac{15}{137} = 11\%$ of the actual state space has to be model checked. The time for computing $MP_{X_3}(N)$ is 4 seconds; reducing the fragments takes no time.

Based on the experimental results, we make the following three observations:

The average number of states per fragment is not very high in general; for example, in nine service models it is lower than 5. Consequently, there is little scope for reduction in such examples. One reason might be that all open nets have been structurally reduced using the well-known Murata rules [Mur89] before transforming their state spaces. We apply these rules, because they significantly speed up the computation of the most permissive X_3-strategy while preserving all relevant properties.

For three examples, the overall computation time takes more than one hour. This reflects the high worst-case complexity of the algorithm for calculating the most permissive X_3-strategy. Also the minimization of the fragments takes in four examples more than 10 minutes. The reason is that the transitive reduction rule has a worst-case complexity of $\mathcal{O}(n^3)$ [AGU72] (where n is the number $|Q|$ of all states of the most permissive X_3-strategy), and we were interested in the best

[3]Fiona does not compute the empty state, as this state is not used for computing fragments.

Table 4.3.: Calculation of $OG_{X_3}(N)$ with Fiona. All experiments were obtained on an UltraSPARC III processor with 900MHz and 4 GB RAM running Solaris 10.

	N			$SA(N)$		Props		$MP_{X_3}(N)$		$\mathcal{F}(N)$				h:mm:ss	
Service	$\|P\|$	$\|T\|$	$\|P^{Int}\|$	$\|Q\|$	$\|\delta\|$	$\|\|$	$\circlearrowleft$	$\|Q\|$	$\|\delta\|$	$\|V\|$	$\|V_{red}\|$	$\frac{\|V\|}{Q}$	$\frac{\|V_{red}\|}{Q}$	t_{MP}	t_{red}
Loan Approval	38	25	6	26	33	✓	-	7	8	43	10	6.1	1.4	0:00:01	0:00:00
Purchase Order	22	7	10	12	15	✓	-	168	548	464	168	2.8	1.0	0:00:03	0:00:04
Olive Oil Ordering	12	6	6	6	6	-	-	16	27	33	20	2.1	1.3	0:00:00	0:00:00
Travel Service 1	15	5	8	7	7	✓	-	56	140	120	56	2.1	1.0	0:00:00	0:00:01
Travel Service 2	22	9	12	10	11	✓	-	288	1,008	672	320	2.3	1.1	0:00:22	0:00:10
Online Shop 1	61	58	7	205	463	✓	-	12	15	137	15	11.4	1.3	0:00:04	0:00:00
Online Shop 2	72	69	8	308	744	✓	-	7	7	77	8	11.0	1.1	0:00:06	0:00:00
Beverage Machine	12	8	7	5	8	-	✓	11	24	37	15	3.4	1.4	0:00:00	0:00:01
Philosophers #3	22	10	6	46	70	✓	✓	67	96	499	67	7.5	1.0	0:00:01	0:00:01
Philosophers #5	36	16	10	574	1,476	✓	✓	1,432	3,435	43,848	1,432	30.6	1.0	0:05:06	0:39:19
SMPT Protocol	123	133	12	8,345	34,941	-	✓	1,216	4,752	13,148	2,894	10.8	2.4	4:10:05	0:29:10
Registration	127	88	6	1,057	2,927	✓	-	7	8	2,239	13	320.0	1.9	0:00:20	0:00:32
Process 1	39	27	20	97,511	468,072	✓	-	896	3,924	1,428	1,365	1.6	1.5	0:00:10	0:02:41
Process 2	29	18	13	244	758	✓	-	1,607	7,206	4,844	3,782	3.0	2.4	0:00:12	0:16:12
Process 3	26	15	13	992	3,744	✓	-	236	724	1,171	236	5.0	1.0	1:14:36	0:00:12
Process 4	29	14	19	160	494	✓	-	6,820	40,350	24,688	13,903	3.6	2.0	4:41:06	6:07:55

possible reduction. So 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 states and transitions that can be removed.

The proposed abstraction rules seem to be powerful, as the state spaces of the fragments are reduced significantly. None of the reduced example processes has more than 2.5 states per fragment in average, and five processes contain only a single state per each fragment. Consequently, building the LTS TS of $N \oplus S$ from reduced fragments results in all examples in a *smaller* state space (compared to the LTS composed from non-reduced fragments). On the one hand, this helps to hide internals of the service. On the other hand, we assume that those reduced state spaces can be model checked more efficiently due to the smaller number of states. A validation of this assumption is, however, subject of ongoing research.

Discussion

We start with a short complexity analysis of X_3-operating guidelines.

Computing $OG_{X_3}(N)$ is proportional to the product of N and the most permissive X_3-strategy $MP_{X_3}(N)$. To compute $MP_{X_3}(N)$, we have to compute $MP_{X_1}(N)$. Hence, the space complexity of $OG_{X_3}(N)$ is $\mathcal{O}(|N| \cdot |MP_{X_1}(N)|)$. As we assume that calculating $MP_{X_3}(N)$ takes the same time than calculating $MP_{X_1}(N)$, time and space complexity of $OG_{X_3}(N)$ and $OG_{X_1}(N)$ are the same.

Also the time complexity for matching remains the same: Computing the state spaces of $N \oplus S$ from fragments of N requires a minimal simulation relation of S by the most permissive X_3-strategy of N, which is proportional to the product of $OG_{X_3}(N)$ and S. So we have $\mathcal{O}(|OG_{X_3}(N)| \cdot |S|)$. Model checking the respective transition system can be done on-the-flight, and hence we have—as for X_1-operating guidelines (see Section 4.1.4)—an overall time complexity of $\mathcal{O}(|OG_{X_3}(N)| \cdot |S|)$.

Although the worst-case complexity is exponential, the experimental results suggest that the proposed X_3-operating guideline of an open net is feasible for practical applications. As in case of X_1-operating guidelines, the complexity is mainly caused by high computation time for the most permissive X_3-strategy. To reduce the average complexity of computing $OG_{X_3}(N)$, we combine the two algorithms for calculating the most permissive X_1-strategy of N (see Definition 4.1.10) and the most permissive X_3-strategy of N (see Definition 4.3.1).

Comparing the experiments of this section with the one on X_1-operating guidelines in Section 4.1.4, we observe that calculating $MP_{X_3}(N)$ takes more time than calculating $MP_{X_1}(N)$. For example, computing $MP_{X_1}(N)$ of the 'SMTP Protocol' takes 47 minutes, whereas computing $MP_{X_3}(N)$ takes more than 4 hours. At this stage we do not have an explanation for this difference. We assume that this is caused by some weaknesses in the implementation. We think that the implementation for both, computation of the most permissive X_3-strategy and minimizing fragments, gives room for improvements.

4.4. Representing strategies in case of weak termination and cover

The aim of this section is to introduce a finite representation of all X_3-strategies S of an open net N such that a given set Y of nodes of N is covered in the composition $N \oplus S$. We refer to S as an $X_4(Y)$-strategy of N. In addition, we are also interested in representing all open nets S' such that $N \oplus S'$ is quasi-live. We refer to S' as an X_4-strategy of N. We will show that the notion of a global constraint, which has been introduced in Section 4.2.2 for X_1-operating guidelines, can be adapted to X_3-operating guidelines (Section 4.4.1). The achieved results are discussed in Section 4.4.2.

4.4.1. A finite representation of all $X_4(Y)$-strategies

In Section 4.2.2, we proved that the information for deciding whether an open net S ensures that in the composition $N \oplus S$ a given set Y of nodes of N are covered can be derived from the knowledge that the most permissive X_1-strategy of N has of N. We showed that we can construct a global constraint χ encoding this information. The global constraint specifies which states of $OG_{X_1}(N)$ have to be used in the minimal simulation relation of $SA(S)$ by $OG_{X_1}(N)$ such that the nodes of N are covered in $N \oplus S$. As the most permissive X_3-strategy of N has in each state knowledge of N, it is easy to see that the notion of a global constraint can be lifted to X_3-operating guidelines. Consequently, extending X_3-operating guidelines by a global constraint with proposition over the states of the most permissive X_3-strategy (i. e., fragments used in the LTS of $N \oplus S$) represents all $X_4(Y)$-strategies of N. Like we did in Section 4.2.3 for X_2-strategies, this representation can then be adjusted to represent all X_4-strategies.

Definition 4.4.1 ($X_4(Y)$-operating guideline).
Let N be an open net with a set $Y \subseteq P \cup T$ of nodes of N. Let $OG_{X_3}(N) = (\mathcal{F}(N), \mathcal{C}(N), \phi)$ be the X_3-operating guideline of N, and let $MP_{X_3}(N)$ be the most permissive X_3-strategy of N. For a place $p \in P$ of N and a state q of $MP_{X_3}(N)$, let $p \sim q$ iff there is a marking $m \in k_{MP(N),N}(q)$, where $m(p) > 0$. For a transition $t \in T$ of N and a state q of $MP_{X_3}(N)$, let $t \sim q$ iff there is a marking $m \in k_{MP(N),N}(q)$, where t is enabled. Then, $OG_{X_4(Y)}(N) = (\mathcal{F}(N), \mathcal{C}(N), \phi, \chi)$ with

$$\chi = \bigwedge_{y:\ y \in Y} \bigvee_{q:\ y \sim q} q$$

is the $X_4(Y)$-*operating guideline of* N.

Let $Y = T$. Then $OG_{X_4(Y)}(N)$ is the X_4-*operating guideline* $OG_{X_4}(N)$ *of* N.⌟

The sets $\mathcal{F}(N)$ and $\mathcal{C}(N)$ of fragments and connections of N, the annotation function ϕ, and the global constraint χ represent all $X_4(Y)$-strategies of N. An open net S matches with $OG_{X_4(Y)}(N)$ if it matches with $OG_{X_3}(N)$ and satisfies the global constraint χ.

If $Y = T$ of N, $OG_{X_4(Y)}(N)$ is the X_4-operating guideline of N and represents all X_4-strategies of N. An open net S is an X_4-strategy of N if (like for X_2-strategies) it matches with $OG_{X_4(Y)}(N)$ and, for every state of the corresponding service automaton $SA(S)$ of S, there exists a state q of $MP_{X_3}(N)$ in the minimal simulation relation ϱ of $SA(S)$ by $MP_{X_3}(N)$ and q is not the empty state of $MP_{X_3}(N)$.

Definition 4.4.2 (matching with an $X_4(Y)$-operating guideline). Let $OG_{X_4(Y)}(N) = (\mathcal{F}(N), \mathcal{C}(N), \phi, \chi)$ be the $X_4(Y)$-operating guideline of an open net N. An open net S *matches* with $OG_{X_4(Y)}(N)$ iff

- S matches with $(\mathcal{F}(N), \mathcal{C}(N), \phi)$ using the minimal simulation relation ϱ of states of $SA(S)$ by states Q_{MP} of $MP_{X_3}(N)$; and
- χ evaluates to *true* in the assignment $\gamma_{SA(S)} : Q_{MP} \rightarrow \{\mathit{true}, \mathit{false}\}$, where $\gamma_{SA(S)}(q) = \mathit{true}$, for $q \in Q_{MP}$ iff there is a state q_S of $SA(S)$ such that $(q_S, q) \in \varrho$;

Let $OG_{X_4(Y)}(N)$ with $Y = T_N$ be the X_4-operating guideline of N. Open net S *matches* with $OG_{X_4(Y)}(N)$ iff

- S matches with $OG_{X_4(Y)}(N)$ using the minimal simulation relation ϱ of states of $SA(S)$ by states Q_{MP} of $MP_{X_3}(N)$; and
- for all states q_S of $SA(S)$, there is a $q \in Q_{MP}$ with $(q_S, q) \in \varrho$ and q is not the empty state $q_\emptyset$ of Q_{MP}. ⌟

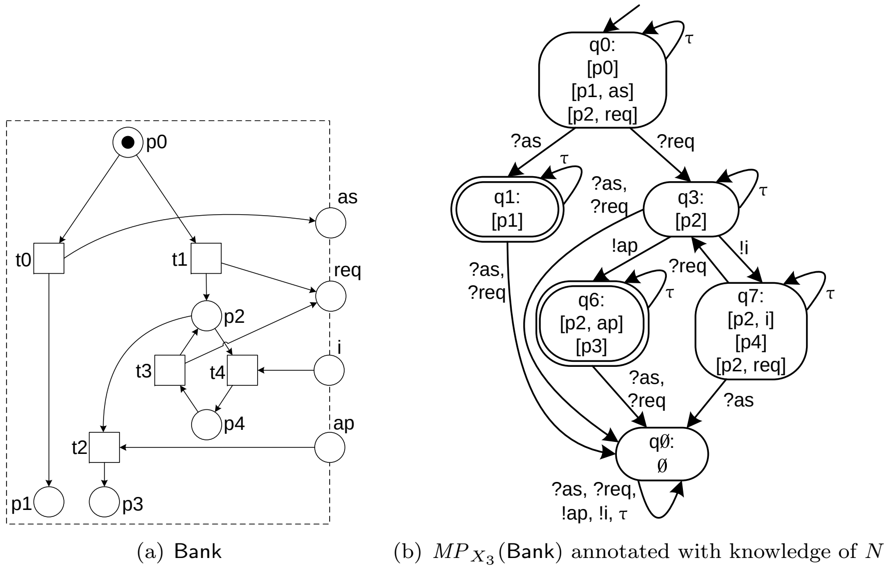
From Theorem 4.2.5 and from Theorem 4.3.22, we can directly conclude that the proposed notion of an $X_4(Y)$-operating guideline indeed represents all $X_4(Y)$-strategies of N.

Corollary 4.4.3 ($OG_{X_4(Y)}(N)$ represents all $X_4(Y)$-strategies). An open net S is an $X_4(Y)$-strategy of an open net N iff S matches with $OG_{X_4(Y)}(N)$. ⌟

Analogously, from Corollary 4.2.8 and from Theorem 4.3.22, we can directly conclude that the proposed notion of an X_4-operating guideline indeed represents all X_4-strategies of N.

Corollary 4.4.4 (X_4-operating guideline represents all X_4-strategies). An open net S is an X_4-strategy of an open net N iff S matches with $OG_{X_4}(N)$. ⌟

The most permissive X_3-strategy of Bank is shown in Figure 4.15(b). The representation of all X_4-strategies of Bank is the X_4-operating guideline $OG_{X_4}(\mathsf{Bank}) = (OG_{X_3}(\mathsf{Bank}), \chi)$ with $\chi \equiv \mathsf{q0} \wedge \mathsf{q0} \wedge \mathsf{q6} \wedge \mathsf{q7} \wedge \mathsf{q7}$, which is equivalent to $\chi \equiv \mathsf{q0} \wedge \mathsf{q6} \wedge \mathsf{q7}$.



(a) Bank (b) MP_{X_3}(Bank) annotated with knowledge of N

Figure 4.15.: Open net Bank and its most permissive X_3-strategy MP_{X_3}(Bank).

4.4.2. Discussion

Comparing an $X_4(Y)$-operating guideline and an X_3-operating guideline of an open net N, we identify that both representations have the same most permissive X_3-strategy. The reason is that, analog to X_1-operating guidelines and $X_2(Y)$-operating guidelines, each $X_4(Y)$-strategy of N is also an X_3-strategy of N. If the most permissive X_3-strategy of N is not an $X_4(Y)$-strategy of N, then the set of $X_4(Y)$-strategies is empty.

The complexity of $X_4(Y)$-operating guidelines can be derived from the complexity of X_3-operating guidelines (cf. Section 4.3.6) and of $X_2(Y)$-operating guidelines (cf. Section 4.2.4). An $X_4(Y)$-operating guideline of N can be calculated in time $\mathcal{O}(|N|\cdot|OG_{X_3}(N)|)$. The space complexity is $\mathcal{O}(|N|\cdot|OG_{X_3}(N)|)$ for the fragments and $\mathcal{O}(|Y|\cdot|Q|)$ for the global constraint, where $|Y|$ is the number of nodes of N to be covered, and $|Q|$ denotes the number of states of $MP_{X_3}(N)$. Hence, the overall space complexity is $\mathcal{O}(|N|\cdot|OG_{X_3}(N)|+|Y|\cdot|Q|)$. The time complexity for matching results from matching an open net S with $OG_{X_3}(N)$, which has complexity $\mathcal{O}(|OG_{X_3}(N)|\cdot|S|)$, and evaluating the global constraint χ, which can be done in linear time with respect to the size of the constraint. So the overall complexity is $\mathcal{O}(|OG_{X_3}(N)|\cdot|S|+|Y|\cdot|Q|)$.

The notion of an $X_4(Y)$-operating guideline has not been implemented so far. Thus, we cannot provide any experimental results. However, as this notion combines X_3-operating guidelines and global constraints (from $X_2(Y)$-operating guidelines), we believe that it is—as the other three representations—feasible in practical applications.

The notion of classical soundness [Aal98] for workflow nets ensures that a workflow net weakly terminates, and every transition of this net is not dead. The X_4-operating guideline of N represents all open nets S of an open net N such that the composition $N \oplus S$ is classical sound.

4.5. Representing strategies in case of strict termination

The aim of this section is to extend the set $X \in \{X_3, X_4, X_4(Y)\}$ of open-net properties considered so far by strict termination. Strict termination excludes those X-strategies S of N that contain a final marking that enables any transition in $inner(S)$.

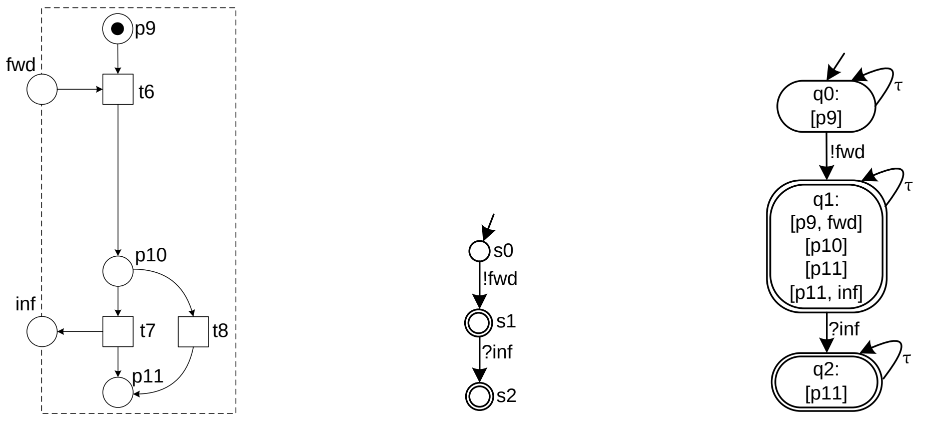
We will show that strict termination can be incorporated into the algorithm for constructing the most permissive X_3-strategy of N (Section 4.5.1). Hence, no further representation has to be introduced. Finally, we discuss some related work (Section 4.5.2).

4.5.1. Restricting the most permissive X_3-strategy to strict termination

The aim of strict termination is to exclude X_3-strategies of an open net N that can execute any transition in a final state. To represent strictly terminating X_3-strategies, we have to adapt the most permissive X_3-strategy $MP_{X_3}(N)$ of N as follows. Consider a state q_K of $MP_{X_3}(N)$ that contains a marking m in its knowledge, where $m \in \Omega_N$ and state (m, q_K) has no successor state in $N \oplus MP_{X_3}(N)$. Let q_K also contain a marking m' such that a transition of $MP_{X_3}(N)$ is enabled at state (m', q_K) in $N \oplus MP_{X_3}(N)$. This scenario corresponds exactly to a situation where an X_3-strategy is in a final state, but it may also be in a state where it can send or receive. In this case, we treat m like a deadlock, because no service can satisfy weak termination and strict termination in such a state.

Definition 4.5.1 (restricting X_3-OGs to strict termination).
Let $MP_{X_3}(N)$ be the most permissive X_3-strategy of an open net N. Proceed like in Definition 4.3.1, but remove each state $q_K \in Q_{MP}$, where $m, m' \in K$ such that $m \in \Omega_N$, (m, q_K) does not enable any transition, and $(m', q_K) \xrightarrow{*} (m'', q_{K'})$ with $q_K \neq q_{K'}$. The resulting service automaton is the *most permissive X-strategy $MP_X(N)$ of N*, for $X = \{$weak termination, strict termination$\}$. ⌟

The algorithm iteratively removes all states that violate strict termination. As $MP_{X_3}(N)$ contains only finitely many states, the algorithm will eventually terminate.



(a) Credit (b) $SA(\mathsf{Cust}')$ (c) $MP_{X_3}(\mathsf{Credit})$

Figure 4.16.: Customer Cust′ (b) is an X_3-strategy of Credit (a). Credit is not {weak termination, strict termination}-controllable.

Lemma 4.5.2 (transformation preserves all *X*-strategies).
For an open net N, let $MP_X(N)$ be constructed according to Definition 4.5.1. Open net N is X-controllable iff $Q_{MP} \neq \emptyset$. ⌟

Figure 4.16(a) illustrates the credit service from our example contract in Figure 2.7. A possible X_3-strategy of this service must be aware that after having sent a request, the service may not be interested, in which case no information will be sent to the customer. Nevertheless, the customer cannot be sure, and hence he must be able to receive the information of the credit service. Customer $SA(\mathsf{Cust}')$ in Figure 4.16(b) is an X_3-strategy of Credit; however, it violates strict termination, because it can receive an information message in the final state s1.

To illustrate the algorithm of Definition 4.5.1 consider Figure 4.16(c) depicting the most permissive X_3-strategy of Credit (annotated with the knowledge that it has of Credit). The state q1 contains a marking $[\mathsf{p11}] \in \Omega_{\mathsf{Credit}}$ and a marking $[\mathsf{p11}, \mathsf{inf}]$ with $([\mathsf{p11}, \mathsf{inf}], \mathsf{q1}) \xrightarrow{?\mathsf{inf}} ([\mathsf{p11}], \mathsf{q2})$ in $\mathsf{Credit} \oplus MP_{X_1}(\mathsf{Credit})$. According to Definition 4.5.1, we have to treat marking [p11] like a deadlock, and hence we remove q1. That way, also q0 has to be removed, because no final state is reachable from $([\mathsf{p9}], \mathsf{q0})$. Hence, service Credit is not X-controllable, for $X =$ {weak termination, strict termination}.

As $X_4(Y)$-operating guidelines use the most permissive X_3-strategy, the results of this section can be directly applied to this notion. Consequently, for $X = X_3, X_4, X_4(Y)$ we can strengthen the finite representation of X-strategies to strictly terminating X-strategies, and the resulting representation is still an X-operating guideline.

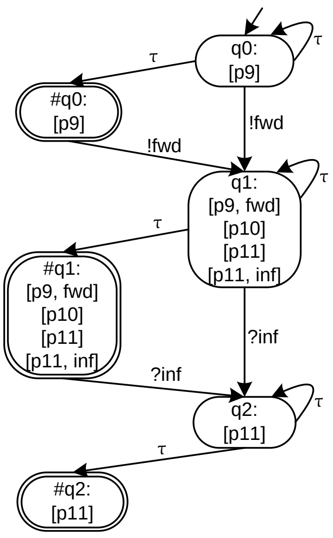


Figure 4.17.: The overapproximation of most permissive X_1-strategy of Credit according to the algorithm in [Wol09]. The empty state is not shown for purposes of simplification.

4.5.2. Discussion

Another approach to remove states that violate strict termination from the most permissive X_3-strategy $MP_{X_3}(N)$ of N has been proposed by Wolf [Wol09]. The approach also starts with the calculation of the most permissive X_1-strategy $MP_{X_1}(N)$ of N. For each state q of $MP_{X_1}(N)$, an explicit final state $\#q$ and a transition $q \xrightarrow{\tau} \#q$ is added during the construction of $MP_{X_1}(N)$. Then, $MP_{X_1}(N)$ is transformed into the most permissive X_3-strategy $MP_{X_3}(N)$ of N. If $MP_{X_3}(N)$ is restricted with respect to strict termination, each state $\#q$ is removed that has an outgoing transition.

Figure 4.17 shows an overapproximation of the most permissive X_1-strategy of Credit according to the approach of [Wol09]. Note that in [Wol09] the notion of responsiveness is used. Hence, there is no state q3. States #q0, #q1, and #q2 model final states. The most permissive X_1-strategy of Credit is obtained from Figure 4.17 by removing the state #q0, as [p9] is not a final marking in Credit. The resulting service automaton is also the most permissive X_3-strategy of Credit. Restriction to strict termination results in removing the state #q1, because it has an outgoing transition (i. e., final states must not be left). As a consequence, there is a marking [p11] in the state q1 from which the only final marking [p11] in the final state #q2 is not reachable. So state q1, and therefore the complete service automaton is removed.

We introduced a different algorithm for constructing the most permissive X_1-

strategy, because in [Wol09] a more restrictive notion of a final state is used, where only receiving transitions may be enabled in a final marking. In contrast, we do not put any restriction on the definition of a final marking (cf. Definition 2.3.1). Furthermore, explicit final states $\#q$ cause some overhead and are not mandatory if no restrictions are put on the final states of a service.

Strict termination has not been implemented in the service analysis tool Fiona so far. Thus, we cannot provide any experimental results.

Part III.

Deciding Substitutability Criteria

5. Deciding Substitutability Under Conformance

In Section 3.1, we proposed the notion of X-conformance as a substitutability criterion in the context of multiparty contracts. An open net N' X-conforms to an open net N if every X-strategy of N is also an X-strategy of N'.

This chapter presents an approach to decide X-conformance of any two open nets N and N'. Therefore, we have a method to check whether N' substitutes N under X-conformance.

Deciding X-conformance of open nets N and N' is a nontrivial problem, because we have to compare the two (in general) *infinite* sets $\mathit{Strat}_X(N)$ and $\mathit{Strat}_X(N')$ of X-strategies. However, $\mathit{Strat}_X(N)$ and $\mathit{Strat}_X(N')$ can be represented in a finite manner as shown in Chapter 4.

The idea is to use the corresponding finite representations of $\mathit{Strat}_X(N')$ and $\mathit{Strat}_X(N)$ to decide whether N' X-conforms to N. To this end, we present, for each $X \in \{X_1, X_2(Y)\}$, a preorder on X-operating guidelines and prove that this preorder coincides with the X-conformance preorder. The preorder on X-operating guidelines is then used to construct an algorithm to decide whether N' substitutes N under X-conformance; see also Figure 1.4. Currently we do not have a solution for extending the results from $X_2(Y)$ to X_2 and for constructing a preorder on finite representations based on state-space fragments (i. e., X_3-operating guidelines, X_4-operating guidelines, and $X_4(Y)$-operating guidelines).

In the remainder of this chapter, we present a decision algorithm for X_1-conformance (Section 5.1) and for $X_2(Y)$-conformance (Section 5.2). The results of Section 5.1 have been published in [ALM$^+$09, SMB09]. Section 5.2 is based on [SW09a].

5.1. Deciding conformance in case of deadlock freedom

In this section, we present an algorithm to decide X_1-conformance of any two open nets N and N'. This algorithm enables us to prove that in our example contract in Figure 2.7 the private view Credit$'$ of the credit service can substitute its public view Credit without violating deadlock freedom in the overall contract. Based on a prototype implementation of this decision algorithm in the service analysis tool Fiona, we experiment with a number of real-life service models.

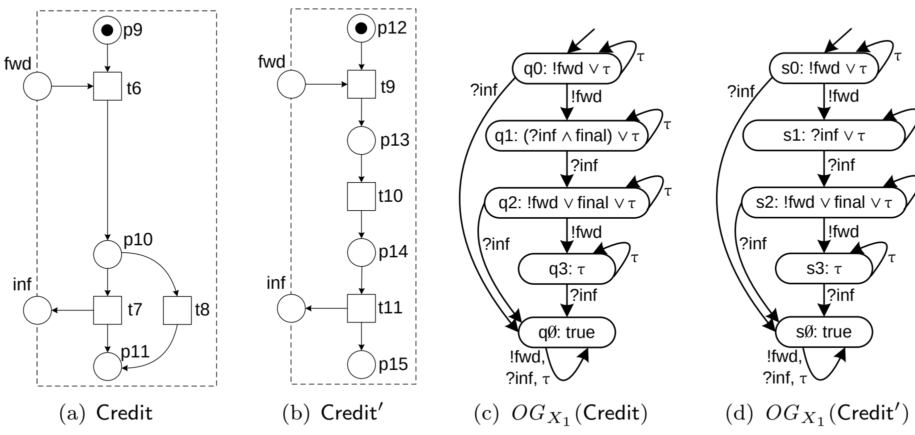


Figure 5.1.: $OG_{X_1}(\mathsf{Credit}')$ refines $OG_{X_1}(\mathsf{Credit})$.

5.1.1. Refinement of X_1-operation guidelines

To avoid a comparison of the in general infinite sets $\mathit{Strat}_{X_1}(N)$ and $\mathit{Strat}_{X_1}(N')$ of X_1-strategies, we use the finite representations of these sets, the X_1-operating guidelines $OG_{X_1}(N)$ and $OG_{X_1}(N')$, for deciding X_1-conformance. To this end, we define a *refinement relation* $\preceq$ on X_1-operating guidelines. Informally, $OG_{X_1}(N')$ refines $OG_{X_1}(N)$ if and only if there is a simulation relation of the states of $OG_{X_1}(N)$ by the states of $OG_{X_1}(N')$ such that the annotations in $OG_{X_1}(N)$ imply the annotations in $OG_{X_1}(N')$.

Definition 5.1.1 (refinement of X_1-operating guidelines).
Let N and N' be any interface equivalent open nets, and let $OG_{X_1}(N) = (Q, MC^I, MC^O, \delta, q_0, \phi)$ and $OG_{X_1}(N') = (Q', MC^I, MC^O, \delta', q_0', \phi')$ be the corresponding X_1-operating guidelines. $OG_{X_1}(N')$ *refines* $OG_{X_1}(N)$, denoted by $OG_{X_1}(N') \preceq OG_{X_1}(N)$, iff

1. there is a minimal simulation relation $\varrho \subseteq Q \times Q'$; and
2. for all $(q, q') \in \varrho$, the formula $\phi(q) \Rightarrow \phi'(q')$ is a tautology. ⌟

Figure 5.1 shows the public and the private view of the credit service and their corresponding X_1-operating guidelines. The only difference between the two X_1-operating guidelines is the annotation in states q1 and s1. A possible X_1-strategy of Credit must be aware that after having sent message fwd, Credit may not be interested, in which case no information will be sent to the customer. Nevertheless, the customer cannot be sure, and hence he must be able to receive the information of Credit. In other words, X_1-strategy of Credit must be in a final state and, in addition, it must be able to receive message inf. In contrast, the public view Credit' will always send information to the customer. Hence, a customer of Credit' must only be able to receive message inf.

It is easy to see that the automaton underlying the private view's X_1-operating guideline $OG_{X_1}(\mathsf{Credit}')$ in Figure 5.1(d) simulates the automaton underlying the public view's X_1-operating guideline $OG_{X_1}(\mathsf{Credit})$ in Figure 5.1(c). Additionally, the implication of an annotation of a state in $OG_{X_1}(\mathsf{Credit})$ and the annotation of the corresponding state in $OG_{X_1}(\mathsf{Credit}')$ is a tautology. For instance, for $\phi(\mathsf{q1}) = (?\mathsf{inf} \wedge \mathit{final}) \vee \tau$ and $\phi'(\mathsf{s1}) = ?\mathsf{inf} \vee \tau$, the formula $\phi(\mathsf{q1}) \Rightarrow \phi'(\mathsf{s1})$ is a tautology. Therefore, we conclude that $OG_{X_1}(\mathsf{Credit}') \preceq OG_{X_1}(\mathsf{Credit})$.

As a counterexample, $OG_{X_1}(\mathsf{Credit})$ does not refine $OG_{X_1}(\mathsf{Credit}')$, because $\phi'(\mathsf{s1}) = ?\mathsf{inf} \vee \tau$ does not imply $\phi(\mathsf{q1}) = (?\mathsf{inf} \wedge \mathit{final}) \vee \tau$: Being in related states $(\mathsf{s1}, \mathsf{q1}) \in \varrho$, for an X_1-strategy of Credit' it is sufficient to receive message inf, whereas an X_1-strategy of Credit must additionally be in a final state.

Intuitively, the first condition in Definition 5.1.1 ensures that every open net that is simulated by the most permissive X_1-strategy $MP_{X_1}(N)$ of N is simulated by the most permissive X_1-strategy $MP_{X_1}(N')$ of N'. The second condition in Definition 5.1.1 guarantees that whenever an open net deadlocks with N', it deadlocks with N as well. With the help of the next theorem, we prove that N' X_1-conforms to N if and only if $OG_{X_1}(N')$ refines $OG_{X_1}(N)$. As a consequence, refinement of X_1-operating guidelines can be used to decide X_1-conformance of N and N'. This result has been first introduced in [ALM+09] for acyclic open nets and has been extended to cyclic open nets in [SMB09]. In [Mas09], this theorem has been proved for arbitrary annotated automata, which are a generalization of X_1-operating guidelines.

Theorem 5.1.2 (checking X_1-conformance).
For any two open nets N and N' with X_1-operating guidelines $OG_{X_1}(N)$ and $OG_{X_1}(N')$ holds:

$$N' \sqsubseteq_{\mathit{conf},X_1} N \text{ iff } OG_{X_1}(N') \preceq OG_{X_1}(N) \quad . \qquad \lrcorner$$

Proof.
Let $OG_{X_1}(N) = (Q, MC^I, MC^O, \delta, q_0, \phi)$ and $OG_{X_1}(N') = (Q', MC^I, MC^O, \delta', q_0', \phi')$ be the X_1-operating guidelines of open nets N and N', respectively.

($\Rightarrow$): Let $N' \sqsubseteq_{\mathit{conf},X_1} N$. We have to prove that $OG_{X_1}(N') \preceq OG_{X_1}(N)$; that is, (1) there is a minimal simulation relation $\varrho_{OG(N),OG(N')} \subseteq Q \times Q'$ such that, (2) for all $(q, q') \in \varrho_{OG(N),OG(N')}$, the formula $\phi(q) \Rightarrow \phi'(q')$ is a tautology.

(1): Consider the service automaton $A = (Q, MC^I, MC^O, \delta, q_0, \Omega)$ with $\Omega = \{q \mid \mathit{final} \text{ occurs in } \phi(q)\}$, which is by Definition 4.1.10 the most permissive X_1-strategy of N, and $PN(A)$ is by Lemma 4.1.6 an X_1-strategy of N. Thus, by $N' \sqsubseteq_{\mathit{conf},X_1} N$, $PN(A)$ is an X_1-strategy of N'. Being an X_1-strategy of N', there is a minimal simulation relation $\varrho_{A,OG(N')} \subseteq Q \times Q'$ of the states of A by the states of $OG_{X_1}(N')$, and hence there is a minimal simulation relation $\varrho_{OG(N),OG(N')} \subseteq Q \times Q'$ of the states of $OG_{X_1}(N)$ by the states of $OG_{X_1}(N')$.

(2): We show that, for all $(q_1, q_1') \in \varrho_{OG(N),OG(N')}$, $\phi(q_1) \Rightarrow \phi'(q_1')$ is a tautology. Let $q_1 \in Q$, and let β be an arbitrary assignment to literals occurring in $\phi(q_1)$ with $\beta(q_1) \models \phi(q_1)$. Remove from A all transitions (q_1, x, q_2) where $\beta(q_1)(x)$ is *false*. By Definition 4.1.4 (matching), the corresponding open net of the resulting service automaton is still an X_1-strategy of N and thus by assumption an X_1-strategy of N'. Using Definition 4.1.4 again, we can see that β satisfies $\phi'(q_1')$ as well. Thus, $\phi(q_1) \Rightarrow \phi'(q_1')$ is a tautology, for all $(q_1, q_1') \in \varrho_{OG(N),OG(N')}$.

Hence, we conclude $OG_{X_1}(N') \preceq OG_{X_1}(N)$.

($\Leftarrow$): Let $OG_{X_1}(N') \preceq OG_{X_1}(N)$, and let $S \in \mathit{Strat}_{X_1}(N)$ be an arbitrary X_1-strategy of N. What remains to be proved is that S is an X_1-strategy of N'.

Because of $S \in \mathit{Strat}_{X_1}(N)$ there is by Definition 4.1.4 (matching) a minimal simulation relation $\varrho_{SA(S),OG(N)} \subseteq Q_{SA(S)} \times Q$ of the states of $SA(S)$ by the states of $OG_{X_1}(N)$, and because of $OG_{X_1}(N') \preceq OG_{X_1}(N)$ there is by Definition 5.1.1 a minimal simulation relation $\varrho_{OG(N),OG(N')} \subseteq Q \times Q'$ of the states of $OG_{X_1}(N)$ by the states of $OG_{X_1}(N')$. Define a relation $\varrho_{SA(S),OG(N')} \subseteq Q_{SA(S)} \times Q'$ of the states of $SA(S)$ by the states of $OG_{X_1}(N')$ such that $(q_S, q') \in \varrho_{SA(S),OG(N')}$ if and only if there is a state q of $OG_{X_1}(N)$ such that $(q_S, q) \in \varrho_{SA(S),OG(N)}$ and $(q, q') \in \varrho_{OG(N),OG(N')}$. We show that (1) $\varrho_{SA(S),OG(N')}$ is a minimal simulation relation, and (2) for each $(q_S, q') \in \varrho_{SA(S),OG(N')}$: $\beta_{SA(S)}(q_S) \models \phi'(q')$.

(1): Clearly, the initial states of $SA(S)$ and $OG_{X_1}(N')$ are related—that is, $(q_{0S}, q_0') \in \varrho_{SA(S),OG(N')}$. As $\varrho_{SA(S),OG(N)}$ and $\varrho_{OG(N),OG(N')}$ are simulation relations, and simulation is transitive, we conclude that $\varrho_{SA(S),OG(N')}$ is a simulation relation as well. Because of $OG_{X_1}(N)$ and $OG_{X_1}(N')$ being deterministic (by Lemma 4.1.12), $\varrho_{SA(S),OG(N')}$ is a minimal simulation relation.

(2): We show, for all $(q_S, q') \in \varrho_{SA(S),OG(N')}$, q_S satisfies $\phi'(q')$. As S matches with $OG_{X_1}(N)$, for all states q_S with $(q_S, q) \in \varrho_{SA(S),OG(N)}$, q_S satisfies $\phi(q)$ (for the assignment described in Definition 4.1.4). By $OG_{X_1}(N') \preceq OG_{X_1}(N)$, we know $\phi(q) \Rightarrow \phi'(q')$, for all $(q, q') \in \varrho_{OG(N),OG(N')}$ (cf. Definition 5.1.1). Hence, q_S satisfies $\phi(q')$, for all $(q_S, q') \in \varrho_{SA(S),OG(N')}$.

Thus, S is an X_1-strategy of N', and we conclude $N' \sqsubseteq_{\mathit{conf},X_1} N$. $\square$

Note that in the proof we make use of the assumption that X_1-operating guidelines are in normal form: For X_1-operating guidelines being not in normal form, the corresponding open net of service automaton A is not necessarily an X_1-strategy (see Lemma 4.1.6).

Based on the preceding considerations, we conclude for our example that the private view Credit' of the credit service X_1-conforms to its pubic view Credit. Hence, Credit' is a valid implementation of Credit, and hence Credit' can substitute Credit without affecting deadlock freedom of the overall contract in Figure 2.7. As for acyclic and bounded open nets deadlock freedom and weak termination coincide (cf. Lemma 2.6.3), we conclude that Credit' X_3-conforms to Credit. However, Credit does not X_1-conform to Credit'. In other words, the private view introduces new X_1-strategies. An X_1-strategy of Credit' can be sure that it will eventually receive an information message, which was not the case for the public view Credit.

The value of Theorem 5.1.2 is that X_1-conformance can be checked independently of the X_1-strategies of N, and only open nets N and N' have to be known to decide X_1-conformance.

Based on the equivalence result of Theorem 5.1.2, we conclude that the preorder property and the precongruence result of X_1-conformance, shown by Lemma 3.1.2 and Lemma 3.1.3, respectively, also hold for the refinement relation on X_1-operating guidelines.

Corollary 5.1.3 (properties of $\preceq$).
Relation $\preceq$ on X_1-operating guidelines is a preorder and a precongruence with respect to $\oplus$. ⌟

This corollary shows that with the refinement relation on X_1-operating guidelines we have found an internal preorder for the external preorder X_1-conformance. That means, the X_1-conformance preorder relates open nets N and N' according to their X_1-strategies, whereas the preorder $\preceq$ on X_1-operating guidelines is based on the respective internal characteristics of $OG_{X_1}(N)$ and $OG_{X_1}(N')$.

In the next step, we extend the refinement relation on X_1-operating guidelines of Definition 5.1.1 to an equivalence relation. So if an X_1-operating guideline $OG_{X_1}(N')$ refines an X_1-operating guideline $OG_{X_1}(N)$ and vice versa, we say that both X_1-operating guidelines are equivalent.

Definition 5.1.4 (equivalence of X_1-operating guidelines).
Any two X_1-operating guidelines $OG_{X_1}(N)$ and $OG_{X_1}(N')$ are *equivalent* iff $OG_{X_1}(N) \preceq OG_{X_1}(N')$ and $OG_{X_1}(N') \preceq OG_{X_1}(N)$. ⌟

The two X_1-operating guidelines in Figure 5.1 are not equivalent, because $OG_{X_1}(\mathsf{Credit})$ does not refine $OG_{X_1}(\mathsf{Credit}')$.

To check X_1-conformance equivalence of two open nets, we can check equivalence of their respective X_1-operating guidelines. As X_1-conformance equivalence means X_1-conformance in both directions, we apply Theorem 5.1.2 in both directions.

Lemma 5.1.5 (checking X_1-conformance equivalence).
Any two open nets N and N' are X_1-conformance equivalent iff their corresponding X_1-operating guidelines are equivalent. ⌟

Using that X-conformance equivalence is an equivalence relation (see Lemma 3.1.6) and Lemma 5.1.5, we immediately conclude that the equivalence of X_1-operating guidelines is indeed an equivalence relation.

Corollary 5.1.6 (equivalence relation for X_1-operating guidelines).
Equivalence on X_1-operating guidelines is an equivalence relation. ⌟

5.1.2. Implementation of the X_1-conformance check in Fiona

For an implementation of the criteria in Theorem 5.1.2, finding the relation ϱ is the crucial task. As both $OG_{X_1}(N)$ and $OG_{X_1}(N')$ are deterministic, this task actually amounts to a depth-first search through $OG_{X_1}(N)$, which is mimicked in $OG_{X_1}(N')$. The time and space required for finding ϱ is thus linear in the product of the number of states and edges of $OG_{X_1}(N)$ and $OG_{X_1}(N')$. This size, in turn, is equal to the product of the number of states and edges of the most permissive X_1-strategies of N and N', respectively.

The X_1-conformance check based on Theorem 5.1.2 has been implemented in the service analysis tool Fiona [MW08]. Based on this implementation and the example services we used in the experiments in Chapter 4, we present some experimental results.

For this experiment, we computed for each open net N its X_1-operating guideline $OG^R_{X_1}(N)$ (remember, Fiona implements responsiveness instead of deadlock freedom). To get a meaningful X_1-operating guideline that can be checked for refinement with $OG^R_{X_1}(N)$, we computed the minimal X_1-operating guideline $(OG^R_{X_1}(N))^{min}$ of N. The minimal X_1-operating guideline of N [Mas09] is the smallest annotated automaton that represents the X_1-strategies of N. Hence, by construction, $OG^R_{X_1}(N)$ and $(OG^R_{X_1}(N))^{min}$ are equivalent.

Table 5.1 provides information about the size of the X_1-operating guideline of N (number of states and transitions), the size of the minimal X_1-operating guideline of N, and the time for checking refinement of the two X_1-operating guidelines. The experiment illustrates that the refinement check is very efficient. In all examples, it took at most 7 seconds.

5.2. Deciding conformance in case of deadlock freedom and cover

In this section, we present an algorithm to decide whether the set of {deadlock freedom, cover(Y)}-strategies of an open net N is contained in the set of {deadlock freedom, cover(Z)}-strategies of an open net N'. In Chapter 4, we denoted these sets of open-net properties by $X_2(Y)$ and $X_2(Z)$, respectively. In other words, we decide whether N' $(X_2(Y), X_2(Z))$-conforms to N. As in the previous section, we develop an internal preorder on the finite representations of $X_2(Y)$-strategies of N and of $X_2(Z)$-strategies of N'. We will prove that this internal preorder coincides with the external $(X_2(Y), X_2(Z))$-conformance preorder of Definition 3.1.1.

In the rest of this section, we develop three conditions—similar to those of Theorem 5.1.2—that enable us to decide whether N' $(X_2(Y), X_2(Z))$-conforms to N. To this end, we start in Section 5.2.1 with taking care of the structure and the local annotations of the $X_2(Y)$-operating guideline of N and the $X_2(Z)$-operating guideline of N'. Afterwards, in Section 5.2.2, we concern with checking the global

Table 5.1.: Deciding X_1-conformance with Fiona. All experiments were obtained on an UltraSPARC III processor with 900MHz and 4 GB RAM running Solaris 10.

	$MP^R_{X_1}(N)$		$(MP^R_{X_1}(N))^{min}$		time
Service	$\lvert Q\rvert$	$\lvert\delta\rvert$	$\lvert Q\rvert$	$\lvert\delta\rvert$	sec
Loan Approval	8	30	8	30	0
Purchase Order	169	1,182	169	1,182	0
Olive Oil Ordering	17	67	13	51	0
Travel Service 1	57	300	57	300	0
Travel Service 2	289	2,252	289	2,252	1
Online Shop 1	13	50	10	41	0
Online Shop 2	8	35	7	32	0
Beverage Machine	12	54	12	54	0
Philosophers #3	68	243	27	108	0
Philosophers #5	1,433	8,390	243	1,620	0
SMPT Protocol	1,217	8,408	381	2,822	1
Registration	8	30	7	27	0
Process 1	897	13,548	641	10,012	1
Process 2	1,608	16,445	1,608	16,445	1
Process 3	237	1,437	173	1,133	0
Process 4	6,821	103,165	6,821	103,165	7

constraints. We explain the algorithmic solution and establish its correctness in Section 5.2.3. The cover property violates the symmetry of X-strategies. Hence, the algorithm for deciding $(X_2(Y), X_2(Z))$-conformance cannot be used in the setting of contracts (see also our comment below Theorem 3.1.4). So far we do not have a solution to extend the result to X_2-conformance. This will be discussed in Section 5.2.4.

Let, for the rest of this section, N and N' be interface equivalent open nets. Let $Y \subseteq P_N \cup T_N$ and $Z \subseteq P_{N'} \cup T_{N'}$ be the respective sets of nodes of N and N' to be covered. Let $OG_{X_2(Y)}(N) = (OG_{X_1}(N), \chi)$ be the $X_2(Y)$-operating guideline of N with $OG_{X_1}(N) = (Q, MC^I, MC^O, \delta, q_0, \phi)$ and global constraint χ. Let $OG_{X_2(Z)}(N') = (OG_{X_1}(N'), \chi')$ be the $X_2(Z)$-operating guideline of N' with $OG_{X_1}(N') = (Q', MC^I, MC^O, \delta', q_0', \phi')$ and global constraint χ'. In this setting, $(X_2(Y), X_2(Z))$-conformance of N and N' should guarantee that every $X_2(Y)$-strategy of N is an $X_2(Z)$-strategy of N' as well.

5.2.1. Structure and local annotations

Subsequently, we show that the same conditions as in Theorem 5.1.2—minimal simulation relation and implication of the local annotations—actually hold in presence of global constraints; that is, the global constraints cannot compensate incompatible local annoations.

The next lemma proves if N' $(X_2(Y), X_2(Z))$-conforms to N, then there exists a minimal simulation relation of $OG_{X_2(Y)}(N)$ by $OG_{X_2(Z)}(N')$. The intuition is that the most permissive X_1-strategy of N, which defines the structure of $OG_{X_2(Y)}(N)$, is also an $X_2(Z)$-strategy of N'.

Lemma 5.2.1 (structure).
If $N' \sqsubseteq_{\mathit{conf},(X_2(Y),X_2(Z))} N$, then there is a minimal simulation relation of $OG_{X_2(Y)}(N)$ by $OG_{X_2(Z)}(N')$. ⌟

Proof.
Let $N' \sqsubseteq_{\mathit{conf},(X_2(Y),X_2(Z))} N$, let $MP_{X_1}(N)$ be the most permissive X_1-strategy of N. The structure of $OG_{X_2(Y)}(N)$ is determined by $MP_{X_1}(N)$, which clearly satisfies the global constraint χ as well. That is, $PN(MP_{X_1}(N)) \in \mathit{Strat}_{X_1}(N)$ and therefore $PN(MP_{X_1}(N)) \in \mathit{Strat}_{X_1}(N')$. By Definition 4.2.3 (matching), this implies the claimed minimal simulation relation. □

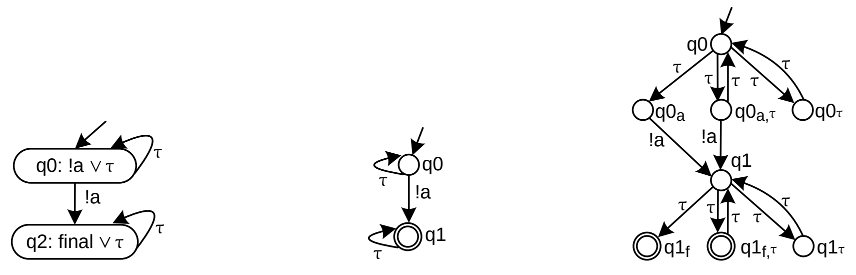
Now we show that $(X_2(Y), X_2(Z))$-conformance of N by N' also implies that the local annotation in $OG_{X_2(Y)}(N)$ imply the local annotations in $OG_{X_2(Z)}(N')$.

Lemma 5.2.2 (local annotations).
If $N' \sqsubseteq_{\mathit{conf},(X_2(Y),X_2(Z))} N$, then, for every $(q, q') \in \varrho_{OG(N),OG(N')}$, the formula $\phi(q) \Rightarrow \phi'(q')$ is a tautology. ⌟

Proof.
Let $N' \sqsubseteq_{\mathit{conf},(X_2(Y),X_2(Z))} N$. We construct from $OG_{X_1}(N)$ the following modification $MP'_{X_1}(N)$ to the most permissive X_1-strategy $MP_{X_1}(N)$ of N. For each state q of $MP_{X_1}(N)$, remove all transitions leaving q. For each satisfying assignment $\beta(q)$ to the propositions of $\phi(q)$, insert a new state q_β and a τ-labeled transition from q to q_β. If $\beta(q)(\mathit{final}) = \mathit{true}$, make q_β a final state. Furthermore, for all transitions (q, a, q^*) in $MP_{X_1}(N)$ with $\beta(q)(a) = \mathit{true}$, add an a-labeled transition from q_β to the former a-successor q^* of q. In addition, if $\beta(q)(\tau) = \mathit{true}$, add an τ-labeled transition from q_β to q. The construction is illustrated in Figure 5.2.

The corresponding open net of the obtained service automaton $MP'_{X_1}(N)$ is by construction an $X_2(Y)$-strategy of N, because every inserted state q_β has successors according to a satisfying assignment of $\phi(q)$. The global constraint χ is satisfied as well, because no state is removed from $MP_{X_1}(N)$, and no state becomes unreachable.

As by assumption N' $(X_2(Y), X_2(Z))$-conforms to N, the corresponding open net of $MP'_{X_1}(N)$ has to be an $X_2(Z)$-strategy of N' as well. Let the minimal simulation relation of $MP'_{X_1}(N)$ by $OG_{X_2(Z)}(N')$ be denoted by $\varrho_{MP',OG(N')}$, and let $\varrho_{OG(N),OG(N')}$ be the minimal simulation relation of $OG_{X_2(Y)}(N)$ (i. e., the underlying automaton of $MP_{X_1}(N)$) by $OG_{X_2(Z)}(N')$. Existence of $\varrho_{OG(N),OG(N')}$



(a) $OG_{X_1}(N)$ (b) $MP_{X_1}(N)$ (c) $MP'_{X_1}(N)$

Figure 5.2.: A modification $MP'_{X_1}(N)$ to $MP_{X_1}(N)$. Consider state $\mathsf{q0}$ of $OG_{X_1}(N)$. For each satisfying assignment for $\phi(\mathsf{q0})$, we add a successor state of $\mathsf{q0}$. State $\mathsf{q0}_{a,\tau}$ denotes a state where $\beta(\mathsf{q0})(\mathsf{a}) = \mathit{true}$ and $\beta(\mathsf{q0})(\tau) = \mathit{true}$. Presence of transition $(\mathsf{q0}, !\mathsf{a}, \mathsf{q1})$ in $OG_{X_1}(N)$ yields a transition $(\mathsf{q0}_{a,\tau}, !\mathsf{a}, \mathsf{q1})$ in $MP'_{X_1}(N)$. In addition, a τ-transition from $\mathsf{q0}_{a,\tau}$ to $\mathsf{q0}$ is added.

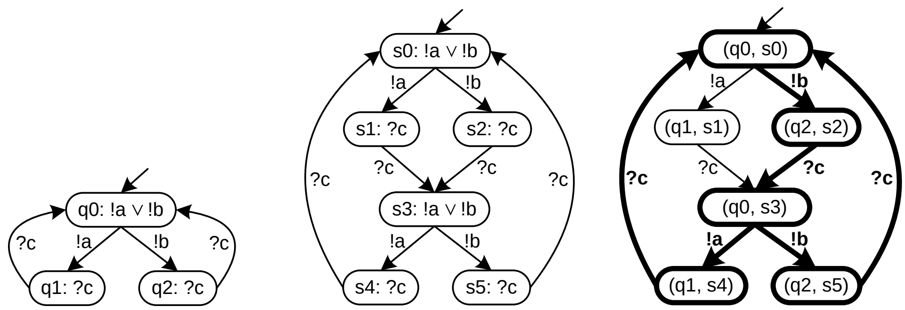
is justified by Lemma 5.2.1. For all q_β with $q \xrightarrow{\tau} q_\beta$ in $MP'_{X_1}(N)$ and $(q, q') \in \varrho_{OG(N),OG(N')}$, we have $(q, q') \in \varrho_{MP',OG(N')}$ and $(q_\beta, q') \in \varrho_{MP',OG(N')}$. As the corresponding open net of $MP'_{X_1}(N)$ is an $X_2(Z)$-strategy of N', the edges leaving q_β, which make assignment $\beta(q_\beta)$, need to satisfy $\phi(q')$. Consequently, the claimed implication holds. □

With Lemma 5.2.1 and Lemma 5.2.2, we justified if every $X_2(Y)$-strategy of N is an $X_2(Z)$-strategy of N', then $OG_{X_2(Y)}(N)$ and $OG_{X_2(Z)}(N')$ satisfy the two conditions of Theorem 5.1.2. In the next subsection, we concern with the global constraint.

5.2.2. Global constraint

In this section, we define the third condition that must hold such that every $X_2(Y)$-strategy of N is an $X_2(Z)$-strategy of N' as well. The remaining third condition specifies when an $X_2(Y)$-strategy of N satisfies the global constraint χ' of $OG_{X_2(Z)}(N')$. Again, as the set of $X_2(Y)$-strategies of N is infinite, we are interested in a *finite* criterion.

Throughout this section, we assume that there is a minimal simulation relation of $OG_{X_2(Y)}(N)$ by $OG_{X_2(Z)}(N')$ and local annotations imply each other as stated in Lemmata 5.2.1 and 5.2.2. Without this assumption, N' would not $(X_2(Y), X_2(Z))$-conform to N, and there would be no reason for considering the global constraints.



(a) $OG_{X_2(Y)}(N)$ with $\chi =$ q1 ∧ q2.

(b) $OG_{X_2(Z)}(N')$ with $\chi' =$ s1 ∧ s5.

(c) $\mathit{lcm}(OG(N), OG(N'))$

Figure 5.3.: Motivation for *lcm*: Although every subautomaton of $OG_{X_2(Y)}(N)$ that satisfies χ (i. e., only the subautomaton having the same structure as $OG_{X_2(Y)}(N)$) can be transformed into an open net that matches with $OG_{X_2(Z)}(N')$, the bold subautomaton in (c) witnesses that $OG_{X_2(Z)}(N')$ does not refine $OG_{X_2(Y)}(N)$. For the sake of simplicity, we omit empty states, τ-labeled selfloops, and τ-literals in the annotations.

A finite generator set for checking the global constraint

We aim at constructing a finite automaton B that satisfies the following condition: If there is an $X_2(Y)$-strategy S of N that violates χ' of $OG_{X_2(Z)}(N')$, then there is also a subautomaton of B—that is, a subgraph of B that contains the initial state—which can be transformed into a service automaton A by making some states final. The corresponding open net $PN(A)$ of A is an $X_2(Y)$-strategy of N and violates χ', too. As B is finite; that is, it has only finitely many states and transitions, it has only finitely many subautomata. Hence, we reduce an infinite check to a finite one.

The apparent candidate for the desired automaton B would be the most permissive X_1-strategy $MP_{X_1}(N)$ of N. However, as Figure 5.3 explains, this does not work. In essence, the problem with $MP_{X_1}(N)$ is that there may be several states q' of $OG_{X_2(Z)}(N')$ related to a single state q of $MP_{X_1}(N)$. This problem can be avoided by constructing B differently.

The idea is to use the simulation relation $\varrho_{OG(N),OG(N')}$ of $OG_{X_2(Y)}(N)$ by $OG_{X_2(Z)}(N')$ as the set of states. It then turns out that simulations of $OG_{X_2(Y)}(N)$ by B as well as of $OG_{X_2(Z)}(N')$ by B are very regularly structured. We call the constructed automaton B the *least common multiple* and denote it by *lcm*. This name is inspired by the observation that if $OG_{X_2(Y)}(N)$ is a simple loop of length m and $OG_{X_2(Z)}(N')$ a simple loop of length n, then B would have exactly $\mathit{lcm}(m, n)$ states.

Definition 5.2.3 (least common multiple).
Let A and A' be deterministic automata with alphabets $MC^I_A = MC^I_{A'}$ and $MC^O_A = MC^O_{A'}$. Let there be a minimal simulation relation ϱ of A by A'. The *least common multiple* of A and A', $lcm(A, A') = (Q_{lcm}, MC^I_{lcm}, MC^O_{lcm}, \delta_{lcm}, {q_0}_{lcm})$, is the automaton defined by

- $Q_{lcm} = \{(q, q') \mid (q, q') \in \varrho\}$;
- $MC^I_{lcm} = MC^I_A \wedge MC^O_{lcm} = MC^O_A$;
- $\delta_{lcm} = \{((q_1, q'_1), x, (q_2, q'_2)) \mid \exists x : (q_1, q'_1), (q_2, q'_2) \in \varrho \wedge (q_1, x, q_2) \in \delta_A \wedge (q'_1, x, q'_2) \in \delta_{A'}\}$; and
- ${q_0}_{lcm} = (q_0, q'_0)$. ⌟

The least common multiple $lcm(A, A')$ is an automaton whose states are given by the minimal simulation relation of A by A'. So by construction, every transition of $lcm(A, A')$ can be mimicked by A and by A'. Automaton $lcm(A, A')$ is further interface equivalent to A and A'.

The least common multiple is an automaton rather than a service automaton, because the underlying structure of an $X_2(Y)$-operating guideline is an automaton. For the same reason, the restriction to deterministic automata does not harm. As by Lemma 5.2.1 $OG_{X_2(Z)}(N')$ simulates $OG_{X_2(Y)}(N)$, $lcm(OG_{X_2(Y)}(N), OG_{X_2(Z)}(N'))$ is the synchronous product [HU79] of the underlying automata.

As an example, the least common multiple of $OG_{X_2(Y)}(N)$ and $OG_{X_2(Z)}(N')$ in Figures 5.3(a) and 5.3(b), respectively, is illustrated in Figure 5.3(c).

The least common multiple $lcm(A, A')$ has some important properties. First, for all states (q_1, q'_1), (q_2, q'_2) with $(q_1, q'_1) \xrightarrow{x} (q_2, q'_2)$ holds that states q_2 and q'_2 are uniquely determined by states q_1, q'_1 and transition x, because A and A' are by definition deterministic. Second, $q_1 \xrightarrow{x} q_2$ implies $q'_1 \xrightarrow{x} q'_2$, because there is a minimal simulation relation of A by A'. Third, there are three simulation relations: A simulates $lcm(A, A')$, denoted by $\varrho_{lcm(A,A'),A}$; A' simulates $lcm(A, A')$, denoted by $\varrho_{lcm(A,A'),A'}$; $lcm(A, A')$ simulates A, denoted by $\varrho_{A,lcm(A,A')}$. The next lemma proves that the three simulation relations are, in fact, minimal simulation relations.

Lemma 5.2.4 (minimal simulation relation of lcm).
Let A and A' be deterministic automata, and let $\varrho_{lcm(A,A')}$ be the minimal simulation relation of A by A' that is used to construct $lcm(A, A')$. Relations $\varrho_{lcm(A,A'),A} = \{((q, q'), q) \mid (q, q') \in Q_{lcm(A,A')}\}$, $\varrho_{lcm(A,A'),A'} = \{((q, q'), q') \mid (q, q') \in Q_{lcm(A,A')}\}$, and $\varrho_{A,lcm(A,A')} = \{(q, (q, q')) \mid (q, q') \in Q_{lcm(A,A')}\}$ are minimal simulation relations. ⌟

Proof.
Let $(q_0, q'_0) \in \varrho_{lcm(A,A')}$. Let $({q_0}_{lcm(A,A')}, q_0) = ((q_0, q'_0), q_0) \in \varrho_{lcm(A,A'),A}$, $(q_0, {q_0}_{lcm(A,A')}) = (q_0, (q_0, q'_0)) \in \varrho_{A,lcm(A,A')}$, and $({q_0}_{lcm(A,A')}, q'_0) = ((q_0, q'_0), q'_0) \in \varrho_{lcm(A,A'),A'}$ be the respective initial elements.

Let $(q_1, q'_1) \xrightarrow{x} (q_2, q'_2)$ be a transition in $lcm(A, A')$. By construction of $lcm(A, A')$, we have $q_1 \xrightarrow{x} q_2$ and $q'_1 \xrightarrow{x} q'_2$; so $\varrho_{lcm(A,A'),A}$ and $\varrho_{lcm(A,A'),A'}$ are indeed simulation relations. Let the other way around $q_1 \xrightarrow{x} q_2$ be a transition in A and $(q_1, (q_1, q'_1)) \in \varrho_{A,lcm(A,A')}$. As there is a simulation relation of A by A', there is a state q'_2 with $q'_1 \xrightarrow{x} q'_2$ in A', which is uniquely determined, because A' is deterministic. Consequently, transition $(q_1, q'_1) \xrightarrow{x} (q_2, q'_2)$ is present in $lcm(A, A')$ and thus $(q_2, (q_2, q'_2)) \in \varrho_{A,lcm(A,A')}$. So there is also a simulation relation of $lcm(A, A')$ by A. We continue by proving that these simulation relations are minimal.

Presence of $((q_1, q'_1), q_1)$ indeed implies presence of $((q_2, q'_2), q_2)$ in the simulation relation of $lcm(A, A')$ by A, because A is deterministic; so there is no other choice to match x in A. Analogously, presence of $((q_2, q'_2), q'_2)$ and $(q_2, (q_2, q'_2))$ are required in the respective simulation relations. Consequently, all considered simulation relations are minimal. □

As a consequence of Lemma 5.2.4, the minimal simulation relation of $lcm(A, A')$ by A is a bisimulation relation, because $lcm(A, A')$ simulates A and vice versa, and A and A' are deterministic.

Corollary 5.2.5 (bisimulation relation between $lcm(A, A')$ and A).
Let $lcm(A, A')$ be the least common multiple of any two deterministic automata A and A'. There is a bisimulation relation between $lcm(A, A')$ and A. ⌟

Consider again Figure 5.3. $OG_{X_2(Y)}(N)$ in Figures 5.3(a) and $OG_{X_2(Z)}(N')$ in Figure 5.3(b) are simulated by the least common multiple in Figure 5.3(c), and there is a bisimulation between the automata in Figures 5.3(a) and 5.3(c).

Next, we consider the least common multiple $lcm(OG_{X_2(Y)}(N), OG_{X_2(Z)}(N'))$ of the two $X_2(Y)$-operating guidelines of open nets N and N'. The structure of the least common multiple ensures that it simulates every service automaton $SA(S)$, for all $X_2(Y)$-strategies S of N. This is proved by the next lemma.

Lemma 5.2.6 (lcm simulates X_2-strategies of N).
For any $S \in Strat_{X_2(Y)}(N)$, there is a minimal simulation relation of $SA(S)$ by $lcm(OG_{X_2(Y)}(N), OG_{X_2(Z)}(N'))$. ⌟

Proof.
Let $S \in Strat_{X_2(Y)}(N)$ and $lcm = lcm(OG_{X_2(Y)}(N), OG_{X_2(Z)}(N'))$. There is by Definition 4.2.3 (matching) a minimal simulation relation of $SA(S)$ by $OG_{X_2(Y)}(N)$. By Lemma 5.2.4, there is a minimal simulation relation of $OG_{X_2(Y)}(N)$ by lcm. As simulation is transitive and both, $OG_{X_2(Y)}(N)$ and lcm are deterministic, there is a minimal simulation of $SA(S)$ by lcm. □

With $lcm = lcm(OG_{X_2(Y)}(N), OG_{X_2(Z)}(N'))$ we have constructed a finite automaton that simulates every $X_2(Y)$-strategy of N. Now we can show the main claim of this section. Assume that N' does not $(X_2(Y), X_2(Z))$-conform to N.

Then, there is an $X_2(Y)$-strategy S of N, which is not an $X_2(Z)$-strategy of N'. Using the minimal simulation relation of $SA(S)$ by lcm, we construct a service automaton, where the underlying automaton is a subautomaton of lcm. The corresponding open net of this service automaton is an $X_2(Y)$-strategy of N as well, and again no $X_2(Z)$-strategy of N'. We call the constructed service automaton the *image* $Im(SA(S))$ of $SA(S)$ in lcm.

Definition 5.2.7 (image).
Let $S \in Strat_{X_2(Y)}(N)$, and let $lcm = lcm(OG_{X_2(Y)}(N), OG_{X_2(Z)}(N'))$. The *image* $Im(A) = (Q_{Im}, MC^I_{Im}, MC^O_{Im}, \delta_{Im}, q_{0\,Im}, \Omega_{Im})$ of $A = SA(S)$ is the service automaton defined by

- $Q_{Im} = \{(q, q') \mid \exists q_A : (q_A, (q, q')) \in \varrho_{A,lcm}\}$;
- $MC^I_{Im} = MC^I_A \wedge MC^O_{Im} = MC^O_A$;
- $\delta_{Im} = \delta_{lcm} \cap (Q_{Im} \times MC^I_{Im} \cup MC^O_{Im} \cup \{\tau\} \times Q_{Im})$;
- $q_{0\,Im} = q_{0lcm}$; and
- $\Omega_{Im} = \{(q, q') \in Q_{Im} \mid \exists q_A \in \Omega_A : (q_A, (q, q')) \in \varrho_{A,lcm}\}$. ⌟

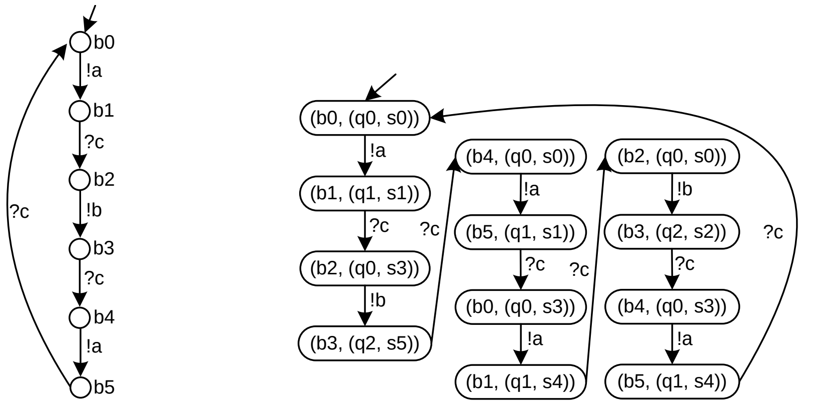
The image $Im(SA(S))$ of $SA(S)$ restricts $lcm(OG_{X_2(Y)}(N), OG_{X_2(Z)}(N'))$ to those states that are contained in the minimal simulation relation with $SA(S)$. That way, infinitely many open nets may have the same image.

As an example, Figure 5.4(a) depicts a service automaton $SA(S)$ of an open net S (not shown) that matches with $OG_{X_2(Y)}(N)$ in Figure 5.3(a). The automaton lcm in Figure 5.3(c) simulates $SA(S)$. The automaton of the minimal simulation relation of $SA(S)$ by lcm is depicted in Figure 5.4(b). The image $Im(SA(S))$ can be derived from Figure 5.4(b). For each related pair (q_S, q_{lcm}), remove the state q_S of $SA(S)$ and fold up the resulting service automaton. As the minimal simulation relation in the example covers every state and every transition of lcm, the resulting image of $SA(S)$ has the structure of the least common multiple in Figure 5.3(c).

To apply the idea of using subautomata $Im(SA(S))$ of $lcm(OG_{X_2(Y)}(N), OG_{X_2(Z)}(N'))$ for our approach, we have to prove that $PN(Im(SA(S)))$ is an X_2-strategy of N. For this proof, we make use of another fact. If S is an $X_2(Y)$-strategy of N, then $SA(S)$ and $Im(SA(S))$ touch the same states of $OG_{X_2(Y)}(N)$ when being simulated by $OG_{X_2(Z)}(N)$. This fact is proved by the next lemma.

Lemma 5.2.8 (A and $Im(A)$ touch the same states in $OG_{X_2(Z)}(N)$).
Let $S \in Strat_{X_2(Z)}(N)$ and $A = SA(S)$. If $(q_A, (q, q')) \in \varrho_{A,Im(A)}$, then $(q_A, q) \in \varrho_{A,OG(N)}$. ⌟

Proof.
For an open net $S \in Strat_{X_2(Y)}(N)$, let $A = SA(S)$, and let $Im(A)$ be the image of A in $lcm = lcm(OG_{X_2(Y)}(N), OG_{X_2(Z)}(N'))$. Let the minimal simulation



(a) $SA(S)$ (b) Minimal simulation relation of $SA(S)$ by $lcm(OG_{X_2(Y)}(N), OG_{X_2(Z)}(N'))$.

Figure 5.4.: Example illustrating the idea of an image.

relation of A by $OG_{X_2(Y)}(N)$ and of A by $Im(A)$ be denoted by $\varrho_{A,OG(N)}$ and $\varrho_{A,Im(A)}$, respectively.

The lemma is obviously true for the initial states. Let $(q_A, (q_1, q_1')) \in \varrho_{A,Im(A)}$, $(q_A, q) \in \varrho_{A,OG(N)}$, and $q_A \xrightarrow{x} q_A'$ in A. By construction of $Im(A)$, there is a minimal simulation relation of A by $Im(A)$. Hence, there exists a state (q_2, q_2') such that $(q_1, q_1') \xrightarrow{x} (q_2, q_2')$ is in $Im(A)$. This implies $(q_A', (q_2, q_2')) \in \varrho_{A,Im(A)}$.

There is a minimal simulation relation of A by $OG_{X_2(Y)}(N)$, because S is by assumption an $X_2(Y)$-strategy of N. So there is a state q_3 in $OG_{X_2(Y)}(N)$, with $q_1 \xrightarrow{x} q_3$. This implies $(q_A', q_3) \in \varrho_{A,OG(N)}$. By construction of lcm, we have $q_2 = q_3$; so the lemma holds. $\square$

With the help of Lemma 5.2.8, we can prove that if S is an $X_2(Y)$-strategy of N, then the corresponding open net $PN(Im(SA(S)))$ of $Im(SA(S))$ is an $X_2(Y)$-strategy of N, too.

Lemma 5.2.9 ($PN(Im(SA(S)))$ is $X_2(Y)$-strategy of N).
For any $X_2(Y)$-strategy S of N holds: $PN(Im(SA(S)))$ is an $X_2(Y)$-strategy of N. ⌟

Proof.
Let $S \in Strat_{X_2(Y)}(N)$, let $lcm = lcm(OG_{X_2(Y)}(N), OG_{X_2(Z)}(N'))$, and let $A = SA(S)$. Let $Im(A)$ be as defined. According to Definition 4.2.3 (matching) we have to show: (1) there is a minimal simulation relation of $Im(A)$ by $OG_{X_2(Y)}(N)$, (2) $Im(A)$ satisfies the local annotations of $OG_{X_2(Y)}(N)$, and (3) $Im(A)$ satisfies the global constraint χ of $OG_{X_2(Y)}(N)$.

(1): The underlying automaton of $Im(A)$ is by Definition 5.2.7 a subautomaton of lcm. As lcm and $OG_{X_2(Y)}(N)$ are by Corollary 5.2.5 bisimilar, a simulation relation of *Im*(A) by $OG_{X_2(Y)}(N)$ clearly exists, and the minimal simulation relation is the restriction of the simulation relation to the states $Q_{Im(A)}$ of *Im*(A).

(2): Now we show that *Im*(A) satisfies the local annotations of $OG_{X_2(Y)}(N)$. Let $((q, q'), q) \in \varrho_{Im(A),OG(N)} = \varrho_{lcm,OG(N)}$. By construction of *Im*(A), there is a state q_A of A where $(q_A, (q, q')) \in \varrho_{A,Im(A)} = \varrho_{A,lcm}$. By Lemma 5.2.8, $(q_A, q) \in \varrho_{A,OG(N)}$. As S is an $X_2(Y)$-strategy of N, the edge labels leaving q_A, with the "final" status of q_A, form a satisfying assignment $\beta_{q_A}(q)$ for $\phi(q)$. As $\varrho_{A,Im(A)}$ is a minimal simulation relation (by Lemma 5.2.6), the edge labels leaving (q, q') include the ones that leave q_A. If the *final* proposition is *true* in state q_A, then it is by Definition 5.2.7 *true* in (q, q'). So the assignment $\beta_{q_{Im(A)}}(q)$, used for checking $\phi(q)$ in (q, q'), assigns true to at least those propositions where the assignment $\beta_{q_A}(q)$ used in q_A is *true*. As $\phi(q)$ is monotonous, $\phi(q)$ is true in (q, q').

(3): By Lemma 5.2.8, $\varrho_{A,OG(N)}$ and $\varrho_{Im(A),OG(N)}$ touch the same states of $OG_{X_2(Y)}(N)$. Consequently, the assignment to χ is the same for both, A and *Im*(A). As A is an $X_2(Y)$-strategy of N, χ evaluates to *true*.

Hence, $PN(Im(A)) \in Strat_{X_2(Y)}(N)$, and the lemma holds. □

Lemma 5.2.9 states that, for each $X_2(Y)$-strategy S of N, there is a service automaton $Im(SA(S))$ whose underlying automaton is a subautomaton in $lcm = lcm(OG_{X_2(Y)}(N), OG_{X_2(Z)}(N'))$, and the corresponding open net of $Im(SA(S))$ is an $X_2(Y)$-strategy of N as well. The other way around, for each subautomaton of *lcm* that can be extended to a service automaton A (by making some states final) such that $PN(A)$ is an $X_2(Y)$-strategy of N, there is trivially an $X_2(Y)$-strategy S of N with $Im(SA(S)) = A$. Consequently, the least common multiple and its subautomata serve as a generator set for $X_2(Y)$-strategies of N.

Recall that the image $Im(SA(S))$ of $SA(S)$ in Figure 5.4(a) has the structure of the least common multiple in Figure 5.3(c). Hence, its corresponding open net matches with $OG_{X_2(Y)}(N)$ in Figure 5.3(a); that is, there is a minimal simulation relation, and $Im(SA(S))$ satisfies the local annotations as well as the global constraint χ of $OG_{X_2(Y)}(N)$.

Finally, we prove that $PN(Im(SA(S)))$ is not an $X_2(Z)$-strategy of N' if S is none. Remember that we assume throughout this section that there is a minimal simulation relation of $OG_{X_2(Y)}(N)$ by $OG_{X_2(Z)}(N')$ and an implication of the local annotations of $OG_{X_2(Y)}(N)$ and of $OG_{X_2(Z)}(N')$. Thus, $OG_{X_2(Z)}(N')$ simulates $SA(S)$, and $SA(S)$ satisfies the local annotations of $OG_{X_2(Z)}(N')$. It is consequently sufficient to show that if $SA(S)$ violates the global constraint χ' of $OG_{X_2(Z)}(N')$, so does $Im(SA(S))$. Using monotonicity of χ', the following lemma is sufficient for establishing this claim. It proves that $SA(S)$ touches at least as many states in $OG_{X_2(Z)}(N')$ as $Im(SA(S))$. Thus, if $Im(SA(S))$ satisfies χ', so does $SA(S)$.

Lemma 5.2.10 (A touches at least as many states as $Im(A)$).
Let S be an $X_2(Y)$-strategy of N, and let $A = SA(S)$. For every state q' of $OG_{X_2(Z)}(N')$ holds: If there is a state $q_{Im(A)}$ of $Im(A)$ such that $(q_{Im(A)}, q') \in$

$\varrho_{Im(A),OG(N')}$, then there is a state q_A such that $(q_A, q') \in \varrho_{A,OG(N')}$. ⌟

Proof.
Let $S \in Strat_{X_2(Y)}(N)$, let $lcm = lcm(OG_{X_2(Y)}(N), OG_{X_2(Z)}(N'))$, and let $A = SA(S)$. Let $Im(A)$ be as defined. We argue by induction on the construction of $\varrho_{A,Im(A)} = \varrho_{A,lcm}$.

By construction of $Im(A)$, for all $q_{Im(A)}$ of $Im(A)$, there is a state q_A of A such that $(q_A, q_{Im(A)}) \in \varrho_{A,Im(A)}$. For a state $(q, q') \in Im(A)$, the minimal simulation relation of lcm by $OG_{X_2(Z)}(N')$ (see Lemma 5.2.4) and hence a minimal simulation relation $\varrho_{Im(A),OG(N')}$ of $Im(A)$ by $OG_{X_2(Z)}(N')$ guarantees that $((q, q'), q'_1) \in \varrho_{Im(A),OG(N')}$ implies that $q' = q'_1$. Thus, it remains to show that, for each $(q_A, (q, q')) \in \varrho_{A,Im(A)}$, $(q_A, q') \in \varrho_{A,OG(N')}$. This is obviously true for the initial element $(q_{0A}, (q_0, q'_0))$. Assume that $(q_A, (q, q')) \in \varrho_{A,Im(A)}$ and $(q_A, q') \in \varrho_{A,OG(N')}$. Let $(q'_A, (q_1, q'_1))$ be the element of $\varrho_{A,Im(A)}$ that is inserted due to $(q_A, (q, q'))$ and transition x. Then clearly $q_A \xrightarrow{x} q'_A$ and $(q, q') \xrightarrow{x} (q_1, q'_1)$, which implies by the construction of lcm that $q' \xrightarrow{x} q'_1$ in $OG_{X_2}(N')$. Consequently, $(q'_A, q'_1) \in \varrho_{A,OG(N')}$. □

The result of this lemma enables us to characterize refinement of $X_2(Y)$-operating guidelines.

Refinement of $X_2(Y)$-operating guidelines

In the following, we define refinement of $X_2(Y)$-operating guidelines and prove that this refinement notion can be used to decide $(X_2(Y), X_2(Z))$-conformance of open nets.

With the help of Lemma 5.2.1, of Lemma 5.2.2, and of Lemma 5.2.10, we can characterize refinement of $X_2(Y)$-operating guidelines.

Definition 5.2.11 (refinement of $X_2(Y)$-operating guidelines).
Let N and N' be any interface equivalent open nets, and let $OG_{X_2(Y)}(N) = ((Q, MC^I, MC^O, \delta, q_0, \phi), \chi)$ and $OG_{X_2(Z)}(N') = ((Q', MC^I, MC^O, \delta', q'_0, \phi'), \chi')$ be the corresponding $X_2(Y)$- and $X_2(Z)$-operating guidelines. $OG_{X_2(Z)}(N')$ *refines* $OG_{X_2(Y)}(N)$, denoted by $OG_{X_2(Z)}(N') \preceq OG_{X_2(Y)}(N)$, iff

1. there is a minimal simulation relation $\varrho \subseteq Q \times Q'$;
2. for all $(q, q') \in \varrho$, the formula $\phi(q) \Rightarrow \phi'(q')$ is a tautology; and
3. for all subautomata of $lcm(OG_{X_2(Y)}(N), OG_{X_2(Z)}(N'))$, which can be extended to a service automaton A by making some states final such that $PN(A)$ is in $Strat_{X_2(Y)}(N)$, the global constraint χ' is satisfied. ⌟

The $X_2(Y)$-operating guidelines $OG_{X_2(Y)}(N)$ and $OG_{X_2(Z)}(N')$ in Figure 5.3 satisfy the first two criteria of Definition 5.2.11. To verify the third criterion of Definition 5.2.11, we need to check nine subautomata. The bold subautomaton in Figure 5.3(c) can be extended to a service automaton A by adding the empty set of final states. The corresponding open net of A is an $X_2(Y)$-strategy of N. However, A violates χ', as it does not assign *true* to the state s1. Hence, we conclude that $OG_{X_2(Z)}(N')$ does not refine $OG_{X_2(Y)}(N)$.

The following theorem proves that $OG_{X_2(Z)}(N')$ refines $OG_{X_2(Y)}(N)$ if and only if N' $(X_2(Y), X_2(Z))$-conforms to N. That way, we can decide $(X_2(Y), X_2(Z))$-conformance of any two open nets N and N' on their corresponding $X_2(Y)$-operating guidelines. This result is a generalization of the result in [SW09a], as in [SW09a] open nets were considered, where no sending transition is enabled in any final marking.

Theorem 5.2.12 (checking $(X_2(Y), X_2(Z))$-conformance).
For any two open nets N and N' with $X_2(Y)$-operating guidelines $OG_{X_2(Y)}(N)$ and $OG_{X_2(Z)}(N')$ holds:

$$N' \sqsubseteq_{\mathit{conf},(X_2(Y),X_2(Z))} N \text{ iff } OG_{X_2(Z)}(N') \preceq OG_{X_2(Y)}(N) \quad .$$ ⌟

Proof.
(⇒): Suppose that $N' \sqsubseteq_{\mathit{conf},(X_2(Y),X_2(Z))} N$. Then Lemma 5.2.1 and 5.2.2 establish the first two conditions of Definition 5.2.11, whereas the third one is evident from the assumption.

(⇐): Suppose that $OG_{X_2(Z)}(N') \preceq OG_{X_2(Y)}(N)$. Hence, all three conditions of Definition 5.2.11 hold. Let $S \in \mathit{Strat}_{X_2(Y)}(N)$. What remains to prove is that $S \in \mathit{Strat}_{X_2(Z)}(N')$.

The first and second condition guarantee that there is a minimal simulation relation of $SA(S)$ by $OG_{X_2(Z)}(N')$ and all local annotations are satisfied. The third condition guarantees by Lemma 5.2.9 that $Im(SA(S))$ can be constructed from one of the subautomata (that serve as a starting point for the third item). Consequently, the global constraint χ' is satisfied as well, as otherwise Lemma 5.2.10 would state that the third condition of Definition 5.2.11 was invalid. □

From the earlier considerations and Theorem 5.2.12, we conclude that in our example in Figure 5.3 N' does not $(X_2(Y), X_2(Z))$-conform to N (the two open nets are not shown). Thus, we cannot substitute N by N'.

Based on the equivalence result of Theorem 5.2.12, we conclude that the preorder property and the precongruence result of $(X_2(Y), X_2(Z))$-conformance stated by Lemma 3.1.2 and Lemma 3.1.3, respectively, also hold for the refinement of $X_2(Y)$-operating guidelines.

Corollary 5.2.13 (properties of $\preceq$).
Relation $\preceq$ on $X_2(Y)$-operating guidelines is a preorder and a precongruence with respect to $\oplus$. ⌟

So with the refinement relation on $X_2(Y)$-operating guidelines, we have found an internal preorder for the external preorder $(X_2(Y), X_2(Z))$-conformance.

In the next step, we extend the refinement relation on $X_2(Y)$-operating guidelines of Definition 5.2.11 to an equivalence. If $OG_{X_2(Z)}(N')$ refines $OG_{X_2(Y)}(N)$ and vice versa, we say that both $X_2(Y)$-operating guidelines are *equivalent.*

Definition 5.2.14 (equivalence of $X_2(Y)$-operating guidelines).
An $X_2(Y)$-operating guideline $OG_{X_2(Y)}(N)$ and an $X_2(Y)$-operating guideline $OG_{X_2(Z)}(N')$ are *equivalent* iff $OG_{X_2(Y)}(N) \preceq OG_{X_2(Z)}(N')$ and $OG_{X_2(Z)}(N') \preceq OG_{X_2(Y)}(N)$. ⌟

Clearly, to decide $(X_2(Y), X_2(Z))$-conformance equivalence of two open nets, we can check equivalence of their respective $X_2(Y)$-operating guidelines. As equivalence means $(X_2(Y), X_2(Z))$-conformance in both directions, we apply Theorem 5.2.12 in both directions.

Lemma 5.2.15 (checking $(X_2(Y), X_2(Z))$-conformance equivalence).
Any two open nets N and N' are $(X_2(Y), X_2(Z))$-conformance equivalent iff $OG_{X_2(Y)}(N)$ and $OG_{X_2(Z)}(N')$ are equivalent. ⌟

Using that X-conformance equivalence is an equivalence relation (see Lemma 3.1.6) and Lemma 5.2.15, we immediately conclude that the equivalence of $X_2(Y)$-operating guidelines is indeed an equivalence relation.

Corollary 5.2.16 (equivalence relation on $X_2(Y)$-operating guidelines).
Equivalence on $X_2(Y)$-operating guidelines is an equivalence relation. ⌟

5.2.3. An algorithm for checking $(X_2(Y), X_2(Z))$-conformance

The least common multiple $lcm = lcm(OG_{X_2(Y)}(N), OG_{X_2(Z)}(N'))$ of $OG_{X_2(Y)}(N)$ and $OG_{X_2(Z)}(N')$ is finite, and thus it contains only finitely many subautomata. Consequently, Theorem 5.2.12 already reduces the $(X_2(Y), X_2(Z))$-conformance problem to a finite check. Nevertheless, we propose to implement the verification in a slightly different manner. To this end, we make use of a particular assignment, the *maximal-false assignment.*

Definition 5.2.17 (maximal-false assignment).
A *maximal-false assignment* assigns to each literal of a Boolean formula a Boolean value such that (1) the formula is evaluated to *false* and (2) changing the truth value of any literal from *false* to *true* yields an assignment that satisfies the formula. ⌟

As an example, for χ' in Figure 5.3(b), we have two maximal-false assignments: assigning *false* to s1 (the first) and assigning *false* to s5 (the second).

To verify $(X_2(Y), X_2(Z))$-conformance of two open nets N and N', we start with checking the first two conditions of Definition 5.2.11. This yields, in particular, the minimal simulation relation $\varrho_{OG(N),OG(N')}$, which we need for building the least common multiple lcm. Then, we propose to iterate through all maximal-false assignments that violate the global constraint χ' of $OG_{X_2(Z)}(N')$. For each such maximal-false assignment, proceed as follows:

1. Remove all states (q, q') from *lcm* where *false* is assigned to q' by the maximal-false assignment.
2. Iteratively remove all states (q, q') from *lcm* where the local annotation of q in $OG_{X_2(Y)}(N)$ is violated, even if the respective state is made a final state.
3. For the remaining subautomaton, evaluate the global constraint χ of $OG_{X_2(Y)}(N)$. If it evaluates to *true*, exit with result 'not $(X_2(Y), X_2(Z))$-conformant'; otherwise, continue with the next maximal-false assignment.

This procedure is justified with the monotonicity of the global constraints. The subautomaton, say A, constructed in the second step is the largest subautomaton of *lcm* that satisfies the local annotations of $OG_{X_2(Y)}(N)$ and violates the global constraint χ' of $OG_{X_2(Z)}(N')$ with some assignment less or equal to the considered one. If A violates χ, every other subautomaton of A does, too. Hence, it is sufficient to consider only those subautomata A.

Consider again the example in Figure 5.3. For the first maximal-false assignment (i. e., assigning *false* to `s1`), we remove state (`q1`, `s1`) and its adjacent edges. This yields the subautomaton, depicted bold in Figure 5.3(c). The removal does not violate the local annotations in $OG_{X_2(Y)}(N)$ (2). As the resulting subautomaton satisfies χ, we exit with 'not $(X_2(Y), X_2(Z))$-conformant' (3). Note that starting with the second maximal-false assignment (i. e., assigning *false* to `s5`) yields a subautomaton that satisfies χ and hence proves non-$(X_2(Y), X_2(Z))$-conformance as well.

Discussion

At this stage, we cannot provide experimental results. However, we believe—confirmed by our experiments in Section 4.2.4—that global constraints do not tend to be too large for practical service models; so the number of maximal-false assignments to be considered should be tractable. The construction in the second step of the proposed procedure is part of the standard algorithm for computing most permissive X_1-strategies in Fiona and is known to be quite efficient. Consequently, we believe that the verification of $(X_2(Y), X_2(Z))$-conformance could be tractable in realistic cases even if worst case complexity is beyond the limits of theoretical tractability.

Recently, an alternative approach for deciding $(X_2(Y), X_2(Z))$-conformance has been published. Kaschner and Wolf [KW09] propose to check $\mathit{Strat}_{X_2(Y)}(N) \cap \overline{\mathit{Strat}_{X_2(Z)}(N')} = \emptyset$ instead of the equivalent inclusion $\mathit{Strat}_{X_2(Y)}(N) \subseteq \mathit{Strat}_{X_2(Z)}(N')$. That means, it is verified whether the set of $X_2(Y)$-strategies of N and the set of open nets that are *not* an $X_2(Z)$-strategy of N' are disjoint. To this end, the set $\overline{\mathit{Strat}_{X_2(Z)}(N')}$ is represented as an extended annotated automaton (where the Boolean formulae may contain negated literals) [KW09]. Furthermore, algorithms to compute the intersection and to check for emptiness are presented. Checking for emptiness instead of inclusion has the advantage that in case emptiness does not hold, an element of $Strat_{X_2(Y)}(N) \cap \overline{Strat_{X_2(Z)}(N')}$ serves as a counterexample. As a drawback, the emptiness check is proved to be NP-complete in worst case [KW09].

5.2.4. Towards deciding X_2-conformance

The previously presented algorithm enables us to decide whether an open net N' substitutes an open net N under $(X_2(Y), X_2(Z))$-conformance. Due to the presence of cover in the sets of open-net properties $X_2(Y)$ and $X_2(Z)$, the respective sets of X-strategies are not symmetric. Consequently, the $(X_2(Y), X_2(Z))$-conformance relation cannot be used in the setting of contracts (because Theorem 3.1.4 does not hold). Therefore, it would be valuable to extend the algorithm for deciding $(X_2(Y), X_2(Z))$-conformance to X_2-conformance, a similar notion that can be used in the contract setting. Recall that we denoted with X_2 the set {deadlock freedom, quasi-liveness} of open-net properties.

The X_2-operating guideline of N coincides with the $X_2(Y)$-operating guideline of N with $Y = T_N$. However, the X_2-operating guideline requires a stricter matching criterion than the $X_2(Y)$-operating guideline: An open net S matches with $OG_{X_2}(N)$ if it matches with $OG_{X_2(Y)}(N)$ and, in addition, for every state of the corresponding service automaton $SA(S)$ of S, there exists a state q of $OG_{X_2}(N)$ in the minimal simulation relation ϱ of $SA(S)$ by $OG_{X_2}(N)$ and q is not the empty state of $OG_{X_2}(N)$. As the notions of $X_2(Y)$-operating guideline and X_2-operating guideline are very similar, it seems that the decision algorithm for $(X_2(Y), X_2(Z))$-conformance can be easily adjusted to X_2-conformance. However, in the following we sketch that this is not the case.

Suppose an open net N' X_2-conforms to an open nets N. Clearly, there is a minimal simulation relation ϱ of $OG_{X_2}(N)$ by $OG_{X_2}(N')$. To deal with the additional matching criterion of X_2-operating guidelines, we can ensure that, for every pair of related states $(q, q') \in \varrho$, if q' is the empty state of $OG_{X_2}(N')$, then q is the empty state of $OG_{X_2}(N)$. Hence, if a state of the corresponding automaton $SA(S)$ of an open net S is related to a non-empty state of $OG_{X_2}(N')$ in the minimal simulation relation with $OG_{X_2}(N')$, this state is also related to a non-empty state of $OG_{X_2}(N)$ in the minimal simulation relation with $OG_{X_2}(N)$. Furthermore, the annotations of $OG_{X_2}(N)$ must imply the annotations of $OG_{X_2}(N')$ (see Lemma 5.2.2).

Now let us investigate the third criterion, specifying when an X_2-strategy S of N satisfies the global constraint χ' of $OG_{X_2}(N')$. According to the proof idea in Section 5.2.2, for every $X_2(Y)$-strategy S of N, the image $Im(SA(S))$ of S is an $X_2(Y)$-strategy of N. Figure 5.5 illustrates that this, however, does not hold for X_2-conformance. The example shows an open net N with final states $\Omega = \{[\mathsf{p2}], [\mathsf{p4}]\}$ and the corresponding service automaton $SA(S)$ of an X_2-strategy of N. We can observe that the information that state s1 of $SA(S)$ is not only related to the empty state of $MP^R_{X_2}(\mathsf{N})$ but also to state q3 is not preserved by the image $Im(SA(S))$. As a consequence, $PN(Im(SA(S)))$ is no X_2-strategy of

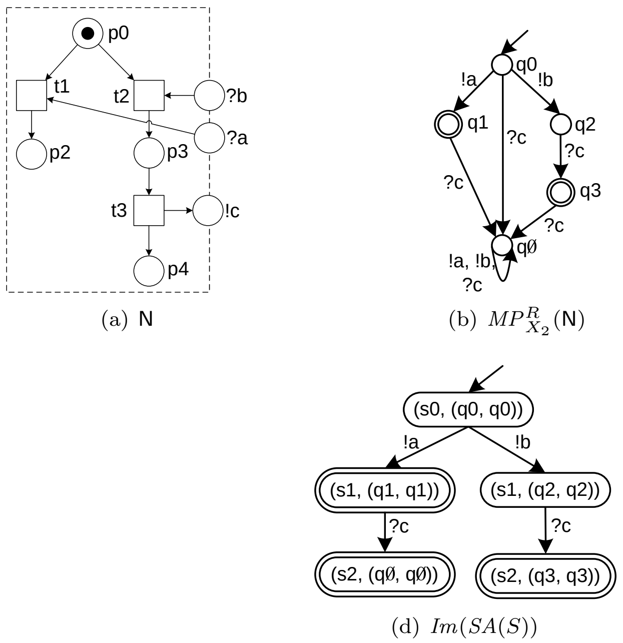


Figure 5.5.: Although $PN(SA(S))$ is an X_2-strategy of N, $PN(Im(SA(S)))$ is not an X_2-strategy of N, thus violating Lemma 5.2.9. For the sake of simplicity, $MP^R_{X_2}(\mathsf{N})$ denotes the most permissive X_2-strategy of N for responsiveness and τ-labeled selfloops are omitted.

N. Thus, for deciding X_2-conformance, we need a proof technique different from the one of Section 5.2.2.

A solution to this problem is part of the future work.

6. Deciding Substitutability Under Preservation

In this chapter, we introduce an approach to decide substitution of an open net N by an open net N' under X-preservation of a set $\mathcal{S} \subseteq \mathit{Strat}_X(N)$ of X-strategies of N. With this approach, we can decide service substitution in the setting of service improvement; see also Figure 1.4.

To decide service substitution under X-preservation, we have to check that every open net $S \in \mathcal{S}$ is an X-strategy of N'. If $\mathcal{S}$ is finite, we can match every open net $S \in \mathcal{S}$ with $OG_X(N')$. For an infinite set $\mathcal{S}$, matching is, however, not feasible. If we assume that $\mathcal{S}$ is represented as an X-operating guideline OG_X, then deciding X-preservation of $\mathcal{S}$ reduces to check that $OG_X(N')$ refines OG_X. That way, we can use an internal preorder on X-operating guidelines to decide substitutability under X-preservation instead of using an external preorder on infinite sets of X-strategies.

The decision procedure for a finite set $\mathcal{S}$ becomes particularly complex if $\mathcal{S}$ contains many services and if we want to check several N'. Therefore, we consider the following alternative: As the notion of an X-strategy is symmetric (if $\text{cover}(Y) \notin X$), it is equivalent to check whether N' is an X-strategy of all $S \in \mathcal{S}$. In other words, $N' \in \bigcap_{S \in \mathcal{S}} \mathit{Strat}_X(S)$ must hold. We show that the intersection $\bigcap_{S \in \mathcal{S}} \mathit{Strat}_X(S)$ of sets of X-strategies can be represented by the *product of the X-operating guidelines* $OG_X(S)$ of all open nets $S \in \mathcal{S}$.

The product can also be used for specifying an infinite set $\mathcal{S}$ of X-strategies of N. To this end, we present an approach to specify *behavioral constraints* as an annotated automaton C^ϕ. Then, the product of $OG_X(N)$ and C^ϕ characterizes exactly those X-strategies of N that satisfy C^ϕ—for example, all X-strategies of N that will never pay by credit card.

In the remainder of this chapter, we introduce the notion of a product of X-operating guidelines in Section 6.1. Afterwards, in Section 6.2, we define several notions of substitutability under X-preservation and present for each notion a decision procedure. The results of this chapter are an extension of [SMB09].

6.1. The product of finite strategy representations

In this section, we introduce a *product operator* $\otimes$ of X-operating guidelines. Given two open nets N and N', the product of their corresponding X-operating guidelines $OG_X(N)$ and $OG_X(N')$ represents exactly the *intersection*

$\mathit{Strat}_X(N) \cap \mathit{Strat}_X(N')$ of the X-strategies of N and N'. As the product is commutative and associative, we can generalize this result to a finite set $\mathcal{S}$ of open nets. In this case, the intersection $\bigcap_{S\in\mathcal{S}} \mathit{Strat}_X(S)$ of sets of X-strategies can be represented by the product of the X-operating guidelines $OG_X(S)$ of all open nets $S \in \mathcal{S}$.

In the two subsequent subsections, we define the product of X-operating guidelines for open-net properties $X \in \{X_1, X_2(Y), X_3, X_4(Y)\}$ and prove that it characterizes the intersection of the open nets represented by these X-operating guidelines. For X_2 and X_4, the notion of a product is future work.

6.1.1. $X_2(Y)$-operating guidelines

In this section, we define the product of extended annotated automata. As extended annotated automata are more general than $X_2(Y)$-operating guidelines, the results of this section can be applied to $X_2(Y)$-operating guidelines without any restriction. The results of this section can also be directly applied to annotated automata, because we can transform every annotated automaton into an extended annotated automaton with equivalent behavior by adding a global constraint $\chi \equiv \mathit{true}$.

For annotated automata the results of this section have been first presented by Bretschneider in [Bre07] and have been published in [SMB09]. We generalize these results to extended annotated automata in the sequel.

A state of the product of two extended annotated automata A_1 and A_2 is a pair consisting of a state of each extended annotated automaton. The transition relation of the product is derived by those transitions that can be executed in both extended annotated automata. The local annotation of a state of the product is the conjunction of the annotations of the corresponding states of the extended annotated automata. Analogously, the product of the global constraints is the conjunction of these global constraints. This causes, however, some technicality. The reason is that each global constraint χ_1 and χ_2 ranges over some alphabet Q_1 and Q_2, respectively. The alphabet of the product of A_1 and A_2 is, however, the product $Q_1 \times Q_2$. The idea is to replace each literal q_1 of χ_1 by the disjunction $\bigvee_{q_2\in Q_2}(q_1, q_2)$. That way we achieve that if upon matching *true* is assigned to a state (q_1, q_2) with q_1 is a literal of χ_1, then the corresponding disjunction will be evaluated to *true*. Likewise, each literal q_2 of χ_2 is replaced by the disjunction $\bigvee_{q_1\in Q_1}(q_1, q_2)$.

Definition 6.1.1 (product of extended annotated automata).
The *product* $A_1^{\phi_1,\chi_1} \otimes A_2^{\phi_2,\chi_2} = ((Q, MC^I, MC^O, \delta, q_0, \phi), \chi)$ *of any two extended annotated automata* $A_1^{\phi_1,\chi_1} = ((Q_1, MC_1^I, MC_1^O, \delta_1, q_{01}, \phi_1), \chi_1)$ and $A_2^{\phi_2,\chi_2} = ((Q_2, MC_2^I, MC_2^O, \delta_2, q_{02}, \phi_2), \chi_2)$ with the same alphabets (i. e., $MC_1^I = MC_2^I$ and $MC_1^O = MC_2^O$) is defined by

- $Q = Q_1 \times Q_2$;

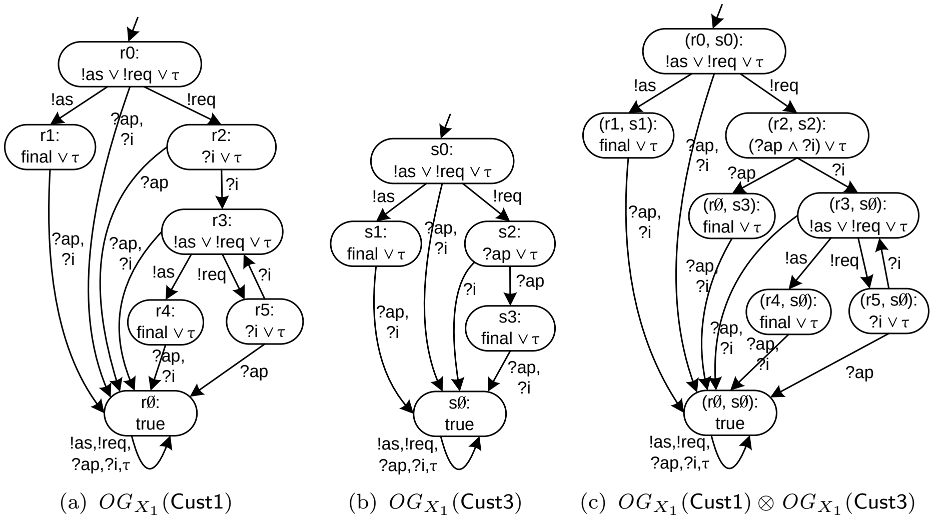


Figure 6.1.: Product of OG_{X_1}(Cust1) and OG_{X_1}(Cust3) of the customers Cust1 and Cust3 (cf. Figure 2.4). For purposes of simplification, we show the smaller X_1-operating guidelines in case of responsiveness and omit τ-labeled selfloops.

- $MC^I = MC_1^I \wedge MC^O = MC_1^O$;
- $((q_1, q_2), x, (q_1', q_2')) \in \delta$ iff $(q_1, x, q_1') \in \delta_1 \wedge (q_2, x, q_2') \in \delta_2$;
- $q_0 = (q_{0\,1}, q_{0\,2})$;
- $\phi((q_1, q_2)) = \phi_1(q_1) \wedge \phi_2(q_2)$, for all $(q_1, q_2) \in Q$; and
- $\chi \equiv \hat{\chi}_1 \wedge \hat{\chi}_2$, where $\hat{\chi}_1$ results from replacing any occurrence of q_1 in χ_1 by $\bigvee_{q_2 \in Q_2}(q_1, q_2)$, for all $q_1 \in Q_1$, and $\hat{\chi}_2$ results from replacing any occurrence of q_2 in χ_2 by $\bigvee_{q_1 \in Q_1}(q_1, q_2)$, for all $q_1 \in Q_1$. ⌟

In a way, the product of extended annotated automata is defined analogously to the synchronous product of finite automata [HU79]. Clearly, the product of two extended annotated automata is again an extended annotated automaton.

As an example, consider the two X_1-operating guidelines OG_{X_1}(Cust1) and OG_{X_1}(Cust3) in Figures 6.1(a) and 6.1(b), respectively. To calculate the product $OG_{X_1}(\otimes) = OG_{X_1}$(Cust1) $\otimes$ OG_{X_1}(Cust3), we relate the initial states of OG_{X_1}(Cust1) and OG_{X_1}(Cust3) yielding state (r0, s0). The annotation of state (r0, s0) is defined by ϕ_{Cust1}(r0) $\wedge$ ϕ_{Cust3}(s0), which is equal to !as $\vee$!req $\vee$ τ. As r0 and s0 enable transitions !as, !req and τ, state (r0, s0) has outgoing transitions !as, !req, and τ. The resulting product is shown in Figure 6.1(c).

The empty states of OG_{X_1}(Cust1) and of OG_{X_1}(Cust3) play an important role in the product. Consider state (r2, s2), for instance. Its annotation $(?\mathsf{ap} \wedge ?\mathsf{i}) \vee \tau$ requires that an X_1-strategy must be able to receive messages ap and i (or do an internal transition). However, message i can only be sent by Cust1, and hence state (r3, s$\emptyset$) contains the empty state s$\emptyset$ of $OG_{X_1}(\mathsf{Cust3})$. Analogously, message ap can only be sent by Cust3, and thus (r$\emptyset$, s3) contains the empty state r$\emptyset$ of $OG_{X_1}(\mathsf{Cust1})$.

The next theorem justifies that the product of two extended annotated automata characterizes the intersection of the sets of open nets represented by these automata.

Theorem 6.1.2 (product characterizes intersection).
For the product $A^{\phi,\chi} = A_1^{\phi_1,\chi_1} \otimes A_2^{\phi_2,\chi_2}$ of any two extended annotated automata $A_1^{\phi_1,\chi_1}$ and $A_2^{\phi_2,\chi_2}$ holds:

$$Match(A^{\phi,\chi}) = Match(A_1^{\phi_1,\chi_1}) \cap Match(A_2^{\phi_2,\chi_2}) \quad . \qquad \lrcorner$$

Proof.
Let $A_1^{\phi_1,\chi_1} = ((Q_1, MC^I, MC^O, \delta_1, q_{01}, \phi_1), \chi_1)$, $A_2^{\phi_2,\chi_2} = ((Q_2, MC^I, MC^O, \delta_2, q_{02}, \phi_2), \chi_2)$, and $A^{\phi,\chi} = A_1^{\phi_1,\chi_1} \otimes A_2^{\phi_2,\chi_2} = ((Q, MC^I, MC^O, \delta, (q_{01}, q_{02}), \phi), \chi)$.

($\Rightarrow$): Let open net $S \in Match(A^{\phi,\chi})$, and let $\varrho_\otimes$ be a minimal simulation relation of $SA(S)$ by $A^{\phi,\chi}$. We show that $S \in Match(A_1^{\phi_1,\chi_1})$ and $S \in Match(A_2^{\phi_2,\chi_2})$.

Let $(q_S, (q_1, q_2)) \in \varrho_\otimes$ be arbitrary. As $\varrho_\otimes$ is a minimal simulation relation, there is a sequence σ such that q_S is reached from q_{0S} via σ in $SA(S)$, and (q_1, q_2) is reached from (q_{01}, q_{02}) via σ in $A^{\phi,\chi}$. By construction of $A^{\phi,\chi}$, q_{01} and q_{02} are reached via σ in $A_1^{\phi_1,\chi_1}$ and $A_2^{\phi_2,\chi_2}$, too. By Definition 4.2.3 (matching), we have $(q_S, q_1) \in \varrho_1$ and $(q_S, q_2) \in \varrho_2$. Let there be an x-transition leaving state q_S. From $S \in Match(A^{\phi,\chi})$ and from Definition 4.2.3, we conclude that there is an x-transition leaving (q_1, q_2), too. By the construction of δ in Definition 6.1.1, there is an x-transition leaving state q_1 and one leaving state q_2. Hence, ϱ_1 and ϱ_2 are minimal simulation relations, too.

Furthermore, we conclude from $S \in Match(A^{\phi,\chi})$ and from Definition 4.2.3 that the assignment $\beta_{SA(S)}(q_S)$ satisfies $\phi((q_1, q_2))$. Hence, by the construction of ϕ in Definition 6.1.1, $\beta_{SA(S)}(q_S)$ also satisfies $\phi_1(q_1)$ and $\phi_2(q_2)$.

By $S \in Match(A^{\phi,\chi})$, we conclude that $SA(S)$ satisfies χ. As χ is according to Definition 6.1.1 the conjunction of $\hat{\chi}_1$ and $\hat{\chi}_2$, $SA(S)$ also satisfies $\hat{\chi}_1$ and $\hat{\chi}_2$ and hence also χ_1 and χ_2.

Consequently, S matches with $A_1^{\phi_1,\chi_1}$ and $A_2^{\phi_2,\chi_2}$, and therefore $S \in Match(A_1^{\phi_1,\chi_1}) \cap Match(A_2^{\phi_2,\chi_2})$.

($\Leftarrow$): Let $S \in Match(A_1^{\phi_1,\chi_1})$ and $S \in Match(A_2^{\phi_2,\chi_2})$. We show that $S \in Match(A^{\phi,\chi})$.

By $S \in Match(A_1^{\phi_1,\chi_1})$ and by Definition 4.2.3 (matching), there is a minimal simulation relation ϱ_1 of $SA(S)$ by $A_1^{\phi_1,\chi_1}$. Let $(q_S, q_1) \in \varrho_1$ be arbitrary. As ϱ_1 is a minimal simulation relation, there is a sequence σ such that q_S is reached from

q_{0S} via σ in $SA(S)$, and q_1 is reached via σ in $A_1^{\phi_1,\chi_1}$. By $S \in \mathit{Match}(A_2^{\phi_2,\chi_2})$ and Definition 4.2.3, there is a minimal simulation relation ϱ_2 and a state q_2 such that $(q_S, q_2) \in \varrho_2$ and q_2 is reached via σ in $A_2^{\phi_2,\chi_2}$. By the construction of δ in Definition 6.1.1, state (q_1, q_2) is reachable in $A^{\phi,\chi}$ via σ, too. Let there be an x-transition leaving state q_S. From $S \in \mathit{Match}(A_1^{\phi_1,\chi_1})$, from $S \in \mathit{Match}(A_2^{\phi_2,\chi_2})$, and from Definition 4.2.3 we conclude that there is an x-transition leaving q_1 and q_2. Hence, by construction of δ in Definition 6.1.1, transition x leaves state (q_1, q_2) of $A^{\phi,\chi}$, too. Consequently, there is a minimal simulation relation of $SA(S)$ by $A^{\phi,\chi}$.

From $S \in \mathit{Match}(A_1^{\phi_1,\chi_1})$, from $S \in \mathit{Match}(A_2^{\phi_2,\chi_2})$, and from Definition 4.2.3 we conclude that the assignment $\beta_{SA(S)}(q_S)$ satisfies $\phi_1(q_1)$ and $\phi_2(q_2)$. Hence, by the construction of ϕ in Definition 6.1.1, $\beta_{SA(S)}(q_S)$ also satisfies $\phi((q_1, q_2))$.

From $S \in \mathit{Match}(A_1^{\phi_1,\chi_1})$, from $S \in \mathit{Match}(A_2^{\phi_2,\chi_2})$, and from Definition 4.2.3 we conclude that $SA(S)$ satisfies χ_1 and χ_2. By the construction of $\hat{\chi}_1$ and $\hat{\chi}_2$ (see Definition 6.1.1) we conclude that $SA(S)$ also satisfies $\hat{\chi}_1$ and $\hat{\chi}_2$ and hence χ, the conjunction of $\hat{\chi}_1$ and $\hat{\chi}_2$.

Consequently, S matches with $A^{\phi,\chi}$, and therefore $S \in \mathit{Match}(A^{\phi,\chi})$. □

The product $\otimes$ of extended annotated automata is commutative and associative; that is, for extended annotated automata $A_1^{\phi_1,\chi_1}, A_2^{\phi_2,\chi_2}, A_3^{\phi_3,\chi_3}$ holds $A_1^{\phi_1,\chi_1} \otimes A_2^{\phi_2,\chi_2} = A_2^{\phi_2,\chi_2} \otimes A_1^{\phi_1,\chi_1}$ and $(A_1^{\phi_1,\chi_1} \otimes A_2^{\phi_2,\chi_2}) \otimes A_3^{\phi_3,\chi_3} = A_1^{\phi_1,\chi_1} \otimes (A_2^{\phi_2,\chi_2} \otimes A_3^{\phi_3,\chi_3})$.

Lemma 6.1.3 (product operator is commutative and associative).
The product operator, $\otimes$, of extended annotated automata is commutative and associative. ⌟

Lemma 6.1.3 enables us to apply the product operator to an arbitrary but finite set of extended annotated automata. By Theorem 6.1.2, we conclude that the product of n extended annotated automata represents exactly the intersection of the respective n sets of matching open nets.

Corollary 6.1.4 (intersection of a set of extended annotated automata).
For any finite set $\{A_1^{\phi_1,\chi_1}, \ldots, A_n^{\phi_n,\chi_n}\}$ of extended annotated automata with the same alphabets holds:

$$\mathit{Match}(A_1^{\phi_1,\chi_1} \otimes \cdots \otimes A_n^{\phi_n,\chi_n}) = \mathit{Match}(A_1^{\phi_1,\chi_1}) \cap \cdots \cap \mathit{Match}(A_n^{\phi_n,\chi_n}) \quad .$$ ⌟

The product of any finite set of $X_2(Y)$-operating guidelines represents the intersection of the sets of open nets characterized by these $X_2(Y)$-operating guidelines. Hence, given any finite set $\mathcal{S}$ of open nets and a set Y_S of nodes to be covered in each $S \in \mathcal{S}$, we can represent all open nets that are an $X_2(Y_S)$-strategy of each S of $\mathcal{S}$. These open nets are, in fact, represented by the product of the $X_2(Y_S)$-operating guidelines $OG_{X_2(Y_S)}(S)$, for all $S \in \mathcal{S}$.

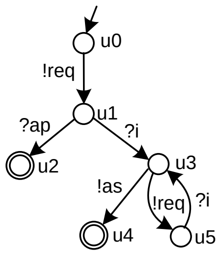


Figure 6.2.: Service automaton $SA(S)$ of an open net S that matches with $OG_{X_1}(\mathsf{Cust1}) \otimes OG_{X_1}(\mathsf{Cust3})$.

Corollary 6.1.5 (product represents intersection of a set of strategies). Let $\mathcal{S}$ be a set of interface equivalent open nets, let Y_S be the set of open-net nodes that should be covered in $S \in \mathcal{S}$, and let $OG_{X_2(Y_S)}(S)$ be the $X_2(Y_S)$-operating guideline of S. For the product $OG_\otimes$ of all $OG_{X_2(Y_S)}(S)$ holds

$$Match(OG_\otimes) = \bigcap_{S \in \mathcal{S}} Strat_{X_2(Y_S)}(S) \quad .$$

┘

If no places and transitions of any $S \in \mathcal{S}$ have to be covered, then the problem reduces to constructing the product of a set of X_1-operating guidelines for $S \in \mathcal{S}$.

In [KW09], the results presented in this section have been extended to extended annotated automata with annotations that may contain negated literals.

According to Corollary 6.1.5, the product in Figure 6.1(c) represents all open nets that are X_1-strategies of Cust1 *and* Cust3. For instance, the corresponding open net S of the service automaton $SA(S)$ in Figure 6.2 matches with the product, and it is an X_1-strategy of Cust1 and Cust3.

Implementation in Fiona

The calculation of the product of any two annotated automata has been implemented in the service analysis tool Fiona. Time and space for calculating the product is proportional to the product of the two annotated automata. Based on this implementation and the example services we used in the experiments in Chapter 4 and in Section 5.1.2, we present some experimental results.

As in case of X_1-conformance (see Section 5.1.2), we computed for each open net N its X_1-operating guideline $OG^R_{X_1}(N)$ (remember, Fiona implements responsiveness). To get a meaningful X_1-operating guideline for the product with $OG^R_{X_1}(N)$, we computed the minimal X_1-operating guideline $(OG^R_{X_1}(N))^{min}$ of N. The minimal X_1-operating guideline of N [Mas09] is the smallest annotated automaton that represents the X_1-strategies of N. Hence, by construction, $OG^R_{X_1}(N)$ and $(OG^R_{X_1}(N))^{min}$ are equivalent. Therefore, the product has the structure of the bigger annotated automaton $OG^R_{X_1}(N)$.

Table 6.1.: Constructing the product of X_1-operating guidelines with Fiona. All experiments were obtained on an UltraSPARC III processor with 900MHz and 4 GB RAM running Solaris 10.

	$MP^R_{X_1}(N)$		$(MP^R_{X_1}(N))^{min}$		time
Service	$\|Q\|$	$\|\delta\|$	$\|Q\|$	$\|\delta\|$	m:ss
Loan Approval	8	30	8	30	0:00
Purchase Order	169	1,182	169	1,182	0:00
Olive Oil Ordering	17	67	13	51	0:00
Travel Service 1	57	300	57	300	0:00
Travel Service 2	289	2,252	289	2,252	0:01
Online Shop 1	13	50	10	41	0:00
Online Shop 2	8	35	7	32	0:00
Beverage Machine	12	54	12	54	0:00
Philosophers #3	68	243	27	108	0:00
Philosophers #5	1,433	8,390	243	1,620	0:07
SMPT Protocol	1,217	8,408	381	2,822	0:08
Registration	8	30	7	27	0:00
Process 1	897	13,548	641	10,012	0:09
Process 2	1,608	16,445	1,608	16,445	0:19
Process 3	237	1,437	173	1,133	0:01
Process 4	6,821	103,165	6,821	103,165	6:16

Table 6.1 provides information about the size of the X_1-operating guideline of N (number of states and transitions), the size of the minimal X_1-operating guideline of N, and the time for computing the product of the two X_1-operating guidelines. The experiment illustrates that calculating the product is rather efficient. Only ‘Process 4’ took more than 6 minutes; all other examples took less than 20 seconds.

6.1.2. $X_4(Y)$-operating guidelines

In this section, we define the product of $X_4(Y)$-operating guidelines. With $X_4(Y)$ we denote the set {weak termination, cover(Y)} of open-net properties. As we can transform every X_3-operating guideline into an $X_4(Y)$-operating guideline with equivalent behavior by adding a global constraint $\chi \equiv \mathit{true}$ (i. e., $Y = \emptyset$), the results of this section can be directly applied to X_3-operating guidelines.

The idea for constructing the product $OG_\otimes$ of $OG_{X_4(Y)}(N)$ and $OG_{X_4(Z)}(N')$ is to construct the synchronous product $MP_\otimes$ of the most permissive X_3-strategies $MP_{X_3}(N)$ and $MP_{X_3}(N')$ and the product of the global constraints. Each state of $MP_\otimes$ forms a fragment of $OG_\otimes$, and a transition of $MP_\otimes$ is a connection of $OG_\otimes$. The states of a fragment of $MP_\otimes$ are the Cartesian product of the states of the corresponding fragments in $OG_{X_4(Y)}(N)$ and $OG_{X_4(Z)}(N')$.

In the product $MP_\otimes$, the empty state of $MP_{X_3}(N)$ and a nonempty state of $MP_{X_3}(N')$ can be related. The current definition of fragments (cf. Definition 4.3.8) defines for the empty state of $MP_{X_3}(N)$ a fragment, which is not connected to any other fragment, and this fragment does not have states. The reason is that the empty state cannot be reached in the composition of N and any open net. However, for a reasonable definition of a product $OG_{\otimes}$ of any two X_4-operating guidelines, the fragment that corresponds to the empty state must be reachable in an X_4-operating guideline. Therefore, we introduce a "dummy" construct, the *empty fragment* $F_{\emptyset}$. The empty fragment corresponds to the empty state of the most permissive X_3-strategy. It has only a single state $v_{\emptyset}$. This state is by definition a final state, as otherwise from a state $(v_{\emptyset}, v')$ in the product $OG_{\otimes}$ no final state is reachable. We add the empty fragment to the set of fragments of an open net N. Furthermore, for each transition in $MP_{X_3}(N)$ that has the empty state as its target, we add a connection.

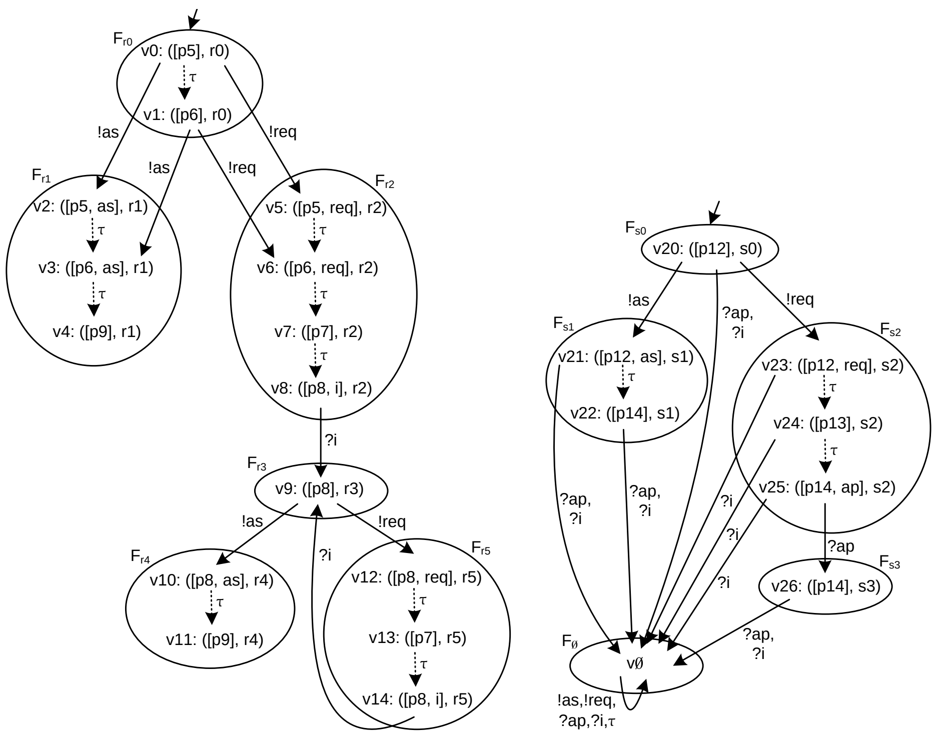
Definition 6.1.6 (empty fragment).
For any open net N, let $MP_{X_3}(N) = (Q, MC^I, MC^O, \delta, q_0, \Omega)$ be the most permissive X_3-strategy of N, and let $OG_{X_4(Y)}(N) = (\mathcal{F}(N), \mathcal{C}(N), \phi, \chi)$. For the empty state $q_{\emptyset}$ of $MP_{X_3}(N)$, define the *empty fragment* by $F_{\emptyset} = (\{v_{\emptyset}\}, MC^I \cup MC^O \cup \{\tau\}, \emptyset, \{v_{\emptyset}\})$. The *extension of $OG_{X_4(Y)}(N)$ by the empty fragment* is defined as $OG'_{X_4(Y)}(N) = (\mathcal{F}'(N), \mathcal{C}'(N), \phi, \chi)$ with

- $\mathcal{F}'(N) = \mathcal{F}(N) \cup \{F_{\emptyset}\}$;
- $$\begin{aligned}\mathcal{C}'(N) = \mathcal{C}(N) &\cup \{C_{(F_{\emptyset},x,F_{\emptyset})} \mid x \in MC^I \cup MC^O \cup \{\tau\}\} \\ &\cup \{C_{(F_q,x,F_{\emptyset})} \mid (q,x,q_{\emptyset}) \in \delta\}, \text{with} \\ &\qquad C_{(F_q,x,F_{\emptyset})} = (V_{F_q} \times \{x\} \times \{v_{\emptyset}\}).\end{aligned}$$

┘

As an example, the X_3-operating guidelines of Cust1 and Cust3 are illustrated in Figure 6.3. For the smaller representation, OG_{X_3}(Cust3), the empty fragment F$_\emptyset$ and the corresponding connections are depicted. For example, the most permissive X_3-strategy of Cust3—that is, the underlying automaton in Figure 6.1(b)—has two transitions ?ap and ?i in state s1 that have the empty state s∅ as its target. Hence, we add in Figure 6.3(b) two connections ?ap and ?i from every state of fragment F$_{\text{s1}}$ to state v∅ .

We can now define the product of two $X_4(Y)$-operating guidelines. Fragments and connections are defined similar to the synchronous product of ordinary automata. As a fragment is an LTS, the product of two fragments is defined as the Cartesian product of two LTSs. Analog to $X_2(Y)$-operating guidelines, the annotation of a fragment in the product is defined as the conjunction of the annotations of the corresponding fragments of the $X_4(Y)$-operating guidelines. Furthermore, the product of the global constraints is the conjunction of these global constraints. To deal with the empty state, we consider $X_4(Y)$-operating guidelines that are extended by the empty fragment.

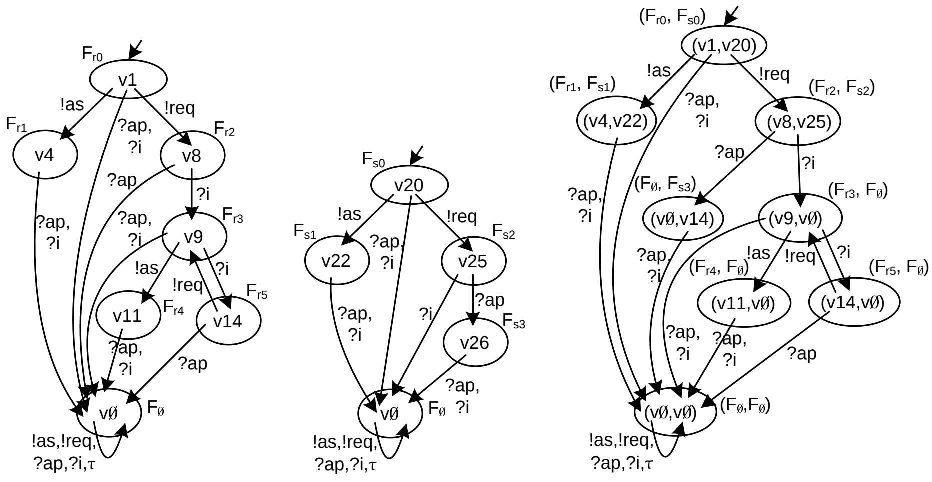


(a) $OG_{X_3}(\mathsf{Cust1})$ (b) $OG_{X_3}(\mathsf{Cust3})$

Figure 6.3.: X_3-operating guidelines of the customers $\mathsf{Cust1}$ and $\mathsf{Cust3}$. $OG_{X_3}(\mathsf{Cust3})$ is extended by the empty fragment $\mathsf{F}_\emptyset$.

Definition 6.1.7 (product of $X_4(Y)$-operating guidelines).
Let N and N' be any two interface equivalent open nets with $X_4(Y)$-operating guidelines extended by the empty fragment $OG_{X_4(Y)}(N) = (\mathcal{F}(N), \mathcal{C}(N), \phi, \chi)$ and $OG_{X_4(Z)}(N') = (\mathcal{F}(N'), \mathcal{C}(N'), \phi', \chi')$. Let Q and Q' be the states of the most permissive X_3-strategy of N and N', respectively. The *product* $OG_{X_4(Y)}(N) \otimes OG_{X_4(Z)}(N') = (\mathcal{F}_\otimes, \mathcal{C}_\otimes, \phi_\otimes, \chi_\otimes)$ is defined by

- $\mathcal{F}_\otimes = \mathcal{F}(N) \times \mathcal{F}(N')$ and the product of any two fragments $F \in \mathcal{F}(N)$ and $F' \in \mathcal{F}(N')$ is defined as the fragment $(F, F') = (V_{(F,F')}, \Sigma, E_{(F,F')}, \Omega_{(F,F')})$ with
  - $V_{(F,F')} = V_F \times V_{F'}$;
  - $$\begin{aligned} E_{(F,F')} = \quad & \{((v_F, v_{F'}), \tau, (v'_F, v_{F'})) \mid (v_F, v_{F'}) \in V_{(F,F')} \\ & \qquad \wedge (v_F, \tau, v'_F) \in E_F\} \\ \cup\, & \{((v_F, v_{F'}), \tau, (v_F, v'_{F'})) \mid (v_F, v_{F'}) \in V_{(F,F')} \\ & \qquad \wedge (v_{F'}, \tau, v'_{F'}) \in E_{F'}\}; \end{aligned}$$



(a) OG_{X_3}(Cust1) (b) OG_{X_3}(Cust3) (c) OG_{X_3}(Cust1) $\otimes$ OG_{X_3}(Cust3)

Figure 6.4.: Reduced X_3-operating guidelines OG_{X_3}(Cust1) and OG_{X_3}(Cust3) and their product. τ-loops are omitted.

– $\Omega_{(F,F')} = \Omega_F \times \Omega_{F'}$.

- $C_{((q_1,q_1'),x,(q_2,q_2'))} \in \mathcal{C}_\otimes$ iff $C_{(q_1,x,q_2)} \in \mathcal{C}(N) \wedge C_{(q_1',x,q_2')} \in \mathcal{C}(N')$;
- $((v_1,v_1'),x,(v_2,v_2')) \in C_{((q_1,q_1'),x,(q_2,q_2'))}$ iff $(v_1,x,v_2) \in C_{(q_1,x,q_2)} \wedge (v_1',x,v_2') \in C_{(q_1',x,q_2')}$;
- $\phi_\otimes((F,F')) = \phi(F) \wedge \phi'(F')$, for all $(F,F') \in \mathcal{F}_\otimes$; and
- $\chi_\otimes \equiv \hat{\chi} \wedge \hat{\chi}'$, where $\hat{\chi}$ results from replacing any occurrence of q in χ by $\bigvee_{q' \in Q'}(q,q')$, for all $q \in Q$, and $\hat{\chi}'$ results from replacing any occurrence of q' in χ' by $\bigvee_{q \in Q}(q,q')$, for all $q \in Q$.

Let the initial state of the product be defined by the product of the respective initial states of $OG_{X_4(Y)}(N)$ and $OG_{X_4(Z)}(N')$. ⌟

The set $\mathcal{F}_\otimes$ of fragments may contain a fragment that is the product of the empty fragments of $OG_{X_4(Y)}(N)$ and of $OG_{X_4(Z)}(N')$. This ensures that the product is again an $X_4(Y)$-operating guideline extended by the empty fragment.

As an example, the product of the reduced X_3-operating guidelines OG_{X_3}(Cust1) and OG_{X_3}(Cust3) is illustrated in Figure 6.4. The initial state is derived by the fragments $\mathsf{F_{r0}}$ and $\mathsf{F_{s0}}$ and the product of their initial states v1 and v20.

The product of $X_4(Y)$-operating guidelines with an extended annotated automaton is a special case of Definition 6.1.7. Each state of the extended annotated automaton corresponds to a fragment (without any internal states), and each transition corresponds to a connection.

The product of any two $X_4(Y)$-operating guidelines characterizes the intersection of the sets of open nets represented by these $X_4(Y)$-operating guidelines.

Theorem 6.1.8 (product characterizes intersection).
For the product $OG_\otimes = OG_{X_4(Y)}(N) \otimes OG_{X_4(Z)}(N')$ of any two $X_4(Y)$-operating guidelines of N and N' extended by the empty fragment holds

$$\mathit{Match}(OG_\otimes) = \mathit{Match}(OG_{X_4(Y)}(N)) \cap \mathit{Match}(OG_{X_4(Z)}(N')) \quad . \qquad \lrcorner$$

Proof.
Let $OG_{X_4(Y)}(N) = (\mathcal{F}(N), \mathcal{C}(N), \phi, \chi)$, $OG_{X_4(Z)}(N') = (\mathcal{F}(N'), \mathcal{C}(N'), \phi', \chi')$, and $OG_\otimes = OG_{X_4(Y)}(N) \otimes OG_{X_4(Z)}(N') = (\mathcal{F}_\otimes, \mathcal{C}_\otimes, \phi_\otimes, \chi_\otimes)$. Let $TS_\otimes$, TS, and TS' denote the LTS constructed from a service automaton $SA(S)$ and fragments of $OG_\otimes$, $OG_{X_4(Y)}(N)$, and $OG_{X_4(Z)}(N')$, respectively. Let $MP_{X_3}(N)$ and $MP_{X_3}(N')$ be the respective most permissive X_3-strategies of N and N', and let $MP_\otimes = MP_{X_3}(N) \otimes MP_{X_3}(N')$.

($\Rightarrow$): Let $S \in \mathit{Match}(OG_\otimes)$. It remains to show that $S \in \mathit{Match}(OG_{X_4(Y)}(N))$ and $S \in \mathit{Match}(OG_{X_4(Z)}(N'))$. According to Definition 4.4.2, we have to show that

(i) $MP_{X_3}(N)$ and $MP_{X_3}(N')$ simulate $SA(S)$;

(ii) $SA(S)$ satisfies the annotations ϕ and ϕ';

(iii) TS and TS' weakly terminate; and

(iv) $SA(S)$ satisfies the global constraints χ and χ'.

Conditions (i), (ii), and (iv) follow the same argumentation as in the proof of Theorem 6.1.2, because the product is constructed similarly. So it remains to show (iii).

Let $((m_N, q_{MP}), (m_{N'}, q_{MP'}), q_S)$ be an arbitrary state of $TS_\otimes$. By assumption, $TS_\otimes$ weakly terminates; that is, a final state $((m'_N, q'_{MP}), (m'_{N'}, q'_{MP'}), q'_S)$ is reachable from state $((m_N, q_{MP}), (m_{N'}, q_{MP'}), q_S)$. State $((m_N, q_{MP}), q_S)$ is by (i) and (ii) reachable in TS. We show that a final state in TS can be reached from $((m_N, q_{MP}), q_S)$.

Case 1: Suppose $(m_N, q_{MP}) \neq v_\emptyset$ and $(m'_N, q'_{MP}) \neq v_\emptyset$; that is, both states are not in the respective empty fragments. Then, state $((m'_N, q'_{MP}), q'_S)$ can be reached in TS, because (i) ensures that (q'_S, q'_{MP}) are related by the minimal simulation relation of $SA(S)$ by $MP_{X_3}(N)$, and hence the fragment of q'_{MP} is reachable in TS. Furthermore, the Cartesian product of states in a fragment of $OG_\otimes$ ensures reachability of m'_N in the fragment of q'_{MP}. As $((m'_N, q'_{MP}), (m'_{N'}, q'_{MP'}), q'_S)$ is a final state and by the definition of final states in the product (cf. Definition 6.1.7), $((m'_N, q'_{MP}), q'_S)$ is a final state in TS.

Case 2: Now suppose $(m_N, q_{MP}) \neq v_\emptyset$ and $(m'_N, q'_{MP}) = v_\emptyset$; that is, (m_N, q_{MP}) is a state of TS and (m'_N, q'_{MP}) is the empty fragment). In this case, the state $((m'_N, q'_{MP}), q'_S)$ is not reachable in TS. Let $((m''_N, q''_{MP}), (m''_{N'}, q''_{MP'}), q''_S)$ be the last state on a path from state $((m_N, q_{MP}), (m_{N'}, q_{MP'}), q_S)$ to state $((m'_N, q'_{MP}), (m'_{N'}, q'_{MP'}), q'_S)$ with $(m''_N, q''_{MP}) \neq v_\emptyset$. From (ii) we conclude that in fragment q''_{MP} of $OG_{X_3}(N)$ either a final state is reachable or there is another transition in state $((m''_N, q''_{MP}), (m''_{N'}, q''_{MP'}), q''_S)$ such that the empty fragment of $OG_{X_4}(N)$ is not entered. As $TS_\otimes$ is by assumption weakly terminating, a final state is reachable via this transition. That way, a final state of $TS_\otimes$ can be reached such that $OG_{X_4}(N)$ is not in its empty fragment. By Case 1, this path is possible in TS, too.

Case 3: Suppose $(m_N, q_{MP}) = v_\emptyset$; that is, (m_N, q_{MP}) is the empty fragment. In this case, nothing has to be shown, because state $((m_N, q_{MP}), q_S)$ is not reachable in TS.

As the same argumentation also holds for the states in TS', we conclude that $S \in Match(OG_{X_4(Y)}(N))$ and $S \in Match(OG_{X_4(Z)}(N'))$.

($\Leftarrow$): Let $S \in Match(OG_{X_4(Y)}(N)) \cap Match(OG_{X_4(Z)}(N'))$. We show that $S \in Match(OG_\otimes)$. According to Definition 4.4.2, we have to show that

(i) $MP_\otimes$ simulates $SA(S)$;

(ii) $SA(S)$ satisfies $\phi_\otimes$ in each states of the minimal simulation relation;

(iii) $TS_\otimes$ weakly terminates; and

(iv) $SA(S)$ satisfies the global constraint $\chi_\otimes$.

Conditions (i), (i), and (iv) follow the same argumentation as in the proof of Theorem 6.1.2, because the product is constructed similarly. So it is sufficient to show condition (iii); that is, $TS_\otimes$ weakly terminates.

Let $((m_N, q_{MP}), (m_{N'}, q_{MP'}), q_S)$ be an arbitrary state of $TS_\otimes$. We show that a final state can be reached from this state. By the definition of $OG_\otimes$, $((m_N, q_{MP}), q_S)$ is a state of TS, and as TS is by assumption weakly terminating, a final state $((m'_N, q'_{MP}), q'_S)$ is reachable from $((m_N, q_{MP}), q_S)$ with $q_S \xrightarrow{\sigma} q'_S$. The respective sequence σ is also possible in $TS_\otimes$ (follows from condition (i)) yielding a state $((m'_N, q'_{MP}), (m'_{N'}, q'_{MP'}), q'_S)$. We distinguish two cases.

Case 1: Let $(m'_{N'}, q'_{MP'}) = v_\emptyset$; that is, $(m'_{N'}, q'_{MP'})$ is the empty fragment. Then state $((m'_N, q'_{MP}), (m'_{N'}, q'_{MP'}), q'_S)$ is by Definition 6.1.6 (empty fragment) a final state.

Case 2: Let $(m'_{N'}, q'_{MP'}) \neq v_\emptyset$; that is, $(m'_{N'}, q'_{MP'})$ is not the empty fragment. If $(m'_{N'}, q'_{MP'})$ is a final state, then state $((m'_N, q'_{MP}), (m'_{N'}, q'_{MP'}), q'_S)$ is a final state as well. Otherwise, we know that a final state is reachable from $(m'_{N'}, q'_{MP'})$ in TS'. The respective path is also possible in TS (follows from (i) and (ii)). Hence, a final state is reachable, where both $OG_{X_4(Y)}(N)$ and $OG_{X_4(Z)}(N')$ are in a final state. By the definition of a final state in Definition 6.1.7, this is a final state of the product.

Hence, we conclude that $TS_\otimes$ weakly terminates. □

Also the product $\otimes$ of $X_4(Y)$-operating guidelines is commutative and associative.

Lemma 6.1.9 (product is commutative and associative).
The product operator, $\otimes$, of $X_4(Y)$-operating guidelines is commutative and associative. ⌟

Lemma 6.1.9 enables us to apply the product operator to an arbitrary but finite set of $X_4(Y)$-operating guidelines. By Theorem 6.1.8, the product characterizes the intersection of the respective sets of open nets. Hence, given any finite set $\mathcal{S}$ of open nets, we can represent all open nets that are an $X_4(Y_S)$-strategy of each S of $\mathcal{S}$ where Y_S denotes the nodes of S to be covered. This set can be represented by the product of $X_4(Y_S)$-operating guidelines $OG_{X_4(Y_S)}(S)$ of all $S \in \mathcal{S}$.

Corollary 6.1.10 (product represents intersection of a set of strategies).
Let $\mathcal{S}$ be a set of interface equivalent open nets, let Y_S be the set of nodes that should be covered in S, and let $OG_{X_4(Y_S)}(S)$ be the $X_4(Y_S)$-operating guideline of S. For the product $OG_\otimes$ of all $OG_{X_4(Y_S)}(S)$ holds

$$Match(OG_\otimes) = \bigcap_{S \in \mathcal{S}} Strat_{X_4(Y_S)}(S) \quad .$$

⌟

If quasi-liveness is not required, then the problem reduces to generating the product of a set of X_3-operating guidelines for $S \in \mathcal{S}$.

The product of $OG_{X_3}(\mathsf{Cust1})$ and $OG_{X_3}(\mathsf{Cust3})$ in Figure 6.4(c) represents by Corollary 6.1.10 all open nets that match with $OG_{X_3}(\mathsf{Cust1})$ *and* $OG_{X_3}(\mathsf{Cust3})$. As an example, the corresponding open net S of $SA(S)$ in Figure 6.2 matches with the product of the X_3-operating guidelines in Figure 6.4(c): Remove fragment $(\mathsf{F_{r1}}, \mathsf{F_{s1}})$ from Figure 6.4(c). The resulting state space is the composition of S and the product of Cust1 and Cust3. It is easy to see that the weak termination property holds for this state space.

6.2. Preservation check with the product

In this section, we present a procedure to *decide* substitutability under X-preservation. To this end, we use the notion of a product of X-operating guidelines as defined in the previous section.

Given two interface equivalent open nets N and N' and a set $\mathcal{S} \in Strat_X(N)$ of X-strategies of N, we want to decide whether N' substitutes N under preservation of $\mathcal{S}$. Clearly, the decision procedure for substitutability under X-preservation depends on the set $\mathcal{S}$. In this section, we consider the following three representations of $\mathcal{S}$:

- The set $\mathcal{S}$ of open nets is a *finite*. For instance, N' shall restrict N to its core functionalities, or it shall only support N's major clients.
- The set $\mathcal{S}$ characterizes *infinitely* many open nets, and it is represented as an X-operating guideline. For example, this representation might result from the product $OG_X(S_1) \otimes \cdots \otimes OG_X(S_n)$ of a finite set of X-operating guidelines—that is, $\mathcal{S} = \mathit{Match}(OG_X(S_1) \otimes \cdots \otimes OG_X(S_n))$. A possible application would be that only clients that are compatible to all S_i shall be preserved by N'.
- The set $\mathcal{S}$ characterizes all those X-strategies of N that follow a *behavioral constraint*. Such a behavioral constraint can be described as an annotated automaton C^{ψ}. Formally, the set $\mathcal{S}$ can be characterized by the product of the X-operating guideline of N and C^{ψ}—that is, $\mathcal{S} = \mathit{Match}(OG_X(N) \otimes C^{\psi})$. Applications include to exclude customers that pay by credit card.

Each of the three representations of $\mathcal{S}$ is addressed in a separate subsection, where we formalize the respective notion of substitutability under X-preservation and present an algorithm to decide this substitutability criterion.

6.2.1. Deciding X-preservation of a finite set of strategies

This section presents a decision procedure for substitutability of N by N' under X-preservation of a *finite* set $\mathcal{S}$ of open nets. We propose two solutions for deciding substitutability in this setting.

The first solution is obvious: Open net N' X-preserves $\mathcal{S}$ if and only if every open net $S \in \mathcal{S}$ matches with N'. As $\mathcal{S}$ is finite, substitutability can be decided.

The second solution to solve this problem requires the notion of a product of X-operating guidelines. The next theorem shows that substituting N by N' X-preserves $\mathcal{S}$ if and only if N' is an X-strategy characterized by the product of the X-operating guidelines $OG_X(S)$, for all $S \in \mathcal{S}$. As the proof of this theorem makes use of the fact that X-strategies are symmetric, we have to exclude cover(Y) from the set of open-net properties.

Theorem 6.2.1 (preservation check with product).
Let N and N' be interface equivalent open nets, let X be a set of open-net properties with cover(Y) $\notin X$, and let $\mathcal{S} \subseteq \mathit{Strat}_X(N)$. Let $OG_X(S)$ be an X-operating guideline of $S \in \mathcal{S}$, and let $OG_{\otimes}$ denote the product of all $OG_X(S)$. Open net N' X-preserves $\mathcal{S}$ iff $N' \in \mathit{Match}(OG_{\otimes})$. ⌟

Proof.
We show that $\mathit{Match}(OG_{\otimes})$ characterizes all open nets N' that can substitute N

while preserving $\mathcal{S}$. We have:

$$\begin{aligned}
\mathit{Match}(\mathit{OG}_{\otimes}) &= \bigcap_{S \in \mathcal{S}} \mathit{Strat}_X(S) \text{ (by Corollaries 6.1.5 and 6.1.10)}\\
&= \{N' \mid \text{for all } S \in \mathcal{S} : N' \in \mathit{Strat}_X(S)\}\\
&= \{N' \mid \text{for all } S \in \mathcal{S} : S \in \mathit{Strat}_X(N')\}\\
&\quad\ (X\text{-strategy is symmetric})\\
&= \{N' \mid N' \text{ preserves } \mathcal{S}\} \text{ (by Definition 3.2.1)}
\end{aligned}$$

Consequently, the theorem holds. □

Intuitively, the fewer X-strategies shall be preserved by the substitution (i. e., the smaller $\mathcal{S}$ is), the more open nets N' exist that may substitute N (i. e., the bigger is $\mathit{Match}(\mathit{OG}_{\otimes})$).

To decide whether substituting N by N' X-preserves $\mathcal{S} \subseteq \mathit{Strat}_X(N)$, we have to construct the X-operating guideline of each $S \in \mathcal{S}$. Then we calculate the product of these representations. Time and space complexity for calculating the product of two such representations is proportional to the product of their states. Therefore, this complexity effort only pays off if we check several N'. The restriction check based on Theorem 6.2.1 has been implemented for X_1-strategies (i. e., for X_1-operating guidelines) in the service analysis tool Fiona (see also the experimental results in Section 6.1.1).

As Theorem 6.2.1 requires a symmetric X-strategy notions and the product has not been defined for X_2- and X_4-operating guidelines, the decision procedure of Theorem 6.2.1 can only be applied for open-net properties X_1 and X_3.

As an example, Figure 6.5 shows the online bank Bank″. Suppose we are interested in substituting Bank by Bank″ under X_1-preservation of customers Cust1 and Cust3 in Figure 6.1. Then, we have to check whether Bank″ matches with the product $\mathit{OG}_{X_1}(\mathsf{Cust1}) \otimes \mathit{OG}_{X_1}(\mathsf{Cust3})$ in Figure 6.1(c). This does not hold, because after having sent message req, Bank″ can only receive message ap. In contrast, the product requires in state (r2,s2) that an online bank can also receive message i or accepts an internal transition. Consequently, Bank cannot be substituted by Bank″.

6.2.2. Deciding X-preservation of an infinite set of strategies

The application scenario, presented in the previous section, cannot always be applied; for example, if the set $\mathcal{S}$ contains too many open nets or is even infinite. In this case, we prefer a finite representation of all X-strategies of N, which have to be preserved by the substitution.

Lemma 6.2.2 (X-preservation of an infinite set of X-strategies).
Let N and N' be interface equivalent open nets, and let $\mathcal{S} \subseteq \mathit{Strat}_X(N)$ be specified by an X-operating guideline OG_X, i. e., $\mathcal{S} = \mathit{Match}(\mathit{OG}_X)$. Open net N' X-preserves $\mathcal{S}$ iff $\mathit{Match}(\mathit{OG}_X(N')) \supseteq \mathit{Match}(\mathit{OG}_X)$. ⌟

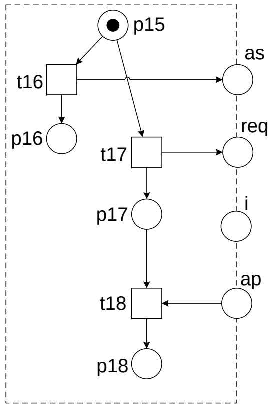


Figure 6.5.: An improved version Bank$''$ of the online bank Bank in Figure 3.3(a).

That way, deciding whether N' substitutes N under X-preservation of $\mathcal{S}$ reduces to check whether $OG_X(N')$ refines the X-operating guideline OG_X that represents $\mathcal{S}$. Based on our results on X-conformance in Chapter 5, we can decide X-preservation for X_1-strategies (cf. Theorem 5.1.2) and for $X_2(Y)$-strategies (cf. Theorem 5.2.12).

It is worthwhile to mention that OG_X may be the product of some X-operating guidelines. For example, one might be interested in all open nets that are compatible to an open net N_1 and to an open net N_2. In this case, we have $OG_X = OG_X(N_1) \otimes OG_X(N_2)$. Another application is the restriction of the set of X-strategies of N according to a behavioral constraint. This is addressed in the following.

6.2.3. Deciding constraint-conforming substitutability

In this section, we show another possibility to specify an X-operating guideline that represents a set $\mathcal{S} \subseteq \mathit{Strat}_X(N)$. The idea is to restrict $\mathcal{S}$ to those X-strategies of N that satisfy a *behavioral constraint*, such as "do not pay by credit card".

To this end, we have to describe all open nets that satisfy a behavioral constraint. In this section, we restrict ourselves to those behavioral constraints, where the corresponding set of open nets can be represented by a *constraint automaton* C^ψ [LMW07a]. A constraint automaton is an annotated automaton that constrains labels of the input and of the output alphabet of an X-operating guideline $OG_X(N)$ of N. Here, to constrain means to exclude some labels or to exclude a particular order of the labels.

Definition 6.2.3 (constraint automaton).
Let $OG_X(N)$ be an X-operating guideline of an open net N. An annotated

automaton C^ψ with the same alphabet as $SA(N)$ is a *constraint automaton* for $OG_X(N)$. ⌟

Intuitively, $OG_X(N)$ represents the set of X-strategies of N, and the constraint automaton C^ψ describes the behavior we want to allow or disallow in the restricted subset of X-strategies. Their product characterizes all X-strategies of N that conform to C^ψ. In Section 6.1, we showed that the product of an X-operating guideline and an annotated automaton is again an X-operating guideline.

Given a product $OG_X(N) \otimes C^\psi$, each open net N', where $OG_X(N')$ characterizes exactly these X-strategies, is a well-suited candidate for substituting N. This yields a more fine-grained notion of substitutability under preservation, which is covered by the following definition.

Definition 6.2.4 (constraint-conforming substitution).
Let N and N' be any interface equivalent open nets with X-operating guidelines $OG_X(N)$ and $OG_X(N')$. Let C^ψ be a constraint automaton for $OG_X(N)$. The substitution of N by N' *conforms to* C^ψ iff $\mathit{Match}(OG_X(N')) = \mathit{Match}(OG_X(N) \otimes C^\psi)$. ⌟

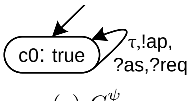
Note that the equation is necessary, as otherwise there might exist an X-strategy of N', which is not contained in $\mathit{Match}(OG_X(N) \otimes C^\psi)$ but violates the constraint automaton C^ψ. If we are only interested in preserving $\mathit{Match}(OG_X(N) \otimes C^\psi)$ and do not want to guarantee that every X-strategy of N' satisfies C^ψ, then symbol '$=$' in Definition 6.2.4 can be replaced by '$\supseteq$'.

For a *given* open net N', we can check whether or not it is a valid constraint-conforming substitution of N. To this end, we apply Lemma 6.2.2 and reduce the decision procedure to decide equivalence of X-operating guidelines. Remember that this thesis only presents decision algorithms for X_1-operating guidelines and X_2-operating guidelines.

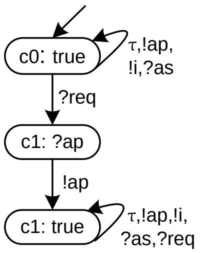
The notion of a constraint automaton and the substitutability check based on Definition 6.2.4 has been implemented for X_1-strategies in the service analysis tool Fiona.

To illustrate the notion of a constraint automaton, consider Figure 6.6. The constraint automaton C_1^ψ in Figure 6.6(a) represents all open nets that do not send any information message. As another example, the constraint automaton C_2^ψ in Figure 6.6(b) represents all open nets that if they send a suggestion for an appointment ap, then this happens immediately after having received a request for an appointment req.

Let us now pick up the example of Section 3.2. There, we argued that the improved banking service Bank'' (see Figure 6.5) can substitute the banking service Bank (see Figure 2.3) under X_1-preservation of all X_1-strategies of Bank that do not send any information message i. The constraint automaton C_2^ψ in Figure 6.6(a) represents all open nets that do not send an information message. The product of $OG_{X_1}(\mathsf{Bank})$ (see Figure 4.5) and C_3^ψ represents all X_1-strategies of



(a) C_1^ψ



(b) C_2^ψ

Figure 6.6.: Examples for constraint-conforming substitutability.

Bank that have to be preserved by the substitution. This product results in removing the states q4, q7, q8, q9 and their adjacent transitions from OG_{X_1}(Bank). Using Fiona, we can check that the X_1-operating guideline of Bank$''$ refines the product—as OG_{X_1}(Bank$''$) has too many states, we do not show this example. Hence, we conclude that Bank$''$ can substitute Bank.

The notion of a constraint automaton is not very expressive. It can be used to specify simple constraints—that is, excluding single labels or simple message causalities; see also [Wol09]. Clearly, we can combine such simple constraints to more complex constraints. The semantics of such complex constraints is, however, the *conjunction* of its simple constraints. To specify more expressive constraints, we need besides the intersection of X-strategies (i. e., the product) also operators for *negation* and *union* of sets of X-strategies. For sets of $X_2(Y)$-strategies, these operators have been presented in [KW09]. Negation, union, and product define an algebra on $X_2(Y)$-operating guidelines; see [KW09]. With the help of this algebra, more expressive constraints can be constructed, and hence the applicability of constraint-preserving substitution can be extended.

Part IV.

Constructing Substitutable Services

7. Deriving Substitutable Services with Transformation Rules

In the setting of multiparty contracts, each party has to design a private view for its public view. The resulting private view has to X-conform to its corresponding public view. Using the results of the two previous chapters, we can decide X-conformance as well as preservation. However, the design of a private view is a nontrivial and error-prone task even for experienced service designers. The main reason is that supporting the design process of service implementations and checking whether the implementation can substitute its specification is not a particular strength of the BPM tools currently available on the marketplace. Hence, the current chapter is devoted to introduce methods to construct open nets N' that can substitute an open net N.

In this chapter, we introduce an approach to *refine* open nets; see also Figure 1.4. Given an open net N, we want to incrementally transform N into an open net N' such that every transformation step preserves X-conformance. To this end, patterns of N are incrementally replaced by other patterns. In this approach, a pattern M of N is replaced by another pattern M' yielding the open net N'. We prove that if M' X-conforms to M, then N' X-conforms to N.

We present several *transformation rules* in this chapter. Some of them are inherited from the literature—for example, the projection-inheritance preserving transformation rules of Basten and Van der Aalst [BA01, AB02] in Section 7.2. These rules are restricted to add or remove internal transitions. We further present some rules that also affect the interface transitions of an open net. Some of these rules preserve X-conformance equivalence (see Section 7.3), whereas other rules only preserve X-conformance in one direction (see Section 7.4). A transformation rule does not necessarily guarantee quasi-liveness. Hence, we refrain from preserving quasi-liveness, and we restrict ourselves throughout this chapter to open-net properties $X_3 = \{\text{weak termination}\}$. As strict termination does not change the finite representation of X_3-strategies (see Section 4.5) and a pattern will never contain a final marking, the results of this section can also be applied if $\{\text{weak termination, strict termination}\}$ is the chosen open-net property.

The transformation rules are sufficient, but they are *not complete*, meaning they do not cover all possible service implementations. However, we think that these rules are highly relevant in practise. Real-life service models are often well-structured rather than complicated, because many enterprises have strict modeling guidelines that only allow the use of a limited set of modeling constructs. In that sense, our transformation rules should meet the requirements in practice.

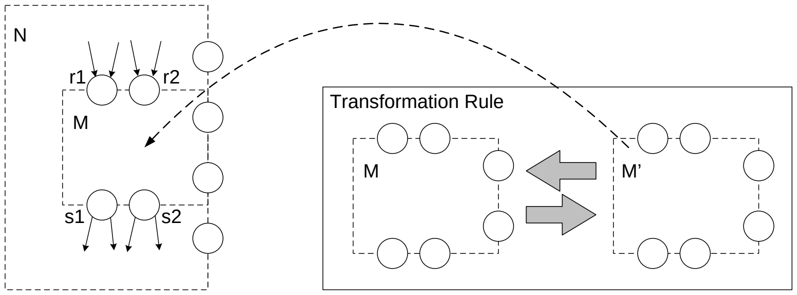


Figure 7.1.: Illustration of the transformation approach.

Before we present the different transformation rules in Sections 7.2–7.4, we introduce the transformation approach and provide the formalities in Section 7.1. The work of this chapter has been partly published in [ALM+08, AMSW09]

7.1. The transformation approach

In this section we provide the basic concepts of our transformation approach. We introduce the notion of a pattern of an open net N, and we define the composition of two patterns to specify more general patterns. Finally, we justify the correctness of the transformation approach.

The idea of the transformation approach is to identify a pattern M in an open net N and to substitute M by a pattern M' under X_3-conformance. To this end, a transformation rule specifies a pair (M, M') of patterns that fulfill X_3-conformance. Basically, we see a *pattern* as a *set* of open nets with some property. A member of a pattern—that is, a single open net—is a *pattern instance*. In the rest of this chapter, we will not distinguish between pattern and pattern instance and only use the term pattern. Figure 7.1 illustrates the transformation approach. We continue by formalizing patterns in terms of open nets.

An open net M is a *pattern* of an open net N if there is an open net N_{rest} and the composition of M and N_{rest} is the open net N. The set P^I of input places of M is divided into two sets: some internal places $R \subseteq P^I$ of N and some input places $P^I \setminus R$ of N. Likewise the set P^O of output places of N is divided into two sets: some internal places $S \subseteq P^O$ of N and some output places $P^O \setminus S$ of N. Places R are the input places and places S are the output places of M to N_{rest}. For technical reasons, we require that the initial marking of a pattern is the empty marking, and the set of final markings is the singleton set with the empty marking.

Definition 7.1.1 (pattern).
Let N, M be any two open nets with $M = (P, T, F, P^I, P^O, [\,], \{[\,]\})$, and let $R \subseteq P^I$ and $S \subseteq P^O$. Open net M is a *pattern of* N iff there exists an open net

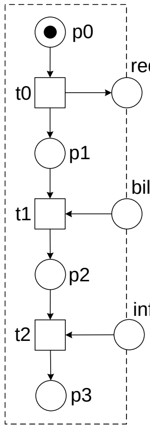


Figure 7.2.: Client of the contract in Figure 2.7.

$N_{rest} = (P_r, T_r, F_r, P_r^I, P_r^O, m_{0r}, \Omega_r)$ such that $N = M \oplus N_{rest}$ and $P_r^I \cap P^I = R$ and $P_r^O \cap P^O = S$. ⌟

In order that an open net M can serve as a pattern, its initial marking must be the empty marking, and its set of final markings must be the singleton set with the empty marking. This restriction is necessary, as otherwise the composition of M and N_{rest} may contain tokens on the shared interface places $R \cup S$ in the initial and final markings violating Definition 2.4.2 (open-net composition). The second requirement concerns with the fact that a pattern M is defined in the context of a particular open net N. To this end, we require the existence of an open net N_{rest} with $N = M \oplus N_{rest}$. The composition ensures that, for all transitions $t \in T_r$, we have ${}^\bullet t \cap R = \emptyset$ and $t^\bullet \cap S = \emptyset$. Furthermore, there are no other arcs from nodes of N_{rest} to nodes of M and vice versa than those to and from the shared interface places $R \cup S$.

As an example, Figure 7.2 depicts the open net Client. A possible pattern M would be the open net with $P_M = \{\text{p1, p2, bill}\}$, $T_M = \{\text{t1}\}$, and the adjacent arcs. In this case, $R_M = \{\text{p1}\}$ and $S_M = \{\text{p2}\}$.

The next corollary states that if an open net N has a pattern M and there is another pattern M' that X_3-conforms to M, then we can substitute M by M' without affecting any X_3-strategy of N. Such transformations can be applied incrementally and thus refine a service specification to an implementation by applying transformation steps. The resulting implementation is correct by construction; that is, it preserves all X_3-strategies of the specification.

Corollary 7.1.2 (justification of transformation rules).
Let $N_1 \oplus N_2$ be an X_3-open net, let M be a pattern of N_1, and let N_{rest} be an open net such that $N_1 = M \oplus N_{rest}$. For any open net M' that X_3-conforms to M, the composition $(M' \oplus N_{rest}) \oplus N_2$ is an X_3-open net. ⌟

Corollary 7.1.2 is an application of Theorem 3.1.4. In contrast to Theorem 3.1.4, we do not substitute a party's public view by a private view, but a pattern of a party's public view by another pattern.

In the rest of this chapter, we present several transformation rules; that is, pairs (M, M') of open nets M, M' that fulfill (1) the requirement of the initial and final markings, see Definition 7.1.1 and (2) X_3-conformance. Hence, transformation rules (M, M') can defined independently of any context. The context only matters if we try to identify M to be a pattern of a concrete open net N.

7.2. Projection-inheritance preserving transformation rules

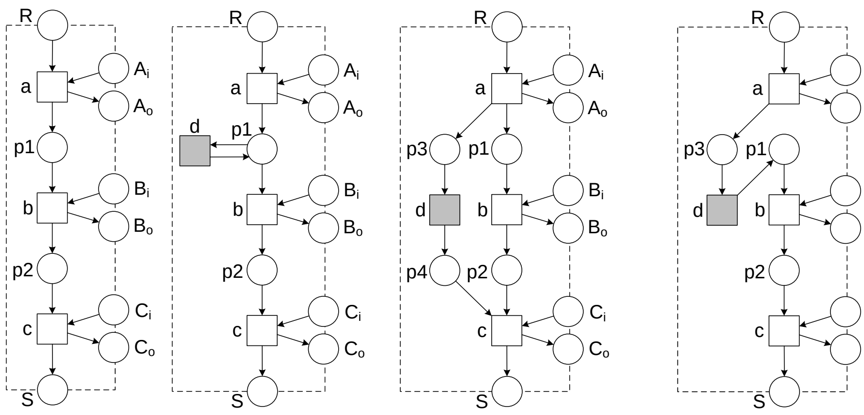
Inheritance is one of the key concepts of object-orientation. In object-oriented design, inheritance is typically restricted to the static aspects (e. g., data and methods) of an object class. In many cases, however, the dynamics is of prime importance. Therefore, *projection inheritance* [BA01, AB02] focuses on the dynamics. Projection inheritance compares process models by establishing a subclass-superclass relationship. The subclass process is indeed a subclass if it inherits particular dynamic properties of its superclass.

Projection inheritance is based on *branching bisimulation* [GW96] (to compare the processes) and *abstraction* (to hide methods). The assumption is that the subclass adds methods to the superclass. The subclass and the superclass are related by projection inheritance if, after hiding all methods from the subclass that are not contained in the superclass, the subclass and the superclass are equivalent.

Projection inheritance was defined for workflow nets in [BA01, AB02], but in this definition projection inheritance refers to "methods" rather than the "sending and receiving of messages". However, projection inheritance can be reformulated for open nets by the following mapping [ALM+08]: An interface transition presents a method, which is present in both the superclass and the subclass.

To decide whether two open nets are related by projection inheritance, it is sufficient to check if their corresponding service automata are branching bisimilar (cf. Definition 2.1.5). In contrast to [BA01, AB02], we do not need to define an abstraction operator. In our mapping, the comparison of the two open nets is restricted to the interface transitions. We abstract from all internal transitions by labeling them with τ. The labeling, however, is fixed in Definition 2.7.2 (transition label) and thus no additional definition of an abstraction is necessary. Consequently, we can define projection inheritance of two open nets as follows.

Definition 7.2.1 (projection inheritance).
Two open nets N and N' are related by *projection inheritance* iff $SA(N)$ and $SA(N')$ are branching bisimilar. ⌟



(a) M_0. (b) M_1: Adding a loop to M_0. (c) M_2: Putting transition d in parallel to b. (d) M_3: Inserting transition d in-between a and b.

Figure 7.3.: X_3-conformance equivalent-preserving transformation rules based on projection inheritance.

Notice that our definition of branching bisimulation (cf. Definition 2.1.5) respects final states. Hence, projection inheritance compares the behaviors of N and N' and also their final markings.

In Section A.1, we proved that fair testing implies X_3-conformance. As it is well-known that branching bisimulation implies fair testing, we can immediately conclude that X_3-conformance is more liberal than projection inheritance; that is, projection inheritance implies X_3-conformance. As projection inheritance is an equivalence, the implication holds even in both directions.

Theorem 7.2.2 (projection inheritance implies conformance).
Let N and N' be two open nets. If N and N' are related by projection inheritance, then N is X_3-conformance equivalent to N'. ⌟

Based on the notion of projection inheritance, three *inheritance-preserving transformation rules* have been defined in [BA01, AB02]. These rules correspond to design patterns for extending a superclass to incorporate new behavior: (1) adding an internal loop, (2) put a new internal transition in parallel with existing transitions, and (3) insert an internal transition in-between existing transitions.

Instead of redefining these rules, we exemplify them in Figure 7.3. Note that Figure 7.3 presents a strong simplification of the rules presented in [BA01, AB02]. In these papers, transitions d correspond to subnets satisfying a particular property. For our purposes, these simplified rules are sufficient.

Figure 7.3(a) represents a pattern M_0 of an open net N. M_0 contains transitions a, b, and c. By Definition 7.1.1, there are no other connections of a, b, c, p1, and p2 than those shown in Figure 7.3(a). Each transition is connected to an input and an output place. However, as indicated by the capital letters, each interface place may correspond to a set of places. Note that $\mathsf{A_i}$, $\mathsf{A_o}$, $\mathsf{B_i}$, $\mathsf{B_o}$, $\mathsf{C_i}$, $\mathsf{C_o}$ do not need to be disjoint. Places R and S denote the input and output places to N_{rest}. Again, R and S may be sets of places. Similar remarks hold for the other three patterns M_1, M_2, and M_3. For example, M_1 is obtained by adding transition d to M_0.

If one considers the behavior of these open nets, then M_0, M_1, M_2, and M_3 are branching bisimilar. Hence, each pair of these four patterns is related by projection inheritance. From Theorem 7.2.2, we conclude that the three transformation rules depicted in Figure 7.3 preserve X_3-conformance equivalence.

Inheritance-preserving transformation rules only change internal transitions of an open net. Next we present transformation rules that affect interface transitions.

7.3. Conformance-equivalence preserving transformation rules

In this section, we present four X_3-conformance-equivalence preserving transformation rules. Given an open net N, each transformation rule specifies a pattern M of N (see Definition 7.1.1), which can be substituted by another open net M' yielding an implementation of N. For each transformation rule, we present a textual description and an illustrating example. Furthermore, we prove that each rule preserves X_3-conformance equivalence. Based on this proof, Corollary 7.1.2 justifies that this substitution preserves all X_3-strategies of N.

Interestingly, the proof of each transformation rule follows always the same proof idea. In a first step, we transform the patterns M and M'. We add to each interface place a transition, and this transition is connected to a new interface place. We did a similar transformation in case of open-net normalization (see Definition 2.7.1). On these transformed nets, we can show that the respective patterns M and M' (i. e., their corresponding service automata) are branching bisimilar.

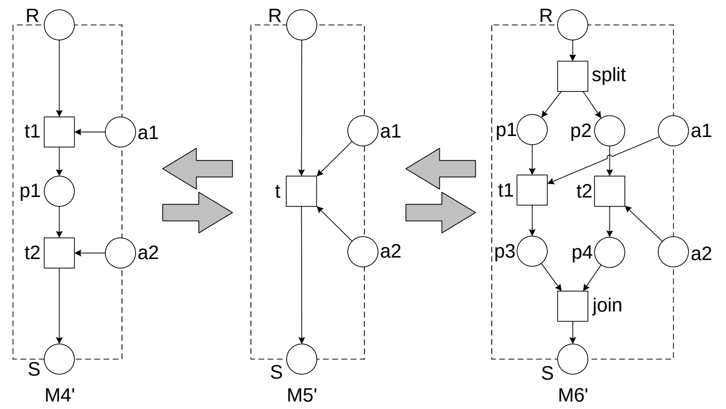
To check branching bisimulation, we remove the newly added interface places; that is, we consider two transition-bordered open nets. That way, we transform an asynchronously communicating open net into a synchronously communicating open net. Transforming an asynchronously communicating model into a synchronously communicating model is a well-known approach in process theory and has also been applied to Petri nets; see [Vog92], for instance. Branching bisimulation compares the state spaces of these open nets. As the state spaces of transition-bordered open nets are infinite, we will represent the branching bisimulation relation symbolically. Informally speaking, we have to show that for every run of M, there exists a run of M' and these two runs fulfill the criteria of branching bisimulation. As branching bisimulation preserves X_3-conformance equivalence, the transformation is valid.

7.3.1. Receive-only rules

The first transformation rule, Rule 1, is restricted to receiving transitions.

Rule 1: receive-only. A sequence of receiving transitions can be executed simultaneously and concurrently. All transformations can be applied in both directions.

Example for Rule 1:



Rule 1 specifies that a sequence of receiving transitions can be merged, and the messages can be received simultaneously. Furthermore, the messages can also be received concurrently. In other words, there is no causality of receiving messages a1 and a2. Rule 1 is formalized by the patterns M_4, M_5, and M_6 in Definitions 7.3.1, 7.3.2, and 7.3.3, respectively.

Pattern M_4 specifies an open net that receives n messages $a_1, \ldots, a_n$ sequentially. Pattern M4′ in Rule 1 illustrates M_4, for $n = 2$.

Definition 7.3.1 (pattern M_4).
Let $n > 0$. Pattern $M_4 = (P, T, F, P^I, P^O, [], \{[]\})$ is defined by

- $P = P^I \cup P^O \cup \{p_1, \ldots, p_{n-1}\}$;
- $T = \{t_1, \ldots, t_n\}$;
- $F = \{(r, t_1) \mid r \in R\} \cup \{(t_n, s) \mid s \in S\} \cup \{(t_i, p_i), (p_i, t_{i+1}) \mid i = 1, \ldots, n-1\} \cup \{(a_i, t_i) \mid i = 1, \ldots, n\}$;
- $P^I = \{a_1, \ldots, a_n\} \cup R$;
- $P^O = S$; ⌟

Pattern M_5 specifies an open net that receives n messages $a_1, \ldots, a_n$ simultaneously. Pattern M5′ in Rule 1 illustrates M_5, for $n = 2$.

Definition 7.3.2 (pattern M_5).
Let $n > 0$. Pattern $M_5 = (P, T, F, P^I, P^O, [\,], \{[\,]\})$ is defined by

- $P = P^I \cup P^O$;
- $T = \{t\}$;
- $F = \quad \{(r,t) \mid r \in R\}$
  $\cup \{(t,s) \mid s \in S\}$
  $\cup \{(a_i,t) \mid i = 1,\dots,n\}$;
- $P^I = \{a_1,\dots,a_n\} \cup R$;
- $P^O = S$. ⌟

Receiving n messages $a_1,\dots,a_n$ concurrently is specified by pattern M_6. Pattern M6$'$ in Rule 1 illustrates M_6, for $n = 2$.

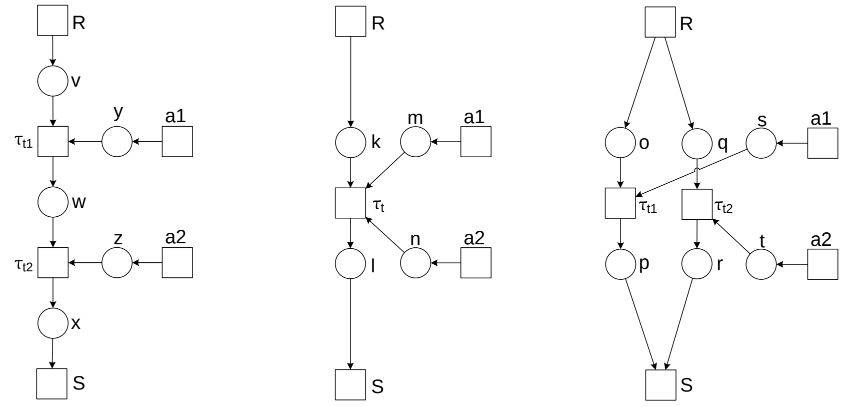
Definition 7.3.3 (pattern M_6).
Let $n > 0$. Pattern $M_6 = (P, T, F, P^I, P^O, [\,], \{[\,]\})$ is defined by

- $P = P^I \cup P^O \cup \{p_1,\dots,p_{2n}\}$;
- $T = \{t_1,\dots,t_n, \mathit{split}, \mathit{join}\}$;
- $F = \quad \{(r, \mathit{split}) \mid r \in R\}$
  $\cup \{(\mathit{join}, s) \mid s \in S\}$
  $\cup \{(\mathit{split}, p_i) \mid i = 1,\dots,n\}$
  $\cup \{(p_i, \mathit{join}) \mid i = n+1,\dots,2n\}$
  $\cup \{(p_i,t_i),(t_i,p_{n+i}) \mid i = 1,\dots,n\}$
  $\cup \{(a_i,t_i) \mid i = 1,\dots,n\}$;
- $P^I = \{a_1,\dots,a_n\} \cup R$;
- $P^O = S$; ⌟

Correctness of Rule 1 is justified by the following lemma.

Lemma 7.3.4 (justification of Rule 1).
Patterns M_4, M_5, and M_6 are X_3-conformance equivalent. ⌟

Proof.
We show that the behavior of the patterns M_4, M_5, and M_6 is branching bisimilar, which is a sufficient condition for X_3-conformance equivalence. To this end, we add a transition to each interface place of all patterns. This transition is labeled with the label of the interface place; all other transitions are relabeled to an internal transition. For the sake of simplicity, we also relabeled all places such that each label consists only of a single symbol without subscript. The resulting patterns (for two receiving transitions) are illustrated in Figure 7.4. As the behaviors of these open nets are infinite, we represent the branching bisimulation relation ϱ symbolically.



(a) modified pattern $\mathsf{M4}'$ (b) modified pattern $\mathsf{M5}'$ (c) modified pattern $\mathsf{M6}'$

Figure 7.4.: Proof of Rule 1. The patterns $\mathsf{M4}'$, $\mathsf{M5}'$, and $\mathsf{M6}'$ are branching bisimilar.

Consider $\mathsf{M4}'$ and $\mathsf{M5}'$ in Figure 7.4. We show that

$$\begin{array}{rrcl} \text{(I)} & v+w+x & = & k+l \\ \text{(II)} & y+w+x & = & m+l \\ \text{(III)} & x+z & = & n+l \end{array}$$

(with $v, w, x, y, z, k, l, m, n \geq 0$) is the required relation ϱ. We have to show, for all transitions in $\mathsf{M4}'$ and $\mathsf{M5}'$, the resulting markings maintain ϱ. Clearly, ϱ holds for the initial markings (where no place is marked). Consider now the τ-transitions. Suppose we are in related markings $(m_4, m_5) \in \varrho$ and firing a τ-transition of $\mathsf{M4}'$ in marking m_4 yields a marking m_4'. Then, (m_4', m_5) must be in ϱ. For example, firing of τ_{t1} yields

$$\begin{array}{rrcl} \text{(I)} & (v-1)+(w+1)+x & = & k+l \\ \text{(II)} & (w+1)+x+(y-1) & = & m+l \\ \text{(III)} & z+x & = & n+l \end{array}$$

Hence, ϱ is maintained. Relation ϱ also holds for transitions τ_{t2} and τ_t.

Next, we consider the case that a visible transition is enabled at a marking m_4 (resp. m_5). Then, this transition must be enabled at m_5 (resp. at m_4) as well, or a run of τ-transitions enabling the respective visible transition exists (cf. Definition 2.1.5). Clearly, transitions R, a, and b are enabled at m_4 if and only if they are enabled at m_5 and again, ϱ is maintained. Suppose S is enabled at m_4 (i. e., $x > 0$). If S is also enabled at m_5 (i. e., $l > 0$), then ϱ is maintained as well. Suppose S is not enabled at m_5 (i. e., $l = 0$). From $x > 0$, $l = 0$, and ϱ we conclude that $k > 0$ (by (I)), $m > 0$ (by (II)), and $n > 0$ (by (III)). Hence, τ_t is enabled at m_5 and firing this transition yields a marking m_5', and S is enabled at m_5'. The other way around, suppose $l > 0$ and $x = 0$; that is, S is only enabled at m_5. Then, we conclude from (III) that $z > 0$. If $w > 0$, τ_{t2} is enabled at m_4 yielding marking m_4', and S is enabled at m_4'. Otherwise, if $w = 0$, we conclude from (I) and (II) that $v > 0$ and $y > 0$. Recall that also $z > 0$. Hence, a run $\tau_{t1}\tau_{t2}S$ starting from m_4 clearly exists. Finally, as the final markings of a pattern are defined by the singleton set of the empty marking, ϱ is trivially fulfilled for the final markings. Hence, ϱ is indeed a branching bisimulation.

Consider $\mathsf{M5}'$ and $\mathsf{M6}'$ in Figure 7.4. We show that

$$\begin{array}{rrcl} \text{(I)} & o+p & = & k+l \\ \text{(II)} & q+r & = & k+l \\ \text{(III)} & s+p & = & m+l \\ \text{(IV)} & t+r & = & n+l \end{array}$$

(with $k, l, m, n, o, p, q, r, s, t \geq 0$) is the required branching bisimulation relation ϱ. Clearly, ϱ holds for the initial marking, the internal transitions, and for transitions R, a, b.

So the only interesting case is the firing of S. Suppose $p > 0$ and $r > 0$, and let $l = 0$. From (I), (III), and (IV) follows that $k > 0$, $m > 0$, $n > 0$; hence, τ_t is enabled and after firing of τ_t, S is enabled. Now suppose $l > 0$, and let $p = 0$ and $r = 1$. We conclude from (III) and (I) that $s > 0$ and $o > 0$. Hence, τ_{t1} is enabled, and firing τ_{t1} enables S. For $p = 1$ and $r = 0$, we conclude from (II) and (IV) that $q > 0$ and $t > 0$. Thus, τ_{t2} is enabled, and firing τ_{t2} enables S. Finally, for $p = 0$ and $r = 0$, we conclude that both, τ_{t1} and τ_{t2} are enabled. As ϱ is for final markings trivially fulfilled, we conclude that ϱ is indeed a branching bisimulation relation. As a branching bisimulation is transitive, $\mathsf{M4}'$ and $\mathsf{M6}'$ are branching bisimilar as well.

These proofs can be generalized (by induction) to any number of receiving transitions. Thus, we conclude that branching bisimulation holds also for the general patterns M_4, M_5, and M_6. □

From Lemma 7.3.4, we conclude that a sequence of receiving transitions can also be reordered while preserving X_3-conformance equivalence.

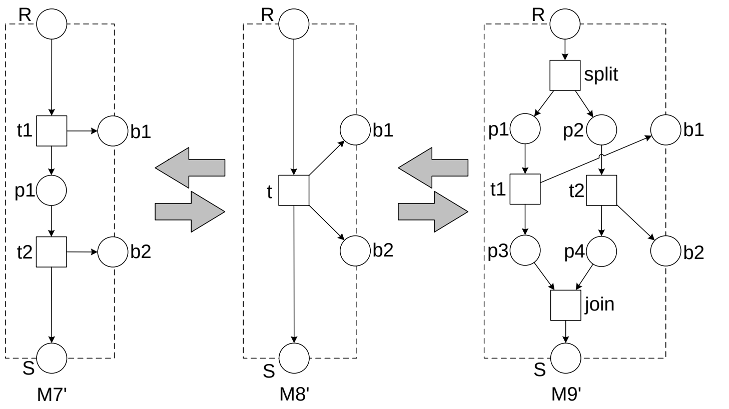
Corollary 7.3.5 (reordering of receiving transitions).
Let pattern M_4 be as defined. Let t_i, t_j be any two transitions of M_4 with $i \neq j$. Let a_i, a_j be any two input places of M_4 with $a_i, a_j \notin R$ and (a_i, t_i), (a_j, t_j). Replacing arcs (a_i, t_i), (a_j, t_j) by (a_i, t_j), (a_j, t_i) preserves X_3-conformance equivalence. ⌟

7.3.2. Send-only rules

The next transformation rule, Rule 2, is restricted to sending transitions. It corresponds to Rule 1, where every receiving transition is replaced by a sending transition.

Rule 2: send-only. A sequence of sending transitions can be executed simultaneously and concurrently. All transformations can be applied in both directions.

Example for Rule 2:



Rule 2 specifies that a sequence of sending transitions can be merged, and the messages can be sent simultaneously. Furthermore, the messages can also be sent concurrently. That means, there is no causality of sending messages b1 and b2. Rule 2 is formalized by the patterns M_7, M_8, and M_9 in Definitions 7.3.6, 7.3.7, and 7.3.8, respectively.

Pattern M_7 specifies an open net that sends n messages $b_1, \ldots, b_n$ sequentially. An illustration of M_7, for $n = 2$, is given by $\mathsf{M7}'$ in Rule 2.

Definition 7.3.6 (pattern M_7).
Let $n > 0$. Pattern $M_7 = (P, T, F, P^I, P^O, [\,], \{[\,]\})$ is defined by

- $P = P^I \cup P^O \cup \{p_1, \ldots, p_{n-1}\}$;
- $T = \{t_1, \ldots, t_n\}$;
- $$\begin{aligned} F = \; & \{(r, t_1) \mid r \in R\} \\ \cup \; & \{(t_n, s) \mid s \in S\} \\ \cup \; & \{(t_i, p_i), (p_i, t_{i+1}) \mid i = 1, \ldots, n-1\} \\ \cup \; & \{(t_i, b_i) \mid i = 1, \ldots, n\}; \end{aligned}$$
- $P^I = R$;
- $P^O = \{b_1, \ldots, b_n\} \cup S$; ⌟

Pattern M_8 specifies an open net that sends n messages $b_1, \ldots, b_n$ simultaneously. An illustration of M_8, for $n = 2$, is given by $\mathsf{M8}'$ in Rule 2.

Definition 7.3.7 (pattern M_8).
Let $n > 0$. Pattern $M_8 = (P, T, F, P^I, P^O, [\,], \{[\,]\})$ is defined by

- $P = P^I \cup P^O$;

- $T = \{t\}$;
- $F = \{(r,t) \mid r \in R\} \cup \{(t,s) \mid s \in S\} \cup \{(t,b_i) \mid i = 1,\ldots,n\}$;
- $P^I = R$;
- $P^O = \{b_1,\ldots,b_n\} \cup S$. ⌟

Sending n messages $b_1,\ldots,b_n$ concurrently is specified by pattern M_9. Pattern M9′ in Rule 2 illustrates M_9, for $n = 2$.

Definition 7.3.8 (pattern M_9).
Let $n > 0$. Pattern $M_9 = (P, T, F, P^I, P^O, [\,], \{[\,]\})$ is defined by

- $P = P^I \cup P^O \cup \{p_1,\ldots,p_{2n}\}$;
- $T = \{t_1,\ldots,t_n, \mathit{split}, \mathit{join}\}$;
- $F = \{(r,\mathit{split}) \mid r \in R\} \cup \{(\mathit{join},s) \mid s \in S\} \cup \{(\mathit{split},p_i) \mid i = 1,\ldots,n\} \cup \{(p_i,\mathit{join}) \mid i = n+1,\ldots,2n\} \cup \{(p_i,t_i),(t_i,p_{n+i}) \mid i = 1,\ldots,n\} \cup \{(t_i,b_i) \mid i = 1,\ldots,n\}$;
- $P^I = R$;
- $P^O = \{b_1,\ldots,b_n\} \cup S$; ⌟

Correctness of Rule 2 is justified by the following lemma.

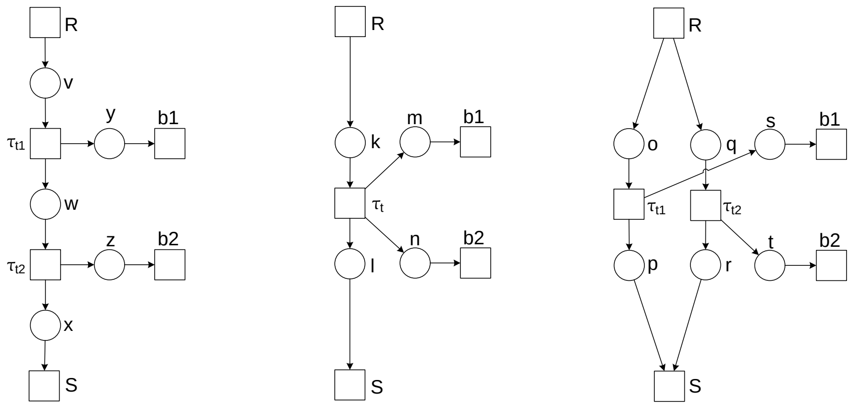
Lemma 7.3.9 (justification of Rule 2).
Patterns M_7, M_8, and M_9 are X_3-conformance equivalent. ⌟

Proof.
We show that the behavior of the patterns M_7, M_8, and M_9 is branching bisimilar, which is a sufficient condition for X_3-conformance equivalence. As in the proof of Lemma 7.3.4 we transform the respective patterns. The resulting patterns (for two receiving transitions) are illustrated in Figure 7.5. As the behaviors of these open nets are infinite, we represent the branching bisimulation relation ϱ symbolically.

Consider M7′ and M8′ in Figure 7.5. We show that

$$\begin{array}{rrcl}
\text{(I)} & v+w+x & = & k+l \\
\text{(II)} & v+y & = & k+m \\
\text{(III)} & v+w+z & = & k+n
\end{array}$$

(with $v,w,x,y,z,k,l,m,n \geq 0$) is the required relation ϱ. Clearly, the proof holds for the initial marking, the internal transitions, and for transition R.



(a) modified pattern M7′ (b) modified pattern M8′ (c) modified pattern M9′

Figure 7.5.: Proof of Rule 2. The patterns M7′, M8′, and M9′ are branching bisimilar.

To prove correctness of transition $b1$, we distinguish the following cases: Relation ϱ holds for $y > 0$ and $m > 0$. For $y > 0$ and $m = 0$ we conclude from (II) that $k > 0$ and hence τ_t can fire which enables $b1$. Likewise, for $y = 0$ and $m > 0$ we conclude from (II) that $v > 0$ and hence τ_{t1} can fire which enables $b1$.

Next, consider transition $b2$. For $z > 0$ and $n > 0$ relation ϱ holds. If $z > 0$ and $n = 0$, we conclude from (III) that $k > 0$ and hence τ_t can fire which enables $b2$. If $z = 0$ and $n > 0$, then there are two cases. Either $w > 0$ and hence τ_{t2} can fire which enables $b2$ or $w = 0$. If $w = 0$, we conclude from (III) that $v > 0$ and hence a run $\tau_{t1}\tau_{t2}$ exists yielding a token on z.

Consider transition S. For $x > 0$ and $l > 0$ relation ϱ holds. If $x > 0$ and $l = 0$, we conclude from (I) that $k > 0$ and hence τ_t can fire which enables S. If $x = 0$ and $l > 0$, then there are two cases. Either $w > 0$ and hence τ_{t2} can fire which enables S or $w = 0$. If $w = 0$, we conclude from (I) that $v > 0$ and hence a run $\tau_{t1}\tau_{t2}$ exists yielding a token on x.

Finally, as the final markings of a pattern are defined by the singleton set of the empty marking, ϱ is trivially fulfilled for the final markings. Hence, ϱ is indeed a branching bisimulation.

Consider M8′ and M9′ in Figure 7.5. We show that

$$\begin{array}{rrcl}
\text{(I)} & o+p & = & k+l \\
\text{(II)} & q+r & = & k+l \\
\text{(III)} & o+s & = & k+m \\
\text{(IV)} & q+t & = & k+n
\end{array}$$

(with $k, l, m, n, o, p, q, r, s, t \geq 0$) is the required branching bisimulation relation ϱ. Clearly, ϱ holds for the initial marking, the internal transitions, and for transition

R. Consider transition $b1$. Relation ϱ holds for $m > 0$ and $s > 0$. For $m > 0$ and $s = 0$ we conclude from (III) that $o > 0$ and hence τ_{t1} can fire which enables $b1$. Likewise, for $m = 0$ and $s > 0$ we conclude from (III) that $k > 0$ and hence τ_t can fire which enables $b1$.

Consider next transition $b2$. Relation ϱ holds for $n > 0$ and $t > 0$. For $n > 0$ and $t = 0$ we conclude from (IV) that $q > 0$ and hence τ_{t2} can fire which enables $b2$. Likewise, for $n = 0$ and $t > 0$ we conclude from (IV) that $k > 0$ and hence τ_t can fire which enables $b2$.

Consider transition S. For $l > 0$, $p > 0$, and $r > 0$ relation ϱ holds. We distinguish three case: If $l > 0$, $p > 0$, and $r = 0$, we conclude from (II) that $q > 0$ and hence τ_{t2} can fire which enables S. If $l > 0$, $p = 0$, and $r > 0$, then $o > 0$ because of (I) and hence τ_{t1} can fire which enables S. If $l = 0$, $p > 0$, and $r > 0$, then we conclude from (II) that $k > 0$ and hence τ_t can fire yielding a token on l.

Finally, as the final markings of a pattern are defined by the singleton set of the empty marking, ϱ is trivially fulfilled for the final markings. Hence, ϱ is indeed a branching bisimulation. As a branching bisimulation is transitive, $\mathsf{M7}'$ and $\mathsf{M9}'$ are branching bisimilar as well.

These proofs can be generalized (by induction) to any number of receiving transitions. Thus, we conclude that branching bisimulation holds also for the general patterns M_7, M_8, and M_9. $\square$

From Lemma 7.3.9, we conclude that a sequence of sending transitions can also be reordered while preserving X_3-conformance equivalence.

Corollary 7.3.10 (reordering of sending transitions).
Let pattern M_7 be as defined. Let t_i, t_j be any two transitions of M_7 with $i \neq j$. Let b_i, b_j be any two output places of M_7 with $b_i, b_j \notin S$ and (t_i, b_i), (t_j, b_j). Replacing arcs (t_i, b_i), (t_j, b_j) by (t_i, b_j), (t_j, b_i) preserves X_3-conformance equivalence. ⌟

7.3.3. Send-and-receive rules

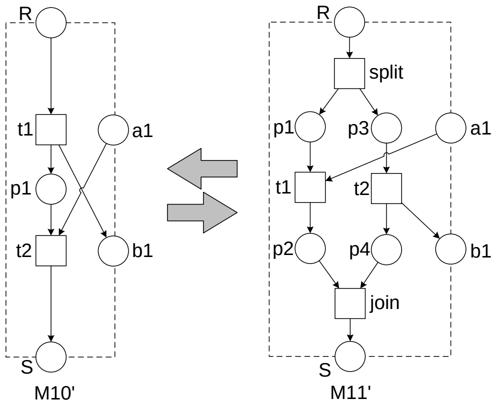
So far, we considered sending and receiving transitions in isolation. In this section, we combine sequences of sending and receiving transitions. We present two X_3-conformance-equivalence preserving transformation rules and show some disallowed rules.

First send and then receive

Transformation rule, Rule 3, shows that a sequence of sending transitions followed by a sequence of receiving transitions can be executed concurrently.

Rule 3: send-receive. A sequence of first sending transitions and then receiving transitions can be executed concurrently and vice versa.

Example for Rule 3:



Rule 3 is formalized by the patterns M_{10} and M_{11} in Definition 7.3.11 and Definition 7.3.12, respectively.

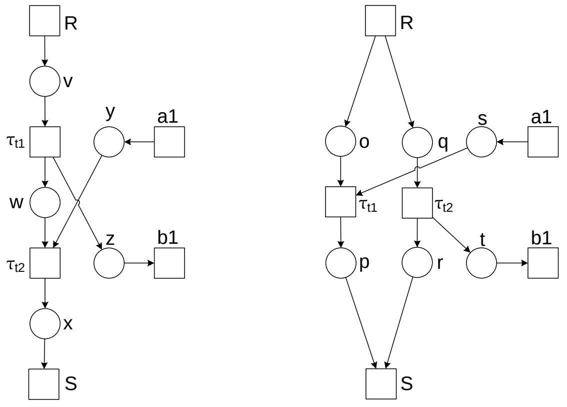
Pattern M_{10} consists of a sequence of n sending transitions, which is followed by a sequence of m receiving transitions. Pattern $\mathsf{M10}'$ in the example for Rule 3 illustrates M_{10}, for $m = 1$ and $n = 1$.

Definition 7.3.11 (pattern M_{10}).
Let $n > 0$ and $m > 0$. Pattern $M_{10} = (P, T, F, P^I, P^O, [\,], \{[\,]\})$ is defined by

- $P = P^I \cup P^O \cup \{p_1, \ldots, p_{m+n-1}\}$;
- $T = \{t_1, \ldots, t_{m+n}\}$;
- $$\begin{aligned} F = \; & \{(r, t_1) \mid r \in R\} \\ \cup \; & \{(t_{m+n}, s) \mid s \in S\} \\ \cup \; & \{(t_i, p_i), (p_i, t_{i+1}) \mid i = 1, \ldots, m+n-1\} \\ \cup \; & \{(a_i, t_{n+i}) \mid i = 1, \ldots, m\} \\ \cup \; & \{(t_i, b_i) \mid i = 1, \ldots, n\}; \end{aligned}$$
- $P^I = \{a_1, \ldots, a_m\} \cup R$;
- $P^O = \{b_1, \ldots, b_n\} \cup S$; ⌟

Pattern M_{11} defines an open net that executes two branches concurrently. The first branch is a sequence of any number n of sending transitions, and the second branch is a sequence of m receiving transitions. Pattern $\mathsf{M11}'$ in the example for Rule 3 illustrates M_{11}, for $m = 1$ and $n = 1$.

Definition 7.3.12 (pattern M_{11}).
Let $n > 0$, $m > 0$. Pattern $M_{11} = (P, T, F, P^I, P^O, [\,], \{[\,]\})$ is defined by



(a) modified pattern M10′ (b) modified pattern M11′

Figure 7.6.: Proof of Rule 3. The patterns M10′ and M11′ are branching bisimilar.

- $P = P^I \cup P^O \cup \{p_1, \ldots, p_{m+n+2}\}$;
- $T = \{t_1, \ldots, t_{m+n}, \mathit{split}, \mathit{join}\}$;
- $$\begin{aligned} F = \; & \{(r, \mathit{split}) \mid r \in R\} \\ \cup \; & \{(\mathit{join}, s) \mid s \in S\} \\ \cup \; & \{(\mathit{split}, p_1), (\mathit{split}, p_{m+2})\} \\ \cup \; & \{(p_{m+1}, \mathit{join}), (p_{m+n+2}, \mathit{join})\} \\ \cup \; & \{(p_i, t_i), (t_i, p_{i+1}) \mid i = 1, \ldots, m\} \\ \cup \; & \{(p_{m+1+i}, t_{m+i}), (t_{m+i}, p_{m+2+i}) \mid i = 1, \ldots, n\} \\ \cup \; & \{(a_i, t_i) \mid i = 1, \ldots, m\} \\ \cup \; & \{(t_{m+i}, b_i) \mid i = 1, \ldots, n\}; \end{aligned}$$
- $P^I = \{a_1, \ldots, a_m\} \cup R$;
- $P^O = \{b_1, \ldots, b_n\} \cup S$; ⌟

Correctness of Rule 3 is justified by the following lemma.

Lemma 7.3.13 (justification of Rule 3).
Patterns M_{10} and M_{11} are X_3-conformance equivalent. ⌟

Proof.
We show that the behavior of the patterns M_{10} and M_{11} is branching bisimilar, which is a sufficient condition for X_3-conformance equivalence. As in the proof of Lemma 7.3.4 we transform the respective patterns. The resulting patterns (for one receiving and one sending transition) are illustrated in Figure 7.6. As the behaviors of these open nets are infinite, we represent the branching bisimulation relation ϱ symbolically.

Consider M10′ and M11′ in Figure 7.6. We show that

$$\begin{array}{rrcl}
\text{(I)} & v+w+x & = & o+p \\
\text{(II)} & v+w+x & = & q+r \\
\text{(III)} & v+z & = & q+t \\
\text{(IV)} & y+x & = & s+p
\end{array}$$

(with $v, w, x, y, z, o, p, q, r, s, t \geq 0$) is the required relation ϱ. Clearly, the proof holds for the initial marking, the internal transitions, and for transitions R and $a1$.

To prove correctness of transition $b1$, we distinguish the following cases: Relation ϱ holds for $z > 0$ and $t > 0$. For $z > 0$ and $t = 0$ we conclude from (III) that $q > 0$ and hence τ_{t2} can fire which enables $b1$. Likewise, for $z = 0$ and $t > 0$ we conclude from (III) that $v > 0$ and hence τ_{t1} can fire which enables $b1$.

Consider transition S. For $x > 0$, $p > 0$, and $r > 0$ relation ϱ holds. If $x > 0$, $p > 0$, and $r = 0$, we conclude from (II) that $q > 0$. Hence τ_{t2} can fire which enables S. If $x > 0$, $p = 0$, and $r > 0$, we conclude from (I) and (IV) that $o > 0$ and $s > 0$, respectively. Hence, τ_{t1} can fire which enables S. The case $x > 0$, $p = 0$, and $r = 0$ follows the same arguments. Consider now $x = 0$, $p > 0$, and $r > 0$. By (IV) we know that $y > 0$. We distinguish two cases. If $w > 0$, then τ_{t2} can fire which enables S. If $w = 0$, then we conclude from (II) that $v > 0$. Hence there exists a run $\tau_{t1}\tau_{t2}$ yielding a token on x.

Finally, as the final markings of a pattern are defined by the singleton set of the empty marking, ϱ is trivially fulfilled for the final markings. Hence, ϱ is indeed a branching bisimulation.

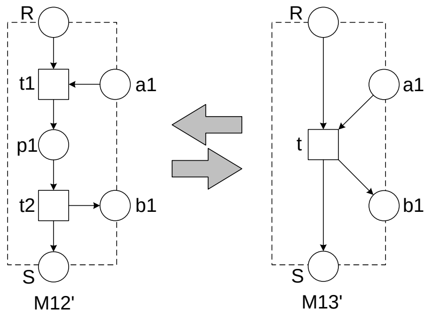
This proof can be generalized (by induction) to any number of sending and receiving transitions. Thus, we conclude that branching bisimulation holds also for the general patterns M_{10} and M_{11}. □

First receive and then send

So far we excluded the possibility that a receiving transition can be followed by a sending transition. Rule 4 specifies that a sequence of receiving transitions followed by a sequence of sending transitions can also be executed simultaneously and vice versa.

Rule 4: receive-send. A sequence of first receiving transitions and then sending transitions can be executed simultaneously and vice versa.

Example for Rule 4:



Rule 4 is formalized by the patterns M_{12} and M_{13} in Definition 7.3.14 and Definition 7.3.15, respectively.

Pattern M_{12} executes a sequence of first m receiving transitions and then n sending transitions. Pattern $\mathsf{M12}'$ in the example for Rule 4 illustrates M_{12}, for $m = 1$ and $n = 1$.

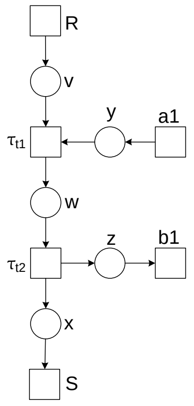
Definition 7.3.14 (pattern M_{12}).
Let $n > 0$ and $m > 0$. Pattern $M_{12} = (P, T, F, P^I, P^O, [\,], \{[\,]\})$ is defined by

- $P = P^I \cup P^O \cup \{p_1, \ldots, p_{m+n-1}\}$;
- $T = \{t_1, \ldots, t_{m+n}\}$;
- $F = \quad \{(r, t_1) \mid r \in R\}$
  $\cup \{(t_{m+n}, s) \mid s \in S\}$
  $\cup \{(t_i, p_i), (p_i, t_{i+1}) \mid i = 1, \ldots, m + n - 1\}$
  $\cup \{(a_i, t_i) \mid i = 1, \ldots, m\}$
  $\cup \{(t_{m+i}, b_i) \mid i = 1, \ldots, n\}$;
- $P^I = \{a_1, \ldots, a_m\} \cup R$;
- $P^O = \{b_1, \ldots, b_n\} \cup S$; ⌟

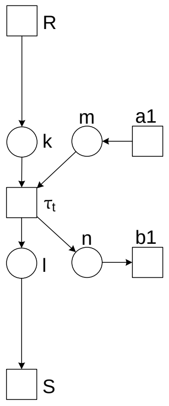
An open net that executes simultaneously m receiving and n sending transitions is specified by pattern M_{13}. Pattern $\mathsf{M13}'$ in the example for Rule 4 illustrates M_{13}, for $m = 1$ and $n = 1$.

Definition 7.3.15 (pattern M_{13}).
Let $n > 0$ and $m > 0$. Pattern $M_{13} = (P, T, F, P^I, P^O, [\,], \{[\,]\})$ is defined by

- $P = P^I \cup P^O$;
- $T = \{t\}$;



(a) modified pattern M12′



(b) modified pattern M13′

Figure 7.7.: Proof of Rule 4. The patterns M12′ and M13′ are branching bisimilar.

- $F = \{(r,t) \mid r \in R\}$
  $\cup \{(t,s) \mid s \in S\}$
  $\cup \{(a_i,t) \mid i = 1,\ldots,m\}$
  $\cup \{(t,b_i) \mid i = 1,\ldots,n\}$;
- $P^I = \{a_1,\ldots,a_m\} \cup R$;
- $P^O = \{b_1,\ldots,b_n\} \cup S$; ⌟

With the following lemma we prove correctness of Rule 4.

Lemma 7.3.16 (justification of Rule 4).
Patterns M_{12} and M_{13} are X_3-conformance equivalent. ⌟

Proof.
We show that the behavior of the patterns M_{12} and M_{13} is branching bisimilar, which is a sufficient condition for X_3-conformance equivalence. As in the proof of Lemma 7.3.4 we transform the respective patterns. The resulting patterns (for one receiving and one sending transition) are illustrated in Figure 7.7. As the behaviors of these open nets are infinite, we represent the branching bisimulation relation ϱ symbolically.

Consider M12′ and M13′ in Figure 7.7. We show that

$$\begin{array}{rrcl}
\text{(I)} & v+w+x & = & k+l \\
\text{(II)} & y+w+x & = & m+l \\
\text{(III)} & v+w+z & = & k+n \\
\text{(IV)} & y+w+z & = & m+n
\end{array}$$

(with $v,w,x,y,z,k,l,m,n \geq 0$) is the required relation ϱ. Clearly, the proof holds for the initial marking, the internal transitions, and for transitions R and $a1$.

To prove correctness of transition $b1$, we distinguish the following cases: Relation ϱ holds for $z > 0$ and $n > 0$. For $z > 0$ and $n = 0$ we conclude from (III) and (IV) that $k > 0$ and $m > 0$, respectively. Hence τ_t can fire which enables $b1$. If $z = 0$ and $n > 0$, we distinguish two cases: If $w > 0$, then τ_{t2} can fire which enables $b1$. Otherwise, if $w = 0$, then we conclude from (III) and (IV) that $v > 0$ and $y > 0$, respectively. Hence, there exists a run $\tau_{t1}\tau_{t2}$ yielding a token on z.

Consider transition S. For $x > 0$ and $l > 0$ relation ϱ holds. If $x > 0$ and $l = 0$, we conclude from (I) and (II) that $k > 0$ and $m > 0$, respectively. Hence τ_t can fire which enables S. If $x = 0$ and $l > 0$, we distinguish two cases: If $w > 0$, then τ_{t2} can fire which enables S. Otherwise, if $w = 0$, then we conclude from (I) and (II) that $v > 0$ and $y > 0$, respectively. Hence, there exists a run $\tau_{t1}\tau_{t2}$ yielding a token on x.

Finally, as the final markings of a pattern are defined by the singleton set of the empty marking, ϱ is trivially fulfilled for the final markings. Hence, ϱ is indeed a branching bisimulation.

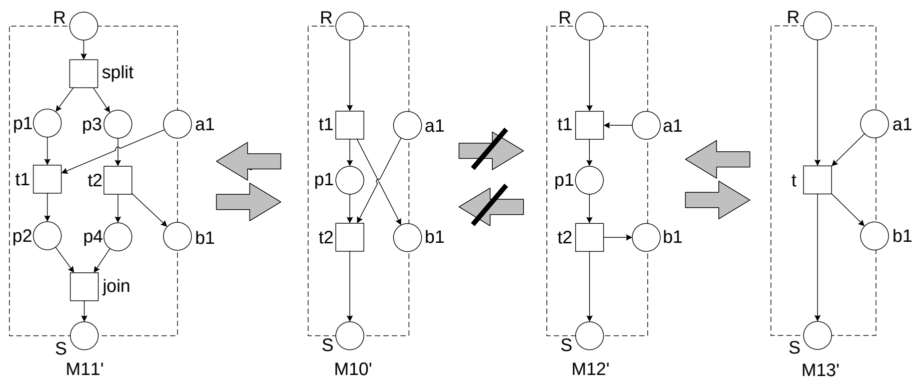
This proof can be generalized (by induction) to any number of sending and receiving transitions. Thus, we conclude that branching bisimulation holds also for the general patterns M_{12} and M_{13}. $\square$

Anti-rules

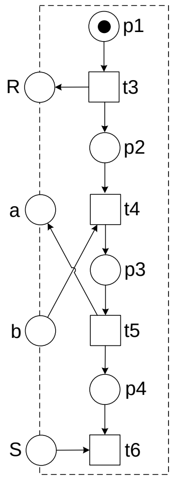
Now we combine the results of Rule 3 and Rule 4 and show that further transformations do not hold. To this end, we present some anti-rules. Such an anti-rule refers to a problematic modification of a service; that is, a modification that violates X_3-conformance.

Anti-rule. A sequence of first sending and then receiving cannot be transformed into first receiving and then sending and vice versa.

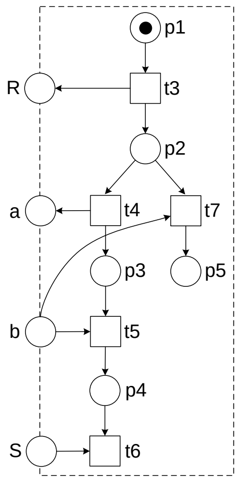
Example for Anti-rule:



The anti-rule shows that first sending and then receiving cannot be reordered in general: M_{10} does not X_3-conform to M_{12}, and M_{12} does not X_3-conform to M_{10}. This anti-rule shows that there is a causality if we first receive a message



(a) X_3-strategy of M10′ but not of M12′.



(b) X_3-strategy of M12′ but not of M10′.

Figure 7.8.: Counterexamples.

and afterwards send a message. It can be easily observed that for example M10′ and M12′ are not branching bisimilar: In the former open net first a token has to produced on R and then t1 can be fired yielding a token on b1, but not in the latter open net. However, proving that a branching bisimulation does not exist is not enough to prove that X_3-conformance does not hold. Hence, we present a concrete counterexample.

Suppose the final markings to be equivalent to the singleton set with the empty marking. Then, the open net in Figure 7.8(a) is an X_3-strategy of M10′, but no X_3-strategy of M12′, and the open net in Figure 7.8(b) is an X_3-strategy of M12′ but not of M10′.

From the counterexamples in Figure 7.8 it follows that first receiving and then sending (cf. M_{12}) cannot be transformed into a pattern that sends and receives concurrently (e. g., M_{11}), because we could transform the latter open net into M_{10} by applying Rule 3. Consequently, first receiving then sending does not X_3-conform to sending and receiving concurrently and vice versa. Analogously, first sending then receiving (M_{10}) cannot be transformed into sending and receiving simultaneously (M_{13}), because the latter can be transformed into M_{12} by applying Rule 4. Thus, first sending then receiving does not X_3-conform to sending and receiving simultaneously and vice versa.

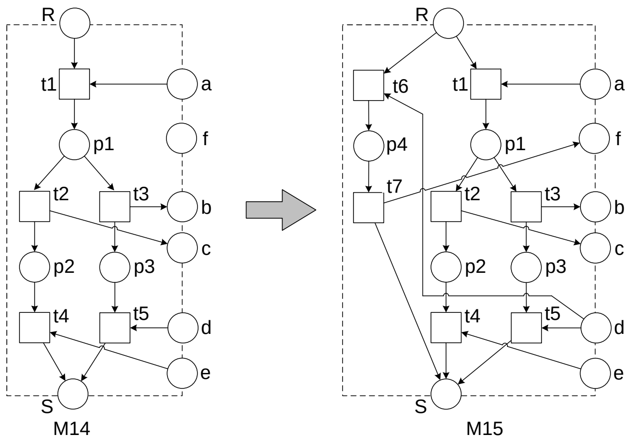
7.4. Conformance-preserving transformation rules

The transformation rules presented in the previous section preserve X_3-conformance equivalence. In this section, we show two rules that preserve X_3-conformance only in one direction. Consequently, branching bisimulation cannot be applied for justifying correctness of these rules. Unfortunately, so far we do not have a proper proof technique to decide X_3-conformance of two patterns. So the idea is to put some restrictions on the context of a pattern such that the decision procedure reduces to X_1-conformance. For X_1-conformance, we presented in Section 5.1 a decision algorithm based on X_1-operating guidelines. We apply these algorithm and prove the correctness of the two transformation rules for restricted contexts and a fixed communication bound.

7.4.1. Adding an alternative branch

The first X_3-conformance-preserving transformation, Rule 5, specifies a way to add an alternative branch to a pattern M_{14}.

Rule 5: adding an alternative branch.



The pattern M_{14} first receives a and then enters either the left or the right branch. In the left (right) branch, message b (c) is sent, and then message d (e) is received. The pattern M_{14} can be transformed into M_{15} by adding an alternative branch. In this branch, d is received, and then a message f is sent. Afterwards, there is a direct continuation in S. Rule 5 preserves X_3-conformance in only one direction. The intuition behind this rule is that any X_3-strategy of M_{15} has to wait for the decision of M_{14} which branch it will enter. Otherwise, it could happen that an environment sends d, but M_{14} enters the left branch and waits for message e.

As we do not have a proof technique for X_3-conformance yet, we prove the correctness of this transformation rule for a certain context. Recall that for acyclic and bounded open nets weak termination and deadlock freedom coincide; see Lemma 2.6.3. Consequently, X_3-conformance reduces to X_1-conformance. For a fixed communication bound, we can decide X_1-conformance; see Section 5.1. Hence, we require that M_{14} is a pattern of an acyclic and bounded open net N.

To apply Lemma 2.6.3 to M_{14}, we have to make sure that M_{14} is acyclic and bounded as well. The latter is, however, not the case, because the inner subnet of M_{14} is unbounded. To overcome this, we have to put some additional restriction to the context N of M_{14}. We require that transition `t1` can fire at most once in the inner subnet of N. In this case, we can add a place p_0 that is initially marked and an arc $(p_0, \texttt{t1})$ to M_{14}. Clearly, adding p_0 does not change the semantics of N; that is, transition `t1` still fires at most once in the inner subnet of N. This construction has a nice effect. The inner subnet of M_{14} is now bounded, and we have a fixed initial marking $m_0 = [p_0]$ for M_{14}.

Definition 7.4.1 (pattern M_{14}).
Let N be an acyclic and bounded open net. Open net M_{14} *is a pattern of* N iff transition `t1` fires at most once in the inner subnet of N. ⌟

Lemma 7.4.2 (Rule 5: adding an alternative branch).
Let N be an open net such that M_{14} is a pattern of N. Pattern M_{14} X_3-conforms to pattern M_{15} in the context of N. ⌟

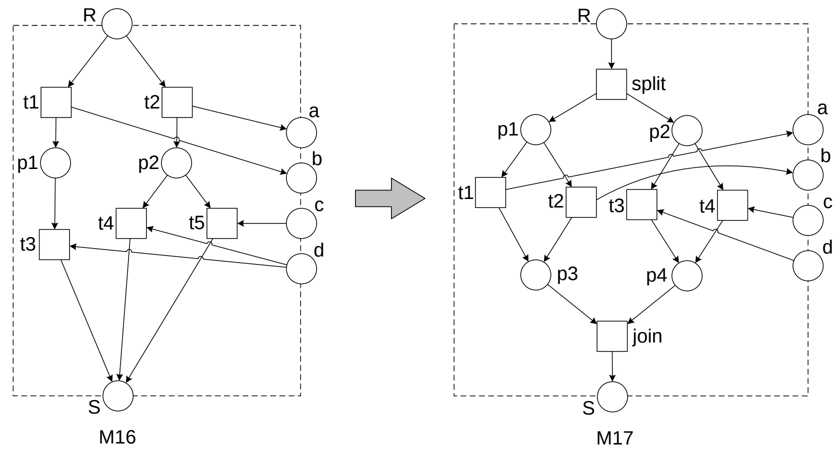
To prove that M_{15} X_3-conforms to M_{14}, we restrict the sets R and S of both patterns to singleton sets. As described above, we further add to each pattern a place p_0 that is initially marked. In M_{14}, we add an arc $(p_0, \texttt{t1})$, and in M_{15} we add two arcs $(p_0, \texttt{t1})$ and $(p_0, \texttt{t6})$. Hence, we have the initial marking, $m_0 = [p_0]$, and define the set of final markings by $\Omega = \{[\,]\}$. For the resulting open nets, the tool Fiona calculated the X_1-operating guidelines and checked that $OG_{X_1}(M_{15})$ refines $OG_{X_1}(M_{14})$. As the two X_1-operating guidelines have too many states, we do not show them.

Note that the other direction of the lemma does not hold: An open net that first sends message d and then waits for message f is an X_1-strategy of M_{15} but not of M_{14}.

7.4.2. Parallelization send-receive

Subsequently, we present another X_3-conformance-preserving transformation rule. The idea is to add more behavior by adding concurrency.

Rule 6: parallelization send-receive.



Starting point is a pattern M_{16} with an *implicit choice.* Pattern M_{16} sends either message a or message b. If the left branch is entered, then only message d can be received, whereas in the right branch either message c or message d can be received. Clearly, we can add behavior to pattern M_{16} if we add this choice also to the left branch. Instead of adding this choice to the left branch, pattern M_{17} executes the sending and the receiving choice concurrently.

To prove that pattern M_{16} X_3-conforms to pattern M_{17}, we follow the same approach as in the previous section and restrict the context of M_{16}.

Definition 7.4.3 (pattern M_{16}).
Let N be an acyclic and bounded open net. Open net M_{16} *is a pattern of* N iff the sum of firings of transitions t1 and t2 in the inner subnet of N is at most 1.⌟

Lemma 7.4.4 (Rule 6: parallelization send-receive).
Let N be an open net such that M_{16} is a pattern of N. Pattern M_{16} X_3-conforms to pattern M_{17} in the context of N. ⌟

We apply the same proof strategy as for Lemma 7.4.2 and prove the correctness of Lemma 7.4.4 using the tool Fiona. Again, we do not show the corresponding X_1-operating guidelines, because they have too many states.

The other direction of this lemma does not hold; for example, an open net that first sends message c and afterwards can receive either message a or message b is an X_1-strategy of M_{17}, but not of M_{16}.

8. Extending the Equivalence Notion for Abstract WS-BPEL Processes

BPEL is a standard for executable processes. An executable process is a business process that can be automated through an IT infrastructure. The BPEL specification also introduces the concept of *abstract processes*. In contrast to an executable process, an abstract process is not executable and can have parts where business logic is hidden.

The BPEL specification introduces a notion of *observable equivalence* between an abstract process and an executable process. Basically, this equivalence notion defines a set of syntactical rules that can be augmented or restricted by *profiles*. As a limiting factor, none of these profiles considers the semantics (i. e., the behavior) of the processes. Consequently, the set of executable processes that are observable equivalent to an abstract process is unnecessarily restricted.

In this chapter, we present a more general approach to decide equivalence between an abstract and an executable process. We propose a novel profile and define an equivalence on the observable behavior of processes; see also Figure 1.4. To this end, we adapt our notion of X_3-conformance equivalence to BPEL processes and reformulate the X_3-conformance-equivalence preserving transformation rules, which we defined in Chapter 7. These reformulated rules provide a necessary condition to decide whether two BPEL processes are equivalent. That way, more executable processes can be considered equivalent to an abstract process without the loss of general applicability.

In the remainder of this chapter, we sketch the basic concepts of BPEL (Section 8.1) and introduce the concept of abstract processes (Section 8.2). Section 8.3 presents the novel abstract profile and defines a notion of equivalence based on a set of transformation rules. The results of this chapter have been published in [KLM$^+$08].

8.1. A glimpse on BPEL

The *Web Services Business Process Execution Language* (WS-BPEL, or BPEL for short) offers a standards-based approach to build distributed applications for business-to-business interactions. A BPEL process implements one Web service by specifying the interactions with other Web services. This allows for building flexible business processes by orchestrating multiple other Web services. Such applications then follow the architectural pattern of an SOA. Business processes implemented in BPEL are a key element in an SOA infrastructure [ABH+07].

The BPEL specification [Alv07] distinguishes two different kinds of business processes: *executable processes* and *abstract processes*. Executable processes must contain all the details that are necessary to be executed by a BPEL engine. To execute a BPEL process means to create a *running instance* of this process. In contrast, abstract processes are not executable and can have parts where business logic is left unspecified or explicitly marked as opaque (i. e., hidden). In the following, we introduce the basic language constructs of BPEL.

For the specification of the internal behaviour of a business process, BPEL provides two kinds of *activities*: *basic* activities and *structured* activities. A basic activity models an elementary action in the process, whereas a structured activity defines some causal order between other activities. Structured activities can be nested, and the activity structure in the process resembles a tree, where the process itself is its root and the basic activities are its leafs. As such, any structured activity contains a number of child activities.

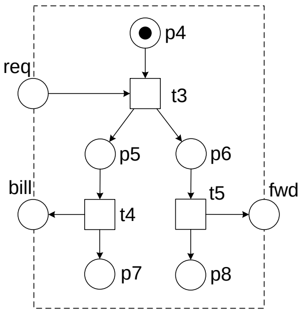
A basic activity can communicate with other processes by messages (*invoke*, *receive*, *reply*), manipulate or check data (*assign*, *validate*), wait for some time (*wait*) or just do nothing (*empty*), signal faults (*throw*), or end the entire process instance (*exit*).

The structured activities are sequential execution (*sequence*), parallel execution (*flow*), data-dependent branching (*if*), timeout- or message-dependent branching (*pick*), and repeated execution (*while*, *repeatUntil*, *forEach*). The most important structured activity is a *scope*. It links an activity to a transaction management and provides fault, compensation, event, and termination handling. A *process* is the outmost scope of the described business process.

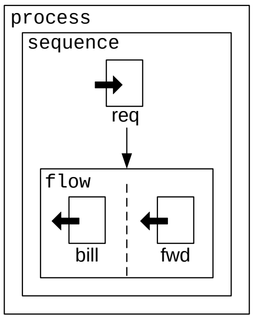
Activities, which are embedded in a flow, can be further ordered by *links*. A link connects a source activity with a target activity. The source may specify a Boolean expression, the status of the link. The target may also specify a Boolean expression—that is, the *join condition*—which evaluates the status of all incoming links. The target activity is only executed when its join condition holds.

For each communicating activity, a BPEL process specifies a partner link and an operation. A *partner link* defines the name of the Web service that sends and receives the message, respectively; an (WSDL) *operation* specifies the corresponding channel the message uses.

As an illustration, Figure 8.1 depicts again the open-net model of the broker service. Recall that the broker receives a credit request from a client. Then he concurrently forwards the request to a credit service and sends the bill to the client. Figure 8.1(b) graphically illustrates the broker specified in BPEL. Receiving the client's request is modeled by a receive activity; each sending is modeled by an invoke activity. The ordering of these three activities is achieved by embedding the two invoke activities into a flow and sequentially execute the receive activity and the flow. Finally, the sequence is embedded into a process. Figure 8.1(c) shows the respective code snippet. The code snippet shows that for



(a) open net Broker



(b) Broker specified in BPEL

```
<process name="broker" ...>
  <partnerLinks>
    <partnerLink name="client" ... />
    <partnerLink name="credit" ... />
  </partnerLinks>
  <sequence>
    <receive partnerLink="client" operation="req" ... />
    <flow>
      <invoke partnerLink="client" operation="bill" ... />
      <invoke partnerLink="credit" operation="fwd" ... />
    </flow>
  </sequence>
</process>
```

(c) BPEL code snippet of the Broker

Figure 8.1.: Illustration of BPEL using the broker service of Figure 2.7.

every partner of the broker—that is, the client and the credit service—a partner link is defined, and an operation refers to the respective message channel.

8.2. Abstract processes in BPEL

The BPEL standard introduces the notion of *abstract processes*. An abstract process is either used to hide language elements of an executable process or is not yet fully specified. We explain the idea of abstract processes by the help of two use cases.

In a first use case, an abstract process serves as a *public view* of an executable BPEL process. A BPEL process usually implements a stateful Web service, which requires its exposed operations to be invoked in a particular order. Therefore, if an enterprise publishes a process P, it must not only provide a description of the interface of P, but it also might want to specify a public view of P. Such a public view—in BPEL, the term *interaction protocol* is used—can be represented by an abstract process. Like in the setting of service contracts, the abstract process describes the rules of engagement that P has to fulfill. The abstract process shows only the externally observable behavior of P while hiding other process model elements; that is, it avoids the disclosure of internal (potentially confidential) business process logic. A partner interacting with P would then have to design its service such that it is compatible to the abstract process of P.

In a second use case, an abstract process serves as a *template for further refinement*. To illustrate this, imagine a business analyst capturing and sketching out a business process. This business process is recorded as an abstract process by omitting all the technical details. In a next step, this abstract process is handed over to an IT department to add details required for the process to become executable, but are not relevant for the business. Such an abstract process may have been generated by business-level process modeling tools.

The syntax of an abstract process is defined by the *Common Base*. The Common Base also describes two types of transformations between abstract and executable processes. The first transformation is a replacement of explicitly modeled *opaque* activities of an abstract process by activities of an executable process. Alternatively, it is also possible to replace an opaque expression by a concrete expression—for example, the condition of a while loop. The second transformation consists of *inserting* entities of an executable process at specific places in a process model; for example, it is possible to insert a start activity. Reordering of activities, removing of activities, or changing the control flow is never permitted.

Based on these transformations, the Common Base defines two relations between an abstract process and a set of executable processes: the *Executable Completion* and the *Basic Executable Completion*. The Executable Completion allows to use both transformations, and it requires that the resulting executable process satisfies all BPEL static validation rules. The Basic Executable Completion is more restrictive than the Executable Completion, because it limits the allowed transformations; for example, it is not allowed to add start activities. The set of executable processes, which are related to an abstract process according to these two relations, are the *executable completions*.

Beyond these two general syntactical transformations, the specification further allows to define additional syntactical restrictions by means of *profiles*. A profile defines a set of rules that specify syntactically a subset of executable completions. For abstract processes, the BPEL standard provides two concrete profiles: the *Abstract Process Profile for Observable Behavior* (APPOB) and the *Abstract Process Profile for Templates* (APPT). The APPOB handles the bottom-up way of providing a public view generated out of an executable process; see the first use case. In contrast, the APPT describes a top-down refinement, where abstract processes can be used as an exchange format between different roles within an enterprise; see the second use case.

The abstract process profiles are used to define permitted completions of abstract processes to executable processes. Figure 8.2 illustrates the relationship between abstract and executable processes as well as abstract process profiles.

Both use cases for abstract processes have in common that the abstract process

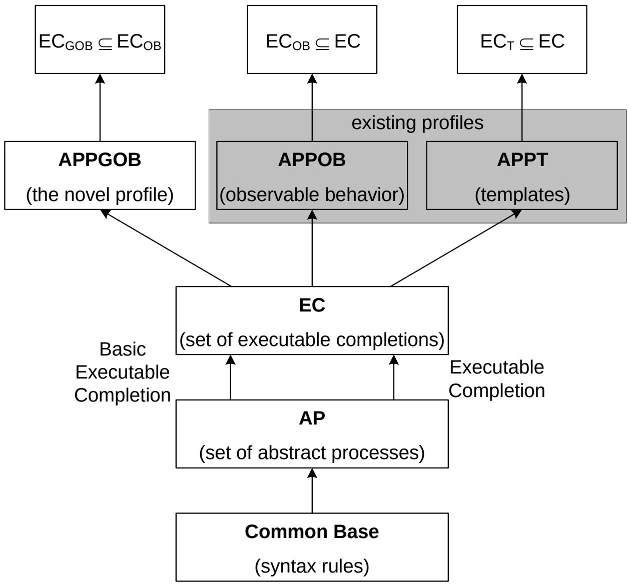


Figure 8.2.: Relationships between abstract processes, executable completions, and abstract process profiles.

definition is regarded as a *specification*, whereas an executable process can then be seen as one *implementation* thereof. When one artifact serves as a specification and one artifact provides an implementation of that specification, obviously there must be a way to ensure that they *conform* to each other[1]. The BPEL specification defines that an executable process conforms to an abstract one if "the executable process is one of the executable completions in the set of permitted completions specified by the abstract process profile". The definition of an executable completion *stops at a syntactical level.* Additional syntactic rules are provided by profiles addressing particular use cases. These rules restrict the set of allowed executable completions.

As we are interested in the public view use case, we consider the APPOB in more detail. The set of executable completions of an abstract process is restricted by this profile such that the externally observable interactions defined in the abstract process are preserved in all its executable completions. In other words, when creating an executable process, no transformation must be applied that modifies interactions via partner links already defined in the abstract process. For example, according to the APPOB, in abstract processes

- join conditions are not allowed to be hidden (otherwise, the synchronization of links and hence the control flow could be changed);

[1]The BPEL specification uses the term compatibility instead of conformance.

- an exit activity must not be inserted (otherwise, the process could end at a different point of the control flow);
- the nesting structure of structured activities around any activity in an abstract process remains unchanged; for example, it is disallowed to insert a loop activity as a new parent of an existing activity;
- the ability to introduce new branches, handlers, links to existing activities, or scoped declarations is substantially restricted, in particular, modifications must be avoided, which affect the branching behavior in a way that conflicts with the specifications in the abstract process;
- the ability to throw new faults is limited to avoid affecting the existing control flow; and
- new partner links may be added and used in additional communicating activities.

Without going into the details of the rules specified by the APPOB, consider the following example, which shows the actual weakness of this profile. Suppose an abstract process specifies that first message a is sent and then message b is sent. In this case, an executable process that executes the sending of a and b concurrently is not permitted according to the APPOB. So the APPOB excludes executable processes that have the same observable behavior, but do not follow the syntactical restrictions.

In the next section, we define a novel profile (see Figure 8.2) and provide some additional rules to check equivalence.

8.3. A novel abstract profile for BPEL

This section defines a novel abstract profile for BPEL. Motivated by the limitations of the APPOB, we extend the set of executable completions of an abstract process P by all those executable processes that are behaviorally equivalent to P. To this end, we formally define behavioral equivalence of BPEL processes by X_3-conformance equivalence. To decide behavioral equivalence, we reformulate the X_3-conformance-equivalence preserving transformation rules of the previous chapter.

As the BPEL specification does not make assumptions about protocols, bindings, and quality of service attributes of interactions, it is necessary to distinguish between transformation rules that are valid for synchronous bindings (Section 8.3.1) and those that are also valid for asynchronous bindings (Section 8.3.2). Section 8.3.3 presents some disallowed transformation rules and, finally, Section 8.3.4 discusses the achieved results.

8.3.1. An equivalence notion for BPEL processes

The APPOB in BPEL allows adding new partner links and communicating activities using these new partner links. We introduce a variation of the APPOB that preserves the observable behavior globally; that is, the set of *all* partner links and interactions across these partner links remain invariant.

We take the existing APPOB and introduce a new profile, the *Abstract Process Profile for Globally Observable Behavior* (APPGOB). The only difference to the APPOB is that we do not permit adding new partner links as a part of the executable completion. In a sense, we thereby restrict the set of executable completions. Figure 8.2 illustrates how the APPGOB is related to the existing profiles.

Furthermore, the Common Base is too restrictive in only allowing replacements of opaque entities and insertions of additional executable entities. There exist a number of cases where the reordering of activities as well as the deletion of activities can be tolerated if these transformations do not affect the main intention of the profile. The APPOB (as well as the more restrictive APPGOB) concentrate on interactions across partner links. Other activities, such as simple assignments, do not need to be handled in such a restrictive fashion. For example, several assignment operations may be independent in the sense that reordering them has no effect on the externally observable behavior of the process. So we want to extend the set of executable completions to a set of executable processes exposing the *same observable behavior*.

It turns out that behavioral equivalence coincides with X-conformance equivalence, for $X = \{\text{weak termination, strict termination}\}$, a notion we introduced in Definition 3.1.5. Two services are X-conformance equivalent if and only if no X-strategy can distinguish between them. Hence, we have a formalization of behavioral equivalence. Given an abstract process P, we are interested in all executable processes P' such that P' is behaviorally equivalent to P. It is not difficult to see that every executable process P' of the set EC_{GOB} of executable completions of P allowed by the APPGOB is behaviorally equivalent to P.

Figure 8.3 shows the relationships between the different executable completions. The outmost ellipse depicts all executable completions EC according to the Common Base. A subset of this set are those executable completions that follow the APPOB, and the APPOB embeds those executable completions that follow the ABBGOB. This set is, however, only a subset of the behaviorally equivalent processes we are interested in.

Unfortunately, we do not have a method to construct a finite representation of all behaviorally equivalent processes P' of a process P. However, the X_3-conformance-equivalence preserving transformation rules in Chapter 7 provide a method to calculate a subset of all X_3-conformance equivalent processes of P (recall that these transformation rules also preserve strict termination). After reformulating them into BPEL, they can be applied in our setting. In other words, we extend the set EC_{GOB} of P by all those executable processes P'' that can be

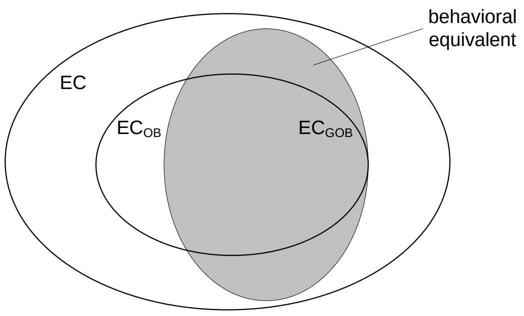


Figure 8.3.: Relationship between executable completions (EC), executable completions following the APPOB (EC_{OB}), executable completions following the APPGOB (EC_{GOB}), and executable processes in EC that are behaviorally equivalent. EC_{GOB} is the intersection of EC_{OB} and the set of behaviorally equivalent processes.

derived from any $P' \in EC_{GOB}$ by applying zero or more of the (reformulated) X_3-conformance-equivalence preserving transformation rules presented in Chapter 7.

Definition 8.3.1 (equivalence of observable behavior).
Let P be an abstract process, and let EC_{GOB} be its executable completion allowed by the APPGOB. An executable process P'' *is behaviorally equivalent to* P iff $P'' \in EC_{GOB}$ or there is an $P' \in EC_{GOB}$, and P'' can be derived from P' by applying zero or more of the following seven transformation rules:

1. Looping existing activity
2. Activity removal from sequence
3. Activity removal from flow
4. Activity reordering
5. Invoke-flow serialization
6. Receive-flow serialization
7. Invoke and receive ⌟

In the following, we introduce these rules. We distinguish between the first four rules on the one hand, as they consider only non-communicating activities and the remaining rules for communicating activities, on the other hand. For each rule, we present a textual description and an illustrating example by help of a code snippet. On the left-hand side of each example the part of the existing BPEL process is shown, whereas on the right-hand side the respective part after applying the transformation is illustrated. Note that every rule is a reformulation of an X_3-conformance-equivalence preserving transformation rule of Chapter 7 that justifies the application of this rule in this setting.

Rules for non-communicating activities

Based on the projection-inheritance preserving transformation rules of Figure 7.3 in Section 7.2, we identify the following four rules for non-communicating activities.

Rule 1: looping existing activity. Given any sequence of activities, we can embed a present non-communicating activity into a finite (while/repeatUntil/forEach) loop.

Example for Rule 1:

```
<sequence>
  <activity1 />
  <activity2 />
  <activity3 />
</sequence>
```

```
<sequence>
  <activity1 />
  <while>
    <condition … />
    <activity2 />
  </while>
  <activity3 />
</sequence>
```

Although the Common Base already allows for inserting activities, it does not consider that a present non-communicating activity can be embedded into a loop. Justification of this rule is given by applying the following three projection-inheritance preserving transformation rules: remove activity2, add the loop, and insert activity2 into the loop.

Rule 2: activity removal from sequence. A non-communicating basic or structured activity can be deleted from a sequence of activities.

Example for Rule 2:

```
<sequence>
  <activity1 />
  <activity2 />
  <activity3 />
</sequence>
```

```
<sequence>
  <activity1 />
  <activity3 />
</sequence>
```

The Common Base allows for inserting an activity. That means, we can apply the transformation illustrated in the preceding example in the other direction as well. However, also removing an activity from a sequence does not change the observable behavior.

Rule 3: activity removal from flow. A non-communicating basic or structured activity can be deleted from a flow.

Example for Rule 3:

```
<flow>
  <activity1 />
  <activity2 />
  <activity3 />
</flow>
```

```
<flow>
  <activity1 />
  <activity3 />
</flow>
```

Like for the previous rule, the Common Base considers only activity insertion, rather than the removal of an non-communicating activity from a flow.

Rule 4: activity reordering. A sequence of solely non-communicating basic or structured activities can be arbitrarily reordered.

Example for Rule 4:

```
<sequence>
  <activity1 />
  <activity2 />
</sequence>
```

```
<sequence>
  <activity2 />
  <activity1 />
</sequence>
```

As we are allowed to remove and to insert non-communicating activities, we can consequently also reorder non-communicating activities that are sequentially ordered. Note that applying these four rules *must not violate data-dependencies* between activities. However, there exist techniques [Loh08, MMG+07, HAM08] to identify those data-dependencies.

Rules for communicating activities

Next, we introduce three additional rules that change the order of communicating activities.

Rule 5: invoke-flow serialization. A flow of one-way invoke activities can be transformed into a sequence.

Example for Rule 5:

```
<flow>
  <invoke operation="a" />
  <invoke operation="b" />
</flow>
```

```
<sequence>
  <invoke operation="b" />
  <invoke operation="a" />
</sequence>
```

If n one-way invoke activities are executed concurrently in a flow, then these activities can be executed in any sequential order without changing the observable behavior. Correctness of Rule 5 is justified by Rule 2 in Section 7.3.

Rule 6: receive-flow serialization. A flow of receive activities can be transformed into a sequence.

Example for Rule 6:

```
<flow>
  <receive operation="a" />
  <receive operation="b" />
</flow>
```

```
<sequence>
  <receive operation="b" />
  <receive operation="a" />
</sequence>
```

Rule 6 is the analogous of Rule 5; that is, n receive activities being executed concurrently in a flow can be executed in any sequential order without changing the observable behavior. Correctness of Rule 6 is justified by Rule 1 in Section 7.3.

Rule 7: invoke and receive. A sequence of first a one-way invoke activity and then a receive activity can be transformed into a flow.

Example for Rule 7:

```
<sequence>
  <invoke operation="a" />
  <receive operation="b" />
</sequence>
```

```
<flow>
  <invoke operation="a" />
  <receive operation="b" />
</flow>
```

Applying Rule 7 allows to increase the amount of concurrency in the BPEL process without affecting the observable behavior. Correctness of Rule 7 is justified by Rule 3 in Section 7.3.

8.3.2. A relaxed equivalence notion for asynchronous bindings

We mentioned already that the BPEL specification does not make any assumption about protocols, bindings, and quality of service attributes of interactions. So far, all presented transformation rules are valid for both synchronous and asynchronous binding. However, if we would assume an asynchronous binding, we can *relax* the behavioral equivalence relationship even further by introducing three additional transformation rules. We therefore generalize the relation of behavioral equivalence to *relaxed behavioral equivalence.*

Definition 8.3.2 (relaxed equivalence of observable behavior).
An executable process P'' is *relaxed behaviorally equivalent* to an abstract process P iff P'' is behaviorally equivalent to P, or there is an executable process P' being behaviorally equivalent to P, and P'' can be derived from P' by applying zero or more of the following three transformation rules:

8. Invoke-sequence reordering
9. Receive-sequence reordering
10. Invoke and receive ⌟

In the following, we introduce the three additional rules.

Rule 8: invoke-sequence reordering. A sequence of one-way invoke activities can be arbitrarily reordered, or it can be transformed into a flow.

Example for Rule 8:

```
<sequence>
  <invoke operation="a" />
  <invoke operation="b" />
</sequence>
```

```
<sequence>
  <invoke operation="b" />
  <invoke operation="a" />
</sequence>
```

In case of asynchronous bindings, it is possible that if a process sends first a message `a` and then a message `b`, then its partner process may receive `b` before `a`. The reason is that messages can overtake each other on a message channel. As a sequence of n sending messages can reach the partner in any order, we can arbitrarily reorder a sequence of one-way invoke activities or even embed these activities into a flow. Correctness of Rule 8 is justified by Rule 2 in Section 7.3.

Rule 9: receive-sequence reordering. A sequence of receive activities can be arbitrarily reordered, or it can be transformed into a flow.

Example for Rule 9:

```
<sequence>                              <sequence>
  <receive operation="a" />      ⇨        <receive operation="b" />
  <receive operation="b" />               <receive operation="a" />
</sequence>                             </sequence>
```

For the same arguments as presented for Rule 8, we can arbitrarily reorder a sequence of receive activities or even embed these activities into a flow by applying Rule 9. Correctness of Rule 9 is justified by Rule 1 in Section 7.3.

Rule 10: invoke and receive. A flow that contains a one-way invoke activity and a receive activity can be transformed into a sequence of *first* the invoke and *then* the receive activity.

Example for Rule 10:

```
<flow>                                  <sequence>
  <invoke operation="a" />       ⇨        <invoke operation="a" />
  <receive operation="b" />               <receive operation="b" />
</flow>                                 </sequence>
```

Rule 10 describes in fact the opposite direction of Rule 7. It is only applicable in case of asynchronous bindings: Assume a process similar to the left hand side of the example. Further assume a mirrored version of this process as a partner. Applying Rule 10 to the process in case of a synchronous binding could lead to a deadlocking situation in the case where the message on `b` sent by the partner arrives before the process has sent the message on `a` to the partner. Correctness of Rule 10 is justified by Rule 3 in Section 7.3.

8.3.3. Disallowed transformation rules

Based on the results presented in this section so far, the notion of equivalent behavior can be significantly extended beyond BPEL's APPOB. However, even for our notion of extended behavioral equivalence, there exist limitations, as shown by the following three transformation rules, which are explicitly disallowed.

Anti-rule 1. A sequence of first a one-way invoke and then a receive activity must not be reordered, or vice versa.

Example for Anti-rule 1:

```
<sequence>                              <sequence>
  <invoke operation="a" />       ⇏        <receive operation="b" />
  <receive operation="b" />      ⇍        <invoke operation="a" />
</sequence>                             </sequence>
```

Beside reordering, also a concurrent execution is disallowed:

```
<process name="example" ...>
  <partnerLinks>
    <partnerLink name="a" ... />
    <partnerLink name="b" ... />
  </partnerLinks>
  <sequence>
    <receive partnerLink="a" operation="a1" ... />
    <if>
      <condition> ... </condition>
      <sequence>
        <empty />
        <receive partnerLink="b" operation="b1" ... />
      </sequence>
      <else>
        <sequence>
          <empty />
          <receive partnerLink="b" operation="b2" ... />
        </sequence>
      </else>
    </if>
  </sequence>
</process>
```

Figure 8.4.: Counterexample justifying the correctness of Anti-rule 3.

Anti-rule 2. A sequence of first a receive activity and then a one-way invoke activity must not be transformed into a flow, or vice versa.

Example for Anti-rule 2:

```
<sequence>                         ⇏    <flow>
  <receive operation="a" />               <receive operation="a" />
  <invoke operation="b" />                <invoke operation="b" />
</sequence>                        ⇍    </flow>
```

Anti-rules 1 and 2 are justified by the anti-patterns which we presented in Section 7.3.

Finally, the addition and usage of new partner links is not permitted. This anti-rule differs from the original APPOB where this addition is explicitly allowed.

Anti-rule 3. New partner links or communicating activities must not be added.

Anti-rule 3 excludes the addition of new partner links and new communicating activities. Although this is permitted in the original APPOB and does not affect the observable behavior from one partner's point of view, it would change the global observable behavior by introducing "unintended" behavior. As an example, consider the process in Figure 8.4.

The original APPOB would allow to add a partner link `b` and the two receive activities (the bold lines in Figure 8.4). Although this addition would not affect the partner communicating via partner link `a`, the addition would introduce unintended behavior: A partner communicating via partner link `b` would have to *guess* how the condition of the if activity was evaluated to decide whether to send a message using operation `b1` or `b2`. Therefore, the process could either deadlock (in case the wrong operation was used) or would complete with a redundant pending message (in case both operations were used). Both cases are certainly not desirable, but not excluded by the APPOB.

8.3.4. Discussion

This chapter presented an application of our theory on service substitutability. We adapted our notion of X-conformance equivalence, for $X = \{\text{weak termination}, \text{strict termination}\}$ to BPEL processes and reformulated the X-conformance-equivalence preserving transformation rules in the setting of BPEL.

The attentive reader will have recognized that some of the X-conformance-equivalence preserving transformation rules of Section 7.3 were not reformulated in the previous subsection. As an example, the simultaneously sending of several messages cannot be modeled in BPEL.

We presented a notion of behavioral equivalence on BPEL processes. This notion can of course be relaxed to a preorder. In this case, we could reformulate all X-conformance-preserving transformation rules of Section 7.4.

As an alternative to the presented transformation rules, we can check relaxed behavioral equivalence of two BPEL processes a posteriori with the tools BPEL2oWFN/Fiona [ALM$^+$09]. However, our theoretical results are limited to X-conformance equivalence for acyclic services, because only in that case the analysis question coincides with deciding X_1-conformance equivalence (cf. Chapter 5).

Checking behavioral equivalence (or conformance) between an abstract and an executable process is not a particular strength of the BPM tools currently available in the marketplace. However, in cases where such a functionality is offered, our approach has beneficial practical implications. The proposed profile makes it much easier to find an executable process that is equivalent to a given abstract one, as the set of behaviorally equivalent processes is substantially larger than the set of executable completions in BPEL (cf. Figure 8.3). Thus, this may potentially save development time when creating BPM solutions.

Finally, it is worthwhile to mention the novel abstract profile for BPEL, which we defined in the previous section, is *not* a WS-BPEL language extension. It should be taken as a new profile—something that is already allowed by the standard today.

Part V.

Wrap-up

9. Related Work

In this chapter, we review related work in the area of service substitution. We start with an overview of substitutability criteria related to our notion of X-conformance in Section 9.1. Subsequently, work related to our substitutability criterion X-preservation is surveyed in Section 9.2, and finally Section 9.3 reviews work on transformation rules.

9.1. Work on conformance

In this section, we review work related to our substitutability criterion X-conformance. To structure the different approaches, we distinguish among work based on Petri nets, on process calculi, and on automata.

9.1.1. Work based on Petri nets

Work of Vogler Vogler presents in [Vog92] a few tens of equivalences to support modular construction of Petri nets. The setting of his work is, however, more general than ours. Besides asynchronously communicating Petri nets, Vogler also studies synchronously communicating Petri nets; that is, Petri nets are composed by transition fusion. In what follows, we relate our results to the results of Vogler for asynchronous communication.

One of the behavioral properties that are considered in [Vog92] is deadlock freedom. This notion is stricter than ours (see Definition 2.6.1), because a livelock that consists only of internal transitions is treated as a deadlock in [Vog92]. In addition, the notion of *invariant reachability* of places is introduced in [Vog92]. A place p of N is invariantly reachable if and only if, for every reachable marking m in N, there is some marking m' with $m \xrightarrow{*} m'$, such that $m'(p) > 0$. For singleton sets $\Omega = \{m_f\}$ of final markings, invariant reachability of all p with $m_f(p) > 0$ is equivalent to our notion of weak termination.

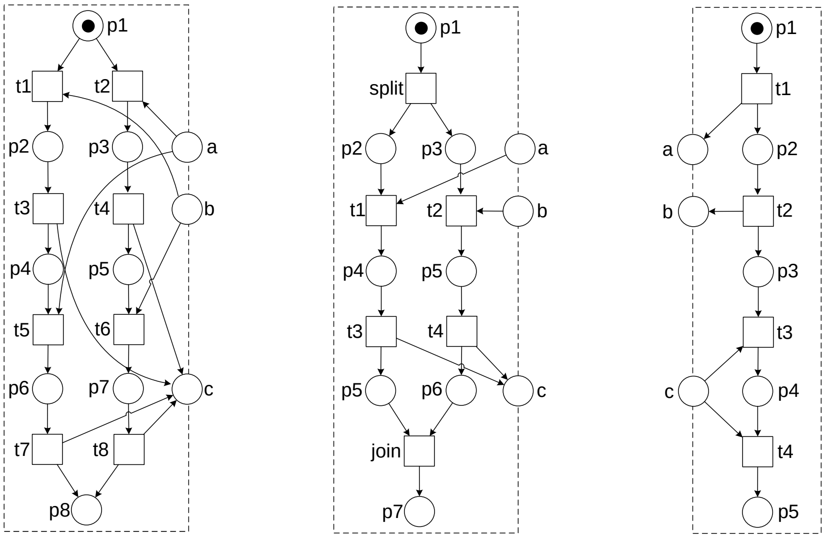
Vogler defines the notion of *IR*-equivalence. Two open nets N and N' are *IR*-equivalent if, for all open nets S, every place $p \in P_S$ is invariantly reachable in $N \oplus S$ if and only if it is invariantly reachable in $N' \oplus S$. As a main difference to X_3-conformance equivalence, *IR*-equivalence is an asymmetric notion; that is, it only focuses on the partner S of N. In contrast, X_3-conformance equivalence requires that both, N and its partner, always reach a final state. *IR*-equivalence coincides with fair testing (called $\mathcal{PF}^{++}$-equivalence in [Vog92]). In Appendix A.1, we proved that fair testing implies weak termination, but the opposite direction does not hold in general. The notion of deadlock freedom, which is considered in [Vog92], is also equivalent to fair testing.

Work of Basten and Van der Aalst For workflow nets (WFNs) [Aal98], the notion of *projection inheritance* [BA01, AB02] is used two relate two WFNs that can be substituted. Projection inheritance is based on branching bisimulation. As a difference to our work, the projection-inheritance approach assumes a synchronous communication model (i. e., Petri nets are composed by transition fusion). In Section 7.2, we reformulated projection inheritance for open nets and proved that X_3-conformance equivalence is more liberal than the notion of projection inheritance; that is, projection inheritance implies X_3-conformance equivalence, and hence it also implies X_1-conformance equivalence.

Work of Martens Martens presents in [Mar05] a refinement notion for workflow modules. Workflow modules — like open nets — are a Petri-net formalism to model (Web) services. However, workflow modules are subject to several syntactic restrictions (similar to workflow nets) and therefore less general than open nets. Martens' refinement notion is equivalent to X_3-conformance. To decide refinement of acyclic workflow modules, Martens introduces a data structure, called *communication graph*, and a simulation relation on these graphs. A communication graph in some sense represents the communication behavior of a service and can be compared to a reduced version of our most permissive X_3-strategy (cf. Definition 4.3.1). Due to the limitations of workflow modules and the loss of information by the reductions compared to our X_1-operating guidelines, the structural simulation relation on communication graphs is only sufficient to decide X_3-conformance of *acyclic* services. In contrast, for acyclic services, we were able to prove a criterion that is necessary and sufficient (in this case X_1-conformance and X_3-conformance coincide; see Lemma 2.6.3). Figure 9.1 shows that the decision procedure in [Mar05] is only sufficient but not necessary.

Work of Bonchi et al. Bonchi et al. present Open Consume-Produce Read nets to model the control flow and the data flow of a service [BBCG08]. For Open Consume-Produce Read nets, *saturated bisimulation* is proposed as equivalence notion. Saturated bisimulation is characterized as weak bisimulation in [BBCG08]. Consequently, saturated bisimulation implies X_3-conformance equivalence, but the opposite does not hold in general. As an example, the two open nets in Figure 3.1 are not weak bisimilar; however, they are X_3-conformance equivalent.

Work of Van der Aalst et al. Van der Aalst et al. use open nets (where the set of final markings is a singleton set) to model *service trees* [AHM$^+$09]. A service tree is a special open-net composition. If each open net is considered as a node of a graph, and the set of shared interface places between two open nets is considered as



(a) N (b) N' (c) S

Figure 9.1.: Martens' refinement notion is only sufficient: According to [Mar05], S is an X_3-strategy of N' and not of N. Thus, N' X_3-conforms to N. However, N and N' are X_3-conformance equivalent, for a communication bound b=2.

an edge between the respective nodes of the graph, then the corresponding graph has a tree structure. Van der Aalst et al. study when an open net N in a service tree can be substituted by an open net N' such that weak termination of the service tree is preserved. In other words, they study X_3-conformance in a slightly restricted setting. Let N_1, N_2, and N_3 be pairwise interface compatible open nets such that N_1 and N_3 have pairwise disjoint interface places. If $\mathit{inner}(N_1 \oplus N_2)$ weakly terminates and a condition γ holds, then $\mathit{inner}(N_1 \oplus N_2 \oplus N_3)$ weakly terminates [AHM+09]. The condition γ requires that the reachable markings of $N_2 \oplus N_3$ simulate the restriction of the reachable markings of $N_2 \oplus N_3$ to the places of N_2. This condition is proved to be equivalent to the requirement that every run of $\mathit{inner}(N_2)$ is the projection of a run of $\mathit{inner}(N_2 \oplus N_3)$ to the transitions of N_2. Condition γ is proved to be sufficient [AHM+09]. Using the results of this thesis, we can only decide substitutability in the setting of service trees for acyclic open nets, as for acyclic open nets X_1-conformance and X_3-conformance coincide. In this case, our decision procedure is necessary and sufficient.

9.1.2. Work based on process calculi

Work of Fournet et al. Fournet et al. [FHRR04] consider CCS processes of asynchronous message passing software components. They present *stuck-free conformance* of such processes. Stuck-freedom formalizes the absence of deadlocks in the system. It can be checked using the model checker Zing [AQR$^+$04]. Stuck-free conformance requires that an implementation simulates its specification and that the failures of the implementation are contained in the failures of the specification. Furthermore, the accepted actions in the implementation are contained in the accepted actions of the specification. The authors show that the CSP stable-failures preorder [Ros98] does not imply stuck-free conformance. Furthermore, stuck-free conformance is strictly larger than the refusal preorder of Phillips [Phi87]. As the stable failures preorder and X_1-conformance coincide (cf. Theorem A.2.2), we conclude that X_1-conformance does not imply stuck-free conformance.

Work of Malik et al. Malik et al. [MSR06] introduce the *conflict preorder*, which is equivalent to our notion of X_3-conformance. They prove that fair testing implies the conflict preorder. In contrast to us, they do not show explicitly the difference between fair testing and the conflict preorder, but formalize the conflict preorder by help of a new failures semantics.

Work of Padovani et al. Laneve and Padovani introduce in [LP07] the *subcontract preorder* for CCS-like processes without τ-actions. The subcontract preorder is equivalent to must testing [NH84]. Must testing is known to be incomparable to fair testing, as it cannot distinguish between a loop and a livelock. As another difference, the subcontract preorder is an asymmetric notion; that is, it is focusing only on the test, rather than on the service being tested. In contrast, our notion of X_3-conformance is a symmetric notion; that is, both the test and the service have to terminate. Must testing is equivalent to stable failures—at least for finitely branching processes without divergences [Nic87]. Hence, it is equivalent to X_1-conformance. As an additional difference, our service models may contain τ-actions.

In [CGP08], the *weak subcontract preorder* is introduced. It is a relaxed version of the subcontract preorder. The idea is to extend the set of compatible partners of a service by the help of *dynamic filters*. A filter specifies a set of actions of the service that may occur. All other actions are blocked. In our setting of asynchronously communicating open nets we would specify which messages an open net may send. A concept like filters we did not consider in this thesis.

Work of Bravetti and Zavattaro Bravetti and Zavattaro [BZ07a, BZ08] model services as CCS-like processes. Their proposed *subcontract preorder* coincides with our notion of X_3-conformance. As Malik et al. [MSR06], they also prove that fair testing implies their subcontract preorder. In Section A.1, we showed under which conditions fair testing and X_3-conformance coincide.

As an additional requirement, Bravetti and Zavattaro introduce the output persistence property. A composition of processes is output persistent if no sending event is enabled in a final state.

Bravetti and Zavattaro study the subcontract preorder for different communication paradigms. They consider synchronous handshake in [BZ07a, BZ08] and asynchronous communication via unbounded message queues in [BZ09]. In addition, a stronger notion is introduced ensuring, whenever a message can be sent, the other service is ready to receive this message. Systems that behave this way are strongly compliant [BZ07b]. Bravetti and Zavattaro show that independent of the chosen communication paradigm, the notion of fair testing implies the corresponding subcontract preorder.

9.1.3. Work based on automata

The following approaches use automata models to decide substitutability. As the main difference, all these approaches use only synchronous communication, whereas open nets model asynchronous message passing.

Work of Benatallah et al. Benatallah et al. consider deterministic automata [BCT06]. They present two notions of compatibility: partial compatibility and full compatibility. Full compatibility coincides with weak termination. In contrast, partial compatibility only ensures that that the composition contains some run to a final state. A similar criterion is relaxed soundness [DA04]. The authors introduce a notion of conformance and of conformance equivalence. Automaton A' conforms to automaton A if and only if A' simulates A'. If A also simulates A', then both automata are conformance equivalent. The simulation relation thereby respects final states.

Work of Sharygina et al. The *ComFoRT* framework [SCCS05] analyzes whether a software component S, implemented in the programming language C, can be substituted by another software component S'. S can be substituted by S' if (i) every behavior possible in S is a behavior of S', and (ii) the new version of the software system satisfies previously established correctness properties. Behavior inclusion is verified by trace comparison of the software components, which does not allow the reordering of messages, for instance. In contrast to [SCCS05], refinement for interface automata [AH01] defined by alternating simulation assumes that the environment remains the same [CSS06].

Work of Cerna et al. In [CVZ07], Cerna et al. present an approach to check component substitutability. As a formal model, component interaction (CI) automata are used. A CI automaton implements a hierarchy concept. Cerna et al. introduce three substitutability notions: conformance equivalence, conformance, and substitutability in case the environment is fixed. To decide conformance equivalence, the authors define an equivalence between two CI automata, which is similar to weak bisimulation. In this thesis, we cover all three notions with X-conformance, with X-conformance equivalence, and in case the environment is known, we could check whether the new service is an X-strategy of the environment.

9.2. Work on preservation

In this section, we review work related to our substitutability criterion X-preservation. In comparison to X-conformance, there are only few related approaches, which are introduced in the following.

Work of Beyer et al. Beyer et al. [BCH05] introduce protocol automata for modeling service behavior. These automata follow a synchronous communication paradigm, whereas we consider asynchronous message passing. For protocol automata, a refinement notion is defined, which is proved to be a precongruence with respect to composition. As behavioral properties, temporal logic properties are considered which are preserved by the refinement. In comparison to our proposed behavioral constraints on visible actions of open nets, this approach generalizes our property-preserving substitution in Section 6.2.3.

Work of Benatallah et al. Benatallah et al. introduce beside X_3-conformance and X_3-conformance equivalence (cf. Section 9.1.3) also two other substitutability criteria [BCT06]. In the first criterion, a service S' substitutes a service S assuming the partner U is known. In this setting, we would check whether S is an X-strategy of U. In the second criterion, S' substitutes S with respect to an interaction role R; that is, the intersection of S and R has to behave as S'. In this case, we would check if $\mathit{Match}(OG_X(R) \otimes OG_X(S)) \subseteq \mathit{Strat}_X(S')$. This can only be decided for $X_1 = \{\text{deadlock freedom}\}$ and $X_2(Y) = \{\text{deadlock freedom, cover}(Y)\}$. However, as mentioned before, in [BCT06] deterministic and synchronously communicating automata are considered.

Work of Pathak et al. Pathak et al. focus on a substitutability criterion that preserves some property of a service S to be substituted [PBH07]. The property is expressed by a μ-calculus formula ϕ. Then, a μ-calculus formula ψ is calculated such that all services S' that satisfy ψ can substitute S. Due to the expressiveness of the μ-calculus in comparison to our proposed behavioral constraints on visible actions of open nets, this approach generalizes our property-preserving substitution in Section 6.2.3, but it assumes a synchronous communication model.

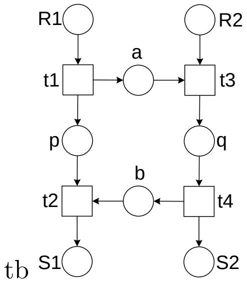


Figure 9.2.: Block structure ensuring that place p is synchronized with place q [AHM+09].

9.3. Work on conformance-preserving transformation rules

Refinement of Petri nets has been addressed by many researchers—for example [Val79, SM83, Ber87]. However, most of the results require restricted Petri-net classes or Petri nets without interfaces. The Murata rules [Mur89] (known for general Petri nets) maintain X_3-conformance if we consider every input place as a place with some additional incoming arcs, and every output place as a place with some additional outgoing arcs.

Refinement in a more general setting is considered by Vogler [Vog92], for instance. There it is shown how a place or a transition of a Petri net can be refined by a subnet. The basic idea is that replacing a place by a subnet is a special case of substitutability in the case of transition fusion, as the transitions that surround a place p serve as an interface for p. In contrast, replacing a transition is a special case of substitutability in the case of place fusion. For both cases, Vogler presents several equivalence notions. However, this work goes beyond the idea of our transformation rules.

Our work can be seen as a generalization of the projection inheritance-preserving transformation rules of Basten and Van der Aalst [BA01, AB02]. In this thesis, we have proved that these rules also preserve X_3-conformance. Transformation rules that reorder interface transitions, as the ones we introduced in Section 7.3, have been to the best of our knowledge not considered so far.

Recently, another refinement technique has been proposed in [AHM+09]. Van der Aalst et al. consider open nets and identify a pattern that allows place refinement. This pattern consists of any two places p and q of an open net N such that p is synchronized with q. Here, synchronization means that place q is only marked if p is marked, and that p can only be unmarked, after q became unmarked. Figure 9.2 illustrates a pattern that satisfies the synchronization property. If p is synchronized with q, then these places can be refined by open nets N_p and N_q, respectively, if and only if N_p and N_q are interface compatible, and the composition $N_p \oplus N_q$ is a closed net that weakly terminates. This refinement guarantees that the refined open net N' X_3-conforms to N [AHM$^+$09].

10. Conclusions

In this final chapter, we conclude this thesis. Section 10.1 summarizes the main contributions of our approach on behavioral service substitutability. Open problems of this approach are discussed in Section 10.2. Finally, Section 10.3 sketches ideas how this research can be continued.

10.1. Contribution of this thesis

The Service-Oriented Computing (SOC) paradigm aims at building complex systems by composing them from less complex systems, called services. Such a (complex) system is a distributed application often involving several cooperating enterprises. As a system usually is subject to change, individual services will be substituted by other services during the system's life-cycle. Substituting one service by another one should not affect the correctness of the overall system. Verification of correctness is challenging, as the overall system is usually not known to any of the involved enterprises.

In this thesis, we restricted ourselves to the *changes of the service behavior*. This restriction implies that we abstract from resources and consider only data/message types and not their content. As a formal service model, we used open nets, a subclass of Petri nets tailored towards the modeling of services. Suitability of our model has been demonstrated by open-nets semantics for various languages, such as BPMN, BPEL, and BPEL4Chor.

The contribution of this thesis is threefold. First, we defined substitutability criteria for services that are in particular suitable in the context of multiparty contracts and service improvement. Second, for each substitutability criterion, we developed an algorithm to decide whether a service can substitute another service according to the respective substitutability criterion. And third, we presented refinement rules for constructing a substitutable service from a given service. In what follows, we review each contribution in more detail.

10.1.1. Substitutability criteria

Suppose a system modeled as the composition $N \oplus S$ of two open nets N and S. Clearly, substituting N by another open net N' should preserve some behavioral properties of $N \oplus S$ in the new composition $N' \oplus S$. In this thesis, we put the main focus of attention on the five properties

- deadlock freedom; that is, the open net does not get stuck,

- weak termination; that is, the open net can always reach a final state and hence is free of deadlocks and livelocks,
- cover(Y); that is, a given set Y of places and transitions of N can potentially be marked/enabled in the composition,
- quasi-liveness; that is, every transition of the composition can potentially be enabled, and
- strict termination; that is, being in a final state, no transition of an open net can be enabled.

For any subset X of these properties, S is an X-strategy of N if the composition $N \oplus S$ is an open net that satisfies X. The set $\mathit{Strat}_X(N)$ defines all X-strategies of N. We argued that there are only 9 reasonable sets of open-net properties listed in Table 2.1.

Based on these properties, we defined in Chapter 3 two substitutability criteria, *X-conformance* and *X-preservation.* N' can substitute N under X-conformance if every X-strategy of N is an X-strategy of N'. Hence, X-conformance ensures that substituting N by N' does not affect S. Therefore, X-conformance is well-suited in the context of multiparty contracts, where every enterprise has to locally implement the specification of its service while preserving correctness of the overall system. X-preservation is less restrictive than X-conformance. N' substitutes N under X-preservation if a certain subset $\mathcal{S} \subseteq \mathit{Strat}_X(N)$ is preserved. Hence, the notion of X-preservation allows for removing functionality of a service and is thus well-suited for service improvement.

X-conformance is a preorder and even a precongruence with respect to open-net composition. We compared {weak termination}-conformance with fair testing. Some researchers proved already that fair testing implies {weak termination}-conformance. In Appendix A.1, we extended these works and showed under which conditions {weak termination}-conformance implies fair testing. Finally, in Appendix A.2 we proved that {deadlock freedom}-conformance is equivalent to the stable failures preorder.

10.1.2. Deciding substitutability

To decide whether an open net N' can substitute an open net N under X-conformance or under X-preservation, we have to compare the two (in general) infinite sets $\mathit{Strat}_X(N')$ and $\mathit{Strat}_X(N)$. However, these infinite sets can be represented in a finite manner. In case of deadlock freedom, a finite representation, called operating guideline, already existed. In this thesis, we *extended* the notion of an operating guideline to represent the set $\mathit{Strat}_X(N)$ for any set X of open-net properties. It turned out that such a finite representation depends on the chosen termination criterion deadlock freedom or weak termination and whether quasi-liveness or covering of open-net nodes is required or not. This yields the following 6 sets of open-net properties:

- $X_1 = \{\text{deadlock freedom}\}$;
- $X_2 = X_1 \cup \{\text{quasi-liveness}\}$;
- $X_2(Y) = X_1 \cup \{\text{cover}(Y)\}$;
- $X_3 = \{\text{weak termination}\}$;
- $X_4 = X_3 \cup \{\text{quasi-liveness}\}$;
- $X_4(Y) = X_3 \cup \{\text{cover}(Y)\}$.

Strict termination does not affect the finite representation, but is only reasonable in combination with weak termination. Hence, it can be added to the sets X_3, X_4, and $X_4(Y)$.

As a consequence, we ended up with 6 finite X-strategy representations, called *X-operating guidelines*, for $X \in \{X_1, X_2, X_2(Y), X_3, X_4, X_4(Y)\}$. Table 10.1 summarizes these results. For each X-operating guideline, we developed a *matching procedure* to efficiently check whether an open net S is an X-strategy of N—that is, to check whether S is contained in the X-operating guideline of N; see the third column in Table 10.1.

Table 10.1.: Results overview: "✓" denotes that a solution already existed; "+" denotes that a solution has been provided in this thesis; "−" denotes that there is no solution so far.

X-strategies	finite representation	matching	conformance	product
X_1	✓	✓	+	✓
X_2	+	+	−	−
$X_2(Y)$	+	+	+	+
X_3	+	+	−	+
X_4	+	+	−	−
$X_4(Y)$	+	+	−	+

With the help of these finite representations of all X-strategies, we presented algorithms to decide the two substitutability criteria. The decision algorithms reduce the comparison of two infinite sets of X-strategies to a finite check. For X-conformance, we developed in Chapter 5 a decision procedure for X_1-operating guidelines and for $X_2(Y)$-operating guidelines; see the fourth column in Table 10.1. Deciding X-preservation (cf. Chapter 6) depends on the set $\mathcal{S}$ of X-strategies to be preserved. If $\mathcal{S}$ is a finite set, X-preservation reduces to match each open net $S \in \mathcal{S}$ with the X-operating guideline of N'. This can be decided for every X. If $\mathcal{S}$ is an infinite set that is represented by an X-operating guideline, then deciding X-preservation reduces to an inclusion check of the sets $\mathcal{S}$ and $\mathit{Strat}_X(N')$ based on their X-operating guidelines. This can so far only be decided for X_1 and X_2.

In addition, we presented results to construct a finite representation of $\mathcal{S}$ from the X-operating guideline of N. To this end, we defined the *product* for each two X-operating guidelines (except for X_2 and X_4) and proved this product to be equivalent to the intersection of the respective sets of X-strategies; see the fifth column in Table 10.1. We also showed that X_1-operating guidelines can be used to describe those open nets that follow some behavioral constraint. That way, we can restrict the set $\mathit{Strat}_X(N)$ to those X-strategies that satisfy the behavioral constraint by calculating the product of the X-operating guidelines of N and the X_1-operating guideline of the behavioral constraint.

Based on the theoretical results presented in this thesis, the construction of an $X_2(Y)$-operating guideline and an X_3-operating guideline for an open net have been prototypically implemented in the service analysis tool Fiona. Experimental results with a number of real-life service models in this thesis demonstrated that these finite representations can be calculated efficiently in spite of the high worst-case complexity of the construction algorithms. We also demonstrated that the inclusion check and the product can be computed efficiently; however, these operations are only for X_1-operating guidelines implemented so far.

10.1.3. Constructing substitutable services

This thesis also contributed to the construction of open nets N' that can substitute an open net N according to a substitutability criterion. For X_3-conformance, we presented a number of *transformation rules* to derive an open net N' from an open net N by stepwise refinement (cf. Chapter 7). Thereby, each refinement step preserves X_3-conformance. Some of these rules preserve X_3-conformance in both directions (i. e., X_3-conformance equivalence), and some preserve X_3-conformance only in one direction.

Beside these theoretical results, we also presented a practical contribution in this thesis. In Chapter 8, we studied the equivalence relation between a service specification and a service implementation for services described in the language BPEL. This equivalence is defined only on the syntax of the services. We showed that this equivalence coincides with $X_3 \cup \{\text{strict termination}\}$-conformance equivalence. Based on this result, we adapted our transformation rules to BPEL. Given a service specification S, every service that can be derived from S, by applying any number of transformation rules, is equivalent to S. That way, more services are equivalent to S without the loss of general applicability. This result justifies the importance of our work, on the one hand and its applicability in practice, on the other hand.

10.2. Open problems

Within the context of this thesis not all problems related to the topic of service substitution have been solved. In this section, we discuss unsolved problems that could make the presented solutions more complete.

Conformance The first open problem is an algorithm to decide X-conformance of two open nets in case of X_2 (i. e., deadlock freedom and quasi-liveness), X_3 (i. e., weak termination), X_4 (i. e., weak termination and quasi-liveness), and $X_4(Y)$ (i. e., weak termination and cover(Y)). To decide whether N' X-conforms to N, one has to check $\mathit{Strat}_X(N) \subseteq \mathit{Strat}_X(N')$. In Chapter 5, we developed an algorithm to decide inclusion of two sets of X-strategies on their respective X-operating guidelines, for $X \in \{X_1, X_2(Y)\}$. It would be important to have such algorithms also for the other four finite representations, because X-conformance is of high practical relevance.

Responsiveness The second open problem is the discrepancy between our notion of deadlock freedom and the notion of responsiveness implemented in the tool Fiona. Deadlock freedom is a too less restrictive correctness criterion, as it allows X_1-strategies that may run into an internal livelock. In particular, an open net S that puts the maximal number of messages (according to the message bound) into each input channel of an open net N and then executes a τ-loop is an X_1-strategy of N. Clearly, S contradicts the intention of an X_1-strategy of N. The main drawback of such open nets S is that the X_1-operating guideline becomes great in size. To exclude such services S, the notion of responsiveness [Wol09] has been implemented in Fiona. It would be important to study whether the results of this thesis with respect to deadlock freedom (and quasi-liveness) can be lifted to responsiveness (and quasi-liveness). In particular, the notion of responsiveness must be a precongruence with respect to open-net composition; otherwise, it cannot be used in the setting of multiparty contracts.

Tooling and validation The third open problem is related to the implementation of the algorithms developed in this thesis. As mentioned in the previous section, not all results of this thesis have been implemented in Fiona. For example, the notion of an $X_4(Y)$-operating guideline and the algorithm for $X_2(Y)$-conformance has to be implemented, and the product operator has to be lifted to $X_2(Y)$-, X_3-, and $X_4(Y)$-operating guidelines.

Another important issue is the validation of our algorithms on industrial service models. Experiences from the validation provide useful insights for improving the developed methods. As our proposed algorithms have a high worst-case complexity, it is particularly important to investigate effective data structures to make the implementation more efficient.

10.3. Further research

This section sketches some ways how the results presented in this thesis could be extended. These ideas go beyond this dissertation, but are still part of the grand goal to make behavioral service substitution in SOC become reality.

Additional transformation rules In Chapter 7, we presented several X_3-conformance-preserving transformation rules. These rules are not complete. Although we doubt that it is possible to come up with a complete set of transformation rules, it seems to be possible to find and document additional transformation rules. One obvious approach is to restrict the context of a pattern.

Strong termination In this thesis, we focused on deadlock freedom and weak termination as termination criteria for services. It would be interesting to strengthen the notion of weak termination (i. e., *AG EF final*) to strong termination (i. e., *AG AF final*). Weak termination requires a *fairness* principle, and it does not exclude infinite runs of an open net. In contrast, strong termination excludes infinite runs. To this end, we have to add explicit fairness constraints to open-net transitions. Strong termination has not been studied so far.

Substitutability in the decentralized setting In this thesis, we focused on single-port services [Wol09]; that is, an X-strategy of an open net N is always treated like a single service. In case N is connected to more than one open net, this assumes that all these open nets may interact arbitrarily with each other. However, such an assumption is sometimes too relaxed. Consider an open net N that interacts with *two* open nets N_1 and N_2. Assume that N sends a request to either N_1 or N_2 and concurrently expects an acknowledgement from the respective open net. There is an X-strategy S of N such that S receives the request, which N has sent to N_1, and acknowledges on behalf of N_2. This is, in fact, a valid X-strategy of N, but practically impossible if N_1 and N_2 do not interact with each other. In [Wol09], more restrictive settings of X-controllability are studied, where X-strategies of N do not interact with each other. To widen the scope of this thesis, one could study the substitutability criteria of this thesis also in these settings.

Extending constraint-preserving substitutability In Section 6.2.3, we introduced the notion of a constraint automaton to describe all open nets that follow some behavioral constraint. To make this approach more applicable, one could try to specify more complex behavioral constraints with constraint automata. We already mentioned in Section 6.2.3 the idea of defining an algebra on sets of X-strategies as proposed in [KW09]. In addition, one could generalize behavioral constraints to arbitrary temporal logic properties. A temporal logic property ϕ can be expressed by an automaton. For instance, an LTL formula can be expressed by a Büchi automaton and a CTL* formula by an alternating parity automaton [KVW00]. The product of such an automaton and a finite representation of $Strat_X(N)$ for an open net N represents all $S \in Strat_X(N)$ such that $N \oplus S$ satisfies ϕ (for a suitable notion of product). This approach is strongly related to *robust model checking* [KV06]. The robust model checking problem can be defined in our terminology as follows: Given an open net N and a temporal logic property ϕ, then N robustly satisfies ϕ if and only if, for all X_1-strategies S of N, the composition $N \oplus S$ satisfies ϕ. This problem has been solved for CTL* formulae and synchronously communicating Moore automata in [KV06].

Representing all conformant services Usually there is more than one open net N' that X-conforms to an open net N. Hence, it seems to be interesting to investigate how a finite representation of all open nets N' that are X-conformant and X-conformance equivalent to an open net N, respectively, can be calculated. Each open net N' can be seen as a template for a substitutable service that can be refined using our proposed X-conformance-preserving transformation rules. This representation is not only of theoretical interest. It would also widen our results with respect to an equivalence notion for services specified in BPEL in Chapter 8.

One could also think of generalizing this approach by investigating a finite representation of all open nets N'' that can be made X-conformant and X-conformance equivalent, respectively—for example, by using the concept of a message filter [CGP08] or by the help of an adapter [GMW08, AMSW09].

Instance migration In this thesis, we studied static service substitution (i. e., static business protocol evolution); that is, we assumed that there are no running instances of a service S to be substituted. In practice, however, also long-running services have to be substituted. Hence, in this setting, running instances have to be taken into account. Usually, each running instance of the old service S has to be migrated to an instance of the new service S'. This procedure is known as instance migration (or dynamic business protocol evolution). For workflows, the problem of instance migration has been solved [AB02, RRD04, RRMD09]. Clearly, instance migration builds on static service substitutability; that is, S' must be substitutable for S. Thus, the results of this thesis can be seen as a basis for studying instance migration. However, it is necessary to adapt our substitutability criteria to the setting of instance migration.

A. Placing Conformance in the Linear Time – Branching Time Spectrum

In this appendix, we study how the refinement relation X-conformance, introduced in Section 3.1, is placed in the linear time – branching time spectrum [Gla93, Gla01]. We consider X_3-conformance and X_1-conformance.

In Section A.1, we relate the fair testing preorder [RV07] and X_3-conformance. We show that fair testing implies X_3-conformance and prove under which conditions X_3-conformance implies fair testing. In Section A.2, we prove that X_1-conformance and the stable failures preorder [Hoa85] coincide.

A.1. Relationship between conformance and fair testing

In this section, we study the relation between X_3-conformance (recall that $X_3 = \{\text{weak termination}\}$) and the known preorders from the linear time – branching time spectrum [Gla93, Gla01]. In Figure A.1, some of these preorders and the relations between them are depicted, featuring B (bisimulation), RS (ready simulation), PF (possible futures), FT (fair testing [BRV95, NC95, RV07]), MT (must testing [NH84]), CT (completed trace), and T (trace) preorders. An arrow between two preorders denotes the inclusion relation; for example, B implies (is finer than) PF; if an arrow is absent (in the transitive closure), then the inclusion does not hold.

As noticed by [MSR06, BZ07a], fair testing implies X_3-conformance (called conflict preorder in [MSR06], and subcontract preorder in [BZ07a]), but X_3-conformance does not imply fair testing. In [BZ07a], a counterexample is given,

$$\begin{tikzcd}
 & \mathsf{RS} \arrow[r] & \mathsf{MT} \arrow[r] & \mathsf{CT} \arrow[r] & \mathsf{T} \\
\mathsf{B} \arrow[ur] \arrow[r] & \mathsf{PF} \arrow[r] & \mathsf{FT} \arrow[ur] & &
\end{tikzcd}$$

Figure A.1.: Some known preorders from the linear time – branching time spectrum.

whereas in [MSR06] it is also shown that X_3-conformance implies a new variant of failures preorder. This last result, however, gives no conditions under which X_3-conformance coincides with a single known preorder.

In this section, we show that fair testing is the coarsest preorder known to us that implies X_3-conformance. Moreover, we take the extra step to analyze under which conditions X_3-conformance would imply fair testing. In this way, we identify the real differences between X_3-conformance and fair testing.

In this section, we define a novel variant of fair testing that turns out to be equivalent to X_3-conformance. Moreover, we show how restrictions on the tests in this variant can be transformed into restrictions on the services being tested. Finally, we discuss the practicality of these restrictions, in particular within the realm of services that communicate asynchronously. Nevertheless, the main theory is developed in terms of synchronous communication using LTSs.

A.1.1. Preliminaries

We study the relationship of the two preorders on the level of LTSs. In this section, we define the concepts that we will use. We choose LTSs as a service model rather than open nets or service automata, because in the proof we will use that a service is represented as a tree. As another reason, we consider synchronous communication in the following. To ease the formalization, we use an action-based description of an LTS, called *process*.

Processes

Let *Act* be a set of actions containing the special actions: the internal action τ, the termination action $\checkmark$, and the success action $\circledast$. A process is represented as a nonempty (but possibly infinite) *tree-shaped* LTS; for every reachable state, there is exactly one path from the initial state. A *branch* of a tree is an action in the tree followed by the remainder of the tree.

We build example processes in ACP-style [BW90] with the action prefix operators "$a\,\cdot$", for each action $a \in Act$, and infix choice operator "+" (which is commutative and associative). The "deadlock" process δ represents a state without outgoing edges, and for each action $a \in Act$, we abbreviate the process $a \cdot \delta$ by a. The merge operator $\|_H$ on two processes denotes composition by synchronizing the actions from the set H and interleaving the other actions; the set H should contain $\checkmark$ and should not contain $\circledast$ and τ. We also use the renaming operator ρ_f, where f is a function on *Act* such that $f(\tau) = \tau$. The operational rules are as follows:

$$\frac{p \xrightarrow{b} p',\ b \in Act\backslash H}{p \|_H q \xrightarrow{b} p' \|_H q} \quad , \quad \frac{q \xrightarrow{b} q',\ b \in Act\backslash H}{p \|_H q \xrightarrow{b} p \|_H q'} \quad ,$$

$$\frac{p \xrightarrow{a} p',\ q \xrightarrow{a} q',\ a \in H}{p \parallel_H q \xrightarrow{a} p' \parallel_H q'}\ ,\quad \frac{p \xrightarrow{a} p'}{\rho_f(p) \xrightarrow{f(a)} \rho_f(p')}\quad .$$

For conciseness, in the merge operator $\|_H$ we often leave the set H implicit. Given the operational rules and the relation $\Rightarrow$ as defined in Section 2.1, we can use $p\|q \overset{w}{\Longrightarrow} p'\|q'$ to denote that there exist u and v such that $p \overset{u}{\Longrightarrow} p'$, $q \overset{v}{\Longrightarrow} q'$, and w can be obtained from u and v by synchronizing the actions from H and interleaving the other actions.

Definition A.1.1 (visited state).
Let x and y be processes. Process x can *visit* a state r of process y iff there exists a state q of process x such that $x\|y \Rightarrow q\|r$. ⌟

We introduce the abbreviation $p \rightsquigarrow a$, for any process p and any action a, to specify a class of properties related to the testable properties from [GV06].

Definition A.1.2 (leadsto).
Let p be a process, and let a be an action. We use $p \rightsquigarrow a$ (i. e., p leadsto a) to denote that in every execution of process p, as long as action a has not occurred, action a can occur in the future, i. e.,

$$(\forall w, q : w \in (Act\backslash\{a\})^* :\ p \overset{w}{\Longrightarrow} q \quad \Rightarrow \quad (\exists v : v \in Act^* :\ q \overset{v\ a}{\Longrightarrow}))\quad .$$ ⌟

X_3-conformance and fair testing

The X_3-conformance preorder has been introduced in Section 3.1. As already mentioned, we consider the weak-termination property in the composition of a process with its X_3-strategy; that is, always a final state is reachable. We use action $\checkmark$ to mark the *final states*; it is a synchronized action to deal with the final states in a composed process. Thus, an X_3-strategy is formalized as follows.

Definition A.1.3 (X_3-strategy).
Any process z is an X_3-strategy of any process x iff the composition of x and z can always reach a final state, i. e., $x\|z \rightsquigarrow \checkmark$. ⌟

Having lifted the notion of an X_3-strategy to processes, the semantics of X_3-controllability and X_3-conformance is well-defined by Definitions 2.6.8 and 3.1.1, respectively.

In the theory of testing [Gla01, Bru04], behavioral equivalence of two processes x and y is determined by observing the behavior of x and of y when being composed with another process t, called *test*. Of particular interest are those tests that terminate successfully when being composed with x and y. Thereby, successful termination is defined by the corresponding testing preorder and testing equivalence, respectively.

To formalize the fair testing preorder [BRV95, NC95, RV07], we first define the notion of a successful test. We use the action $\circledast$ to model success of a test; it is an *unsynchronized* action, as it should not occur in the process being tested.

Definition A.1.4 (successful test).
Let x be a process that does not contain the action $\circledast$. A process t is a *successful test* on x iff the composition of x and t can always reach a state where $\circledast$ can be performed, i. e., $x\|t \rightsquigarrow \circledast$. ⌟

Definition A.1.5 (fair testing).
Let x and y be two processes that do not contain the action $\circledast$. Processes x and y are related in the *fair testing preorder*, denoted by $x \sqsubseteq_{\mathit{ft}} y$, iff each successful test on y is a successful test on x. ⌟

Fair testing is known to imply the (completed) trace preorder. The other way around, processes $x = a \cdot \checkmark$ and $y = a + a \cdot \checkmark$ are trace equivalent, but in terms of fair testing we only have $x \sqsubseteq_{\mathit{ft}} y$. For example, $t = a \cdot \checkmark \cdot \circledast$ is a successful on x but not on y. This is also depicted in Figure A.1.

Being an X_3-strategy is a symmetric notion, but being a successful test is not: Given the restrictions on $\circledast$, if t is a successful test on x, then x is not a successful test on t. Although for some processes there exist no X_3-strategies, for every process there exists a successful test. Moreover, some tests are successful on every process. The simplest example of such a trivial test is just $\circledast$.

Definition A.1.6 (trivial test).
A *trivial test* is a successful test on every process that does not contain the action $\circledast$. ⌟

A.1.2. X_3-conformance versus fair testing

In this section, we prove that fair testing implies X_3-conformance, and we show that, in general, X_3-conformance does not imply fair testing.

Fair testing implies X_3-conformance

To position X_3-conformance in the spectrum from Figure A.1, consider the processes $x = a^*a \cdot \checkmark$ and $y = a^*a \cdot \checkmark + a^*\delta$ (the binary Kleene star a^*p, for any action a and process p, is the least fix-point Y of the equation $Y = a \cdot Y + p$). X_3-conformance distinguishes between x and y, as x has an X_3-strategy—for example, $a^*\checkmark$—whereas y is X_3-uncontrollable. On the other hand, ready simulation does not distinguish between x and y; see [GV06]. Thus, we conclude that X_3-conformance is not implied by any of the preorders depicted at the top of Figure A.1.

In [MSR06, BZ07a], it is shown that fair testing implies X_3-conformance. For completeness reasons, and for later use, we also provide a proof for it.

Theorem A.1.7 (fair testing implies X_3-conformance).
For any two processes x and y that do not contain the action $\circledast$ holds:

$$x \sqsubseteq_{\mathit{ft}} y \;\Rightarrow\; x \sqsubseteq_{\mathit{conf},X_3} y \quad .$$ ⌟

Proof.
Let x and y be processes that do not contain the action $\circledast$. Let $x \sqsubseteq_{\mathit{ft}} y$; that is, each successful test on y is a successful test on x. To prove $x \sqsubseteq_{\mathit{conf},X_3} y$, let z be an X_3-strategy of y. What remains to be proved is that z is an X_3-strategy of x.

As $\circledast$ is an unsynchronized action, we can reduce the case that X_3-strategy z contains $\circledast$ to an X_3-strategy that does not contain $\circledast$ (by renaming $\circledast$ to τ). In what follows, we consider the case that z does not contain the action $\circledast$.

We can complete the proof if there exists a test t such that for every process p (like x and y) that does not contain $\circledast$: $p \| z \ \rightsquigarrow\ \checkmark \ \Leftrightarrow\ p \| t \ \rightsquigarrow\ \circledast$. Such a process t can be constructed from the given X_3-strategy z (see also [MSR06, BZ07a], which contain the same observation) by replacing every occurrence of the action $\checkmark$ by the term $\checkmark \cdot \circledast$. The proof of $\Rightarrow$ uses that $\circledast$ is an unsynchronized action, and the proof of $\Leftarrow$ uses that p and z do not contain the action $\circledast$. $\square$

As an example, the processes $x = a\cdot\checkmark + b\cdot\checkmark$ and $y = \tau\cdot a\cdot\checkmark + b\cdot\checkmark$ are related by $x \sqsubseteq_{\mathit{ft}} y$ and by $x \sqsubseteq_{\mathit{conf},X_3} y$. In the proof, the example X_3-strategy $z = a\cdot\checkmark$ is replaced by the test $t = a\cdot\checkmark\cdot\circledast$.

Restricted testing

In the proof of Theorem A.1.7, we have used only a limited number of successful $\sqsubseteq_{\mathit{ft}}$-tests; that is, tests where each $\checkmark$ is immediately followed by a trivial test. To make this explicit, we first define a restricted kind of fair testing.

Definition A.1.8 (restricted test).
A *restricted test* is a test in which each occurrence of action $\checkmark$ is immediately followed by a trivial test. ⌟

Definition A.1.9 (restricted testing).
Let x and y be two processes that do not contain the action $\circledast$. Processes x and y are related in the *restricted testing preorder*, denoted by $x \sqsubseteq_{\mathit{rt}} y$, iff each restricted test that is a successful test on y is a successful test on x. ⌟

From the definition of restricted testing, we can immediately conclude its relation to fair testing.

Corollary A.1.10 (fair testing implies restricted testing).
For any two processes x and y that do not contain the action $\circledast$ holds:

$$x \sqsubseteq_{\mathit{ft}} y \ \Rightarrow\ x \sqsubseteq_{\mathit{rt}} y \quad .$$ ⌟

Using these definitions, we can strengthen Theorem A.1.7 based on its previously given proof.

Theorem A.1.11 (restricted testing implies X_3-conformance).
For any two processes x and y that do not contain the action $\circledast$ holds:

$$x \sqsubseteq_{\mathit{rt}} y \ \Rightarrow\ x \sqsubseteq_{\mathit{conf},X_3} y \quad .$$ ⌟

$$\begin{tikzcd} & \mathsf{CT} \arrow[r] & \mathsf{T} \\ \mathsf{FT} \arrow[ur] \arrow[r] & \mathbf{RT} \arrow[r] & \mathbf{Conf} \end{tikzcd}$$

Figure A.2.: Preorders related to X_3-conformance.

Counterexamples

X_3-conformance does not imply restricted testing, as the processes $x = a$ and $y = \delta$ indicate. As x and y are X_3-uncontrollable, $x \sqsubseteq_{conf,X_3} y$ holds. However, $t = a + \circledast$ is a restricted test that is a successful test on y, but it is not a successful test on x. A similar observation in terms of fair testing has been made in [MSR06, BZ07a]. This example shows that X_3-conformance gives no information on X_3-uncontrollable processes, whereas fair testing does.

Restricted testing is unrelated to the (completed) trace preorder. The processes $x = a \cdot \checkmark$ and $y = a \cdot \checkmark \cdot b$ are restricted testing equivalent, but not (completed) trace equivalent. Similarly, the processes $x = a$ and $y = a + a \cdot \checkmark$ are restricted testing equivalent (in both cases, after an a, every successful test is always able to reach $\circledast$ as long as no synchronization occurs; moreover, after any synchronization on $\checkmark$ the test must be trivial), but not (completed) trace equivalent. This shows that restricted testing (and hence X_3-conformance) does not imply the (completed) trace preorder, nor any finer preorder like fair testing. Using our earlier observation that ready simulation does not imply X_3-conformance, we can conclude that the (completed) trace preorder does not imply restricted testing.

These results indicate that fair testing is the coarsest preorder known to us that implies X_3-conformance. In Figure A.2, our preorders RT (restricted testing) and Conf (X_3-conformance) are shown in relation to those in Figure A.1. In what follows, we study the differences between restricted testing and X_3-conformance.

A.1.3. A covering kind of fair testing

In this section, we combine the observations from the previous section and define a kind of fair testing that is equivalent to X_3-conformance. As X_3-conformance gives no information on states that cannot be visited by any X_3-strategy, the idea is to restrict the tests such that they can only visit states that can be visited by X_3-strategies.

Covering tests

For every process, the set of states can be partitioned into the states that can be visited using any X_3-strategy and the states that cannot. The states that can be visited using an X_3-strategy are called coverable. In Definition 2.6.4, we defined when a place or a transition of an open net is covered.

Definition A.1.12 (coverable state).
A state q of a process x is *coverable* iff there exists an X_3-strategy z of x that can visit state q. ⌟

Definition A.1.13 (covering process).
A process y is a *covering* process for a process x iff process y can only visit coverable states of process x. ⌟

As an example, the process $x = a \cdot (b + b \cdot \checkmark + c \cdot \checkmark)$ has three uncoverable states: after the first b, after the second b, and after the first $\checkmark$. In particular the state after the second b is uncoverable, as no X_3-strategy can use this b due to the first b. As illustrated by the state after the first $\checkmark$, whenever a process is in an uncoverable state, it cannot enter a coverable state anymore.

Every X_3-strategy of a process x is a covering process for x. For every X_3-uncontrollable process, there exist no covering processes, because uncoverable processes have no coverable states, and every process has at least one state. In the X_3-conformance preorder, the process and its X_3-strategy should always be able to reach a final state before reaching an uncoverable state. A similar phenomenon occurs in the safe-must preorder [BNP99], where a process and its observer must reach a success state before reaching a catastrophic (i. e., diverging) state.

Using these definitions, we can define a notion of covering restricted testing such that the tests may only visit coverable states of the process being tested.

Definition A.1.14 (covering restricted testing).
Let x and y be two processes that do not contain the action $\circledast$. Processes x and y are related in the *covering restricted testing preorder*, denoted by $x \sqsubseteq_{\mathit{crt}} y$, iff each restricted test that is a successful covering test on y is a successful covering test on x. ⌟

This preorder is a variant of fair testing, whereas in [MSR06] a variant of failures preorder is used. Although we restrict the considered set of tests, in [MSR06] similar sets are extended. Moreover, in [MSR06] it is not proved that X_3-conformance coincides with their variant of failures preorder, whereas in Corollary A.1.20 we will prove that X_3-conformance coincides with covering restricted testing.

Covering restricted testing implies X_3-conformance

As mentioned before, the successful $\sqsubseteq_{\mathit{ft}}$-tests that we have used in the proof of Theorem A.1.7 are restricted tests. Moreover, the used tests turn out to be covering tests, as every X_3-strategy is a covering process. To make this explicit, we again strengthen Theorem A.1.7 based on its previously given proof.

Theorem A.1.15 (covering restricted testing implies X_3-conformance).
For any two processes x and y that do not contain the action $\circledast$ holds:

$$x \sqsubseteq_{\mathit{crt}} y \;\Rightarrow\; x \sqsubseteq_{\mathit{conf},X_3} y \quad .$$ ⌟

Expansion

Before proving that X_3-conformance implies covering restricted testing, we first develop some theory to relate X_3-strategies and covering tests. We introduce the notion of an expansion of a test, which corresponds to adding certain branches after occurrences of action $\circledast$.

Definition A.1.16 (expansion).
An *expansion* of a test t is a process that can be obtained from t by adding branches to states that occur after any action $\circledast$. ⌟

A covering test on process x cannot visit any uncoverable state of x. So, a successful covering test on x must be expandable into an X_3-strategy of x. We prove the following two useful lemmata relating X_3-strategies and covering tests.

Lemma A.1.17 (successful covering test expands into X_3-strategy).
Let x be a process that does not contain the action $\circledast$. Every successful covering test on x is expandable into an X_3-strategy of x. ⌟

Proof.
Let x be a process that does not contain the action $\circledast$. Let t be a successful covering test on x. What remains to be proved is that there exists an X_3-strategy z of x that is an expansion of t.

As t is a successful covering test on x, in the composition $x\|t$ always some state $q\|r$ is reachable, in which q is a covering state of x and r is a state of t, such that $\circledast$ has already occurred at least once. As q is a covering state of x, there exists an X_3-strategy z' of x such that in the composition $x\|z'$ a state $q\|r'$ is reachable, in which r' denotes a state of z'. If, before reaching any such state q, the action $\checkmark$ has not occurred (recall that we consider processes as trees), then we can ensure that $\checkmark$ is reachable with z by adding to any such a state r of t a τ-labeled edge to such a state r'. □

For the example process $x = a \cdot (b + b \cdot \checkmark + c \cdot \checkmark)$, the successful covering test $t = a \cdot \circledast$ can be expanded using the X_3-strategy $z' = a \cdot c \cdot \checkmark$ into the X_3-strategy $z = a \cdot \circledast \cdot \tau \cdot c \cdot \checkmark$.

Lemma A.1.18 (expansion implies successful covering test).
Let x be a process that does not contain the action $\circledast$. Every restricted test that can be expanded into an X_3-strategy of process x is a successful covering test on x. ⌟

Proof.
Let x be a process that does not contain the action $\circledast$. Let t be a restricted test, and let z be an expansion of t that is an X_3-strategy of process x. What remains to be proved is that t is a successful covering test on x.

X_3-strategy z is a covering process for x, even if some branches of z are removed. So every test that can be expanded into z is a covering test on x.

To show that t is a successful test on x, we distinguish between two kinds of occurrences of $\checkmark$ in z. Each action $\checkmark$ that is also present in the restricted test t is directly followed by a trivial test. Each other action $\checkmark$ in z is in a branch that occurs after an action $\circledast$ in t. In both cases it holds that, as z is an X_3-strategy of x, test t is a successful test on x. $\square$

X_3-conformance implies covering restricted testing

Using this theory on covering restricted testing, we can prove that X_3-conformance implies covering restricted testing.

Theorem A.1.19 (X_3-conformance implies covering restricted testing). For any two processes x and y that do not contain the action $\circledast$ holds:

$$x \sqsubseteq_{\mathit{conf},X_3} y \ \Rightarrow\ x \sqsubseteq_{\mathit{crt}} y \quad . \qquad \lrcorner$$

Proof.
Let x and y be processes that do not contain the action $\circledast$. Let $x \sqsubseteq_{\mathit{conf},X_3} y$; that is, each X_3-strategy of y is an X_3-strategy of x. To prove $x \sqsubseteq_{\mathit{crt}} y$, let t be a restricted test that is a successful covering test on y. What remains to be proved is that t is a successful covering test on x.

Using Lemma A.1.17, there exists an X_3-strategy z of y such that z is an expansion of t. Using $x \sqsubseteq_{\mathit{conf},X_3} y$, z is also an X_3-strategy of x. Using Lemma A.1.18, restricted test t is a successful covering test on x. $\square$

The example processes $x = a \cdot \checkmark + b \cdot \checkmark$ and $y = \tau \cdot a \cdot \checkmark + b \cdot \checkmark$ are related by $x \sqsubseteq_{\mathit{conf},X_3} y$ and $x \sqsubseteq_{\mathit{crt}} y$. In the proof, the example test $t = a \cdot \circledast$ is replaced by the X_3-strategy $z = a \cdot \circledast \cdot \tau \cdot \checkmark$.

From the two implications presented in Theorems A.1.15 and A.1.19, we immediately conclude that X_3-conformance is equivalent to covering restricted testing.

Corollary A.1.20 (X_3-conformance coincides with cov. restr. testing). For any two processes x and y that do not contain the action $\circledast$ holds:

$$x \sqsubseteq_{\mathit{conf},X_3} y \ \Leftrightarrow\ x \sqsubseteq_{\mathit{crt}} y \quad . \qquad \lrcorner$$

Based on Corollary A.1.20, the differences between X_3-conformance and fair testing can be summarized as follows:

- *success and termination*: In case of X_3-conformance, the composed processes must terminate together (using a synchronized action $\checkmark$). In case of fair testing, only the test needs to perform an (unsynchronized) "success" action $\circledast$.
- *testing for termination*: In case of X_3-conformance, there is no information about the states of the process after any occurrence of action $\checkmark$. In case of fair testing, there is information about each state of the process.

- *uncoverable states*: In case of X_3-conformance, there is no information about uncoverable states of the process. In case of fair testing, there is information about each state of the process.

The first item has been addressed by the different actions ✓ and $\circledast$. Item two and three have been addressed by introducing the restricted testing preorder and the covering restricted testing preorder, respectively.

A.1.4. Covering restricted testing versus restricted testing

In this section, we study in which cases restricted testing and X_3-conformance coincide, or rather (based on Corollary A.1.20), in which cases restricted testing and covering restricted testing coincide. Instead of restricting the considered tests, as we did in the previous section, we introduce two restrictions on the tested processes.

Restricted testing implies covering restricted testing

In Theorem A.1.11, we proved that restricted testing implies X_3-conformance, and in Theorem A.1.19, we proved that X_3-conformance implies covering restricted testing. Thus, we conclude that restricted testing implies covering restricted testing.

Corollary A.1.21 (restricted testing implies cov. restricted testing). For any two processes x and y that do not contain the action $\circledast$ holds:

$$x \sqsubseteq_{\boldsymbol{rt}} y \;\Rightarrow\; x \sqsubseteq_{\boldsymbol{crt}} y \quad . \qquad \lrcorner$$

In the remainder of this section, we focus on conditions under which we can prove, for any processes x and y, that $x \sqsubseteq_{\boldsymbol{crt}} y \;\Rightarrow\; x \sqsubseteq_{\boldsymbol{rt}} y$. In contrast to the preorder $\sqsubseteq_{\boldsymbol{rt}}$, the preorder $\sqsubseteq_{\boldsymbol{crt}}$ gives no information on X_3-uncontrollable processes. Therefore, we have to focus on X_3-controllable processes.

Pruned processes

In Section A.1.3, covering tests were introduced to avoid distinguishing between different uncoverable states. To avoid this restriction on the tests, we introduce a normal form for the uncoverable states of the process being tested.

X_3-uncontrollable processes have no X_3-strategies or covering tests, whereas the smallest set of successful tests (in terms of fair testing) is the set of trivial tests. The idea is to introduce a normal form such that uncoverable states correspond to states for which only the trivial tests are successful.

The simplest X_3-uncontrollable process, viz., δ, is not a candidate for this normal form, as nontrivial tests like $a + \circledast$ are successful on process δ, but not on the X_3-uncontrollable process a, for instance. The next lemma gives a normal form U for the uncoverable states, and proves that U is an X_3-uncontrollable process and that every successful test on U is trivial.

Lemma A.1.22 (normalized uncontrollable process).
Let U be the process such that only $U \xrightarrow{a} U$, for every action a, and $U \xrightarrow{\tau} \delta$. Process U is X_3-uncontrollable, and all successful tests on U are trivial tests. ⌟

Proof.
Process U is X_3-uncontrollable because of $U \xrightarrow{\tau} \delta$. To show that all successful tests are trivial tests, consider any process x that does not contain the action $\circledast$, and any test t that is a successful test on U. Process U is such that it can mimic every execution of x and afterwards perform the τ-action and block. Test t is a successful test on U, and hence t is a successful test on every process x. $\square$

Instead of process U, as defined in Lemma A.1.22, we could also use the catastrophic divergent process *CHAOS* from CSP [BHR84], but process *CHAOS* is more complicated than needed.

Definition A.1.23 (pruned process).
A process is *pruned* iff every first uncoverable state is equal to U. ⌟

Note that the first uncoverable states are well-defined, as we consider processes as trees with one initial state. Any process can be transformed into an X_3-conformance-equivalent pruned process; *pruning* is the operation of replacing each first uncoverable state (including all outgoing edges) by U. Pruning the example process $x = a \cdot (b + b \cdot \checkmark + c \cdot \checkmark)$ yields the process $a \cdot (b \cdot U + b \cdot U + c \cdot \checkmark)$, which is bisimilar to $a \cdot (b \cdot U + c \cdot \checkmark)$. Moreover, every pruned X_3-uncontrollable process is equal to U.

Vulnerable set of processes

To prove that $\sqsubseteq_{crt}$ implies $\sqsubseteq_{rt}$, it turns out to be insufficient to restrict the processes to pruned processes. For example, consider the two processes $x = a \cdot \checkmark$ and $y = a \cdot \checkmark + b \cdot U$, which are X_3-controllable and pruned processes containing the synchronized actions a and b. The term $b \cdot U$ in y can be interpreted as that no X_3-strategy may synchronize on action b in this state; in contrast, an X_3-strategy of x may offer synchronization on b. In this case, $x \sqsubseteq_{conf,X_3} y$ holds, and hence $x \sqsubseteq_{crt} y$ holds by Theorem A.1.19, but the restricted test $t = b \cdot \circledast$ is successful on y, but not on x.

The problem is the action on the edge from a coverable to an uncoverable state. Although the proposed kind of pruning does not change the semantics in terms of $\sqsubseteq_{conf,X_3}$ and of $\sqsubseteq_{crt}$, removing this action would change the semantics, and hence we do not consider this to be feasible.

To find a sufficient condition on the processes, let us consider a typical proof attempt for $x \sqsubseteq_{crt} y$ implies $x \sqsubseteq_{rt} y$, for any two processes x and y. Such a proof starts with a successful $\sqsubseteq_{rt}$-test t on y. To use the $\sqsubseteq_{crt}$-relation, test t should be transformed into a $\sqsubseteq_{crt}$-test. The following lemma describes a way to do so.

Lemma A.1.24 (successful test implies successful covering test).
Let x be an X_3-controllable and pruned process that does not contain the action $\circledast$. Any successful test on x can be transformed into a successful covering test on x in two steps. First, replace each trivial test by action $\circledast$. Second, replace some branches $a \cdot \circledast$, where action a can move x from a coverable to an uncoverable state, by action $\circledast$. ⌟

Proof.
Let x be an X_3-controllable and pruned process that does not contain the action $\circledast$. Let t be a successful test on x.

As $\circledast$ is a trivial test, replacing in t every trivial test by action $\circledast$ yields a successful test t' on x. Consider in the composition $x \| t'$ each first reachable state $q' \| r'$, such that q' is an uncoverable state of x and r' is a state of t'. As x is X_3-controllable, there is also a reachable state $q \| r$, in which q is a coverable state of x and r is a state of t, such that there is an edge $q \| r \xrightarrow{a} q' \| r'$ for a synchronized action a different from $\checkmark$. As x is pruned, q' is equal to U. As every trivial test in t' is equal to $\circledast$, we have $r' = \circledast$. By replacing in t' each such branch $a \cdot \circledast$ by the action $\circledast$, we obtain a successful covering test on x. □

For the example process $x = a \cdot (b + b \cdot \checkmark + c \cdot \checkmark)$, the successful test $a \cdot b \cdot (\circledast + a \cdot \circledast)$ is first transformed into the successful test $a \cdot b \cdot \circledast$, and then into the successful covering test $a \cdot \circledast$.

Using Lemma A.1.24, a successful $\sqsubseteq_{rt}$-test t on y can be transformed into a successful $\sqsubseteq_{crt}$-test t' on y. Using the given $\sqsubseteq_{crt}$-relation, test t' is a successful $\sqsubseteq_{crt}$-test on x. To conclude that t is a successful $\sqsubseteq_{rt}$-test on x, we have to replace in t' some actions $\circledast$ by a branch $a \cdot \circledast$, for some synchronized action a. However, it is not guaranteed that a can be accepted, and hence a deadlock may be introduced.

For the counterexample in the beginning of this section (before Lemma A.1.24), test $t = b \cdot \circledast$ is transformed into the successful covering test $t' = \circledast$ on x. To conclude that t is successful on x, action b should be accepted by x, but this is not the case.

To avoid that only one process can synchronize on such an action, we assume not only that the processes have been pruned, but also that the considered set of processes is vulnerable.

Definition A.1.25 (vulnerable set of processes).
A set of processes is *vulnerable* iff every action that can move some of the processes from a coverable state (where $\checkmark$ has not yet occurred) to an uncoverable state, is an action that can be accepted in every coverable state of each of the processes. A process x is vulnerable iff $\{x\}$ is a vulnerable set of processes. ⌟

This may sound like a severe restriction, but in Section A.1.5 we will see that vulnerable sets of processes are quite common in practice. Furthermore, pruning behaves nicely in terms of a vulnerable set of processes, as it only normalizes uncoverable states.

Lemma A.1.26 (pruning does not affect vulnerability).
Pruning each process in a vulnerable set of processes yields a vulnerable set of pruned processes. ⌟

Covering restricted testing implies restricted testing

Using this theory on pruning and vulnerable processes, we can prove that covering restricted testing implies restricted testing.

Theorem A.1.27 (cov. restricted testing implies restricted testing).
Let x and y be two X_3-controllable and pruned processes that do not contain the action $\circledast$. If x and y are in a vulnerable set of processes, then the following holds:

$$x \sqsubseteq_{crt} y \;\Rightarrow\; x \sqsubseteq_{rt} y \quad .$$ ⌟

Proof.
Let x and y be X_3-controllable and pruned processes that do not contain the action $\circledast$, and that are in a vulnerable set of processes. Let $x \sqsubseteq_{crt} y$; that is, for each restricted test t'', it holds that if t'' is a successful covering test on y, then t'' is a successful covering test on x. To prove $x \sqsubseteq_{rt} y$, let t be a restricted test that is successful on y. What remains to be proved is that t is successful on x.

Based on Lemma A.1.24, we replace in t each trivial test by action $\circledast$, yielding a successful test t' on y. Afterwards we replace some branches $a \cdot \circledast$ (where a can move y from a coverable state to an uncoverable state) by action $\circledast$, yielding a successful covering test t'' on y. As all trivial tests in t' had been replaced by $\circledast$, no $\circledast$ occurs before such an a in t'. As test t is restricted, tests t' and t'' are also restricted, and hence no $\checkmark$ occurs before such an a in t'.

Using $x \sqsubseteq_{crt} y$, test t'' is a successful covering test on x. To obtain t again, we first replace some actions $\circledast$ by a branch $a \cdot \circledast$. For the actions $\circledast$ that are not reachable in the composition $x \| t''$, this cannot harm success on x. Every action $\circledast$ that is reachable in the composition $x \| t''$ starts in a coverable state of x, as t'' is a covering test on x. Action a can be accepted in this state, as x and y are in a vulnerable set of processes and action a could move y from a coverable state (where $\checkmark$ had not yet occurred) to an uncoverable state. Finally, we replace some actions $\circledast$ in successful test t' on x by a trivial test, yielding test t, which is guaranteed to be successful on x. □

From Corollary A.1.21 and Theorem A.1.27 we immediately conclude that, for certain processes, covering restricted testing is equivalent to restricted testing.

Corollary A.1.28 (cov. restr. testing coincides with restr. testing).
Let x and y be two X_3-controllable and pruned processes that do not contain the action $\circledast$. If x and y are in a vulnerable set of processes, then

$$x \sqsubseteq_{crt} y \;\Leftrightarrow\; x \sqsubseteq_{rt} y \quad .$$ ⌟

As the notions of X_3-conformance and covering restricted testing are equivalent (see Corollary A.1.20), we can relate X_3-conformance and restricted testing.

Corollary A.1.29 (X_3-conformance coincides with restricted testing). Let x and y be two X_3-controllable and pruned processes that do not contain the action $\circledast$. If x and y are in a vulnerable set of processes, then

$$x \sqsubseteq_{\mathit{conf},X_3} y \;\Leftrightarrow\; x \sqsubseteq_{\mathit{rt}} y \quad . \qquad \lrcorner$$

These restrictions are not yet enough to prove $x \sqsubseteq_{\mathit{conf},X_3} y \;\Rightarrow\; x \sqsubseteq_{\mathit{ft}} y$. Consider the counterexample $x = a \cdot \checkmark + \checkmark$ and $y = a \cdot \checkmark$, where x and y are in a vulnerable set of pruned processes. In relation to process y, process x only makes it easier to terminate properly, so $x \sqsubseteq_{\mathit{conf},X_3} y$ holds. However, the (non-restricted) test $t = \circledast + \checkmark$ is a successful test on y, but not on x. In particular, this shows that fair testing and X_3-conformance do not coincide for processes in which all states are coverable, although this was claimed by [MSR06].

A.1.5. Application to asynchronous processes

In this section, we study the computability of the introduced concepts, and show that all processes that only communicate asynchronously are in a vulnerable set.

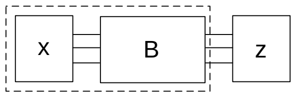
Computability

We first sketch some decision procedures for the introduced concepts. For these procedures, we restrict ourselves to processes with a finite number of states.

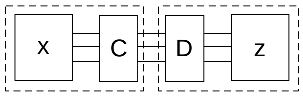
To compute the coverable states, we use the notion of the most-permissive X_3-strategy (cf. Definition 4.3.1), because it can visit all the states that can be visited using any X_3-strategy. So, the coverable states of a process are the states that can be visited by the most-permissive X_3-strategy.

The coverable and uncoverable states can be used to decide whether (sets of) processes are pruned or vulnerable. For any process, an X_3-conformance-equivalent pruned process can be obtained by replacing all first uncoverable states by process U. However, in general there exists no X_3-conformance-equivalent process that is vulnerable.

For example, consider the process $x = a + b \cdot a \cdot \checkmark$. In what follows, we collect necessary conditions on a vulnerable process y that is X_3-conformance-equivalent to x, and show a contradiction. Process x and y have $b \cdot a \cdot \checkmark$ as an X_3-strategy; hence, all the states in y that can be visited with this X_3-strategy are coverable. However, $b \cdot a \cdot \checkmark + a \cdot z'$, for any process z', is not an X_3-strategy of x and y; hence, a is an action that can move y from a coverable state to an uncoverable state. Finally, $b \cdot a \cdot (\checkmark + a)$ is an X_3-strategy of x and y; hence, y cannot accept a in a certain coverable state. So there exists no vulnerable process y that is X_3-conformance-equivalent to x.



(a) One buffer (B)



(b) Two buffers (C and D)

Figure A.3.: Buffering schemes for a process x and an X_3-strategy z.

Modeling asynchronous communication

Processes that communicate asynchronously can be modeled as processes that communicate synchronously by explicitly introducing the communication buffers between them as additional processes (see also a brief remark in [FHRR04]). In line with [BZ09], we consider unbounded uni-directional buffers that are empty in the initial state and that must be empty on termination.

The concept of a communication buffer has already been introduced in Section 2.7.2 for service automata. To model asynchronous communication, the pending messages in the composition of two service automata are stored in a message bag. That means, the message bag implements the functionality of a buffer process.

Let MT be a set of message types. Let the asynchronous processes use synchronized actions (apart from $\checkmark$) of the shape $?m$ and $!m$, for $m \in MT$, to denote respectively receiving and sending a message of type m. For each message type m, a buffer can be modeled as $B_m(0)$, where B_m is defined as:

$$B_m(0) \ = \ \checkmark + !m \cdot B_m(1) \quad , \quad B_m(n+1) \ = ?m \cdot B_m(n) + !m \cdot B_m(n+2)$$

In $B_m(n)$, the natural number n denotes the number of messages in the buffer. Synchronization on action $!m$ puts a message in the buffer, whereas synchronization on $?m$ gets a message from the buffer. The buffer can only terminate properly when it is empty.

Thus, asynchronous communication using a set of message types MT can be modeled as follows. Let x be a given asynchronous process and z be an asynchronous X_3-strategy that use disjoint sets of synchronized actions for the message types MT. Let B denote the merge $\|_{\{\checkmark\}}$ of B_m, for every $m \in MT$. The buffered composition of x and z can be modeled as $x \parallel B \parallel z$; in addition to synchronization over $\checkmark$, the first merge synchronizes over the synchronized actions of the buffer for x, and the second merge synchronizes over the synchronized actions of the buffer for z.

To keep the binary setting of a given process x and an X_3-strategy z, and to avoid imposing restrictions on the set of all X_3-strategies, we integrate the buffers with each given process. The resulting process is $x \parallel B$; see Figure A.3(a). Thus, the X_3-strategy z can send a message of type m to the process x using a synchronized action $!m$ of the buffer; receiving a message from z is an internal action of the composition of the process x and the buffer B. Similarly, the X_3-strategy z can receive a message of type m from the process x using a synchronized action $?m$ of the buffer; sending a message to z is an internal action of the composition of the process x and the buffer B. We call such an integrated process $x \parallel B$ an (I, O)-buffered process.

Definition A.1.30 ((I, O)-buffered process).
Let I and O be two disjoint sets of message types. An (I, O)-*buffered process* is a process where each action is either

1. an internal (unsynchronized) action;
2. the synchronized action $\checkmark$;
3. a synchronized action that gets a message from an output buffer for type $m \in O$; or
4. a synchronized action that puts a message in an input buffer for type $m \in I$.⌟

Properties of asynchronous processes

It is well-known that two (unbounded) buffers in series behave as one (unbounded) buffer; that is, they are branching bisimilar after hiding the synchronized actions in between them. So, B_m can be replaced by $C_m \parallel_{\{\checkmark,m\}} D_m$, where C_m and D_m are buffers that are obtained from B_m by renaming some of the actions:

$$C_m(0) \;=\; \checkmark + !m \cdot C_m(1) \quad , \quad C_m(n+1) \;=\; m \cdot C_m(n) + !m \cdot C_m(n+2)$$

$$D_m(0) \;=\; \checkmark \;+\; m \cdot D_m(1) \quad , \quad D_m(n+1) \;= ?m \cdot D_m(n) \;+\; m \cdot D_m(n+2)$$

Thus, using the associativity of the merge, any X_3-strategy z of an (I, O)-buffered process $x \| B$ can be transformed into a buffered X_3-strategy $D \| z$ for $x \| C$ (or vice versa), without affecting weak termination; see Figure A.3(b). As $x \| B$ and $x \| C$ are identical up to renaming some synchronized actions (like m), $D \| z$ can be transformed into an (O, I)-buffered X_3-strategy of $x \| B$ by some renaming.

Let us investigate which actions can move a buffered process from a coverable to an uncoverable state.

Lemma A.1.31 (characterization of actions).
Let x be an (I, O)-buffered process, let q be a coverable state of x, and let a be an action such that $q \xrightarrow{a} q'$, for some state q' of x. State q' is guaranteed to be coverable if action a is

1. an internal (unsynchronized) action;
2. the synchronized action $\checkmark$; or
3. a synchronized action $?m$ of type $m \in O$ that gets a message from an output buffer. ⌟

Proof.
Let x be an (I,O)-buffered process, let q be a coverable state of x, and let a be an action such that $q \xrightarrow{a} q'$, for some state q' of x.

As q is coverable, there is an X_3-strategy z that can visit q while being in a state r. To show that q' is coverable, we show how to construct an X_3-strategy that can visit state q' in case X_3-strategy z does not:

1. As internal action a is unsynchronized, X_3-strategy z can also visit q'.
2. Adding in r a $\checkmark$-labeled outgoing edge gives an X_3-strategy that can visit q'.
3. Replacing X_3-strategy z by an (O,I)-buffered X_3-strategy, as described before, gives an X_3-strategy that can synchronize on $?m$ in every state, and hence it can visit q'. □

So, the only actions that can move an (I,O)-buffered process from a coverable state to an uncoverable state are actions that put a message into an input buffer. As the buffers are unbounded, these actions can be accepted in every state before any occurrence of action $\checkmark$. Combining these ingredients, we can conclude that certain sets of (I,O)-buffered processes are vulnerable.

Lemma A.1.32 ((I,O)-buffered processes are vulnerable).
Let I and O be two disjoint sets of message types. Any set of (I,O)-buffered processes is a vulnerable set of processes. ⌟

Combining our previous results, we obtain the following result for asynchronous processes.

Corollary A.1.33 (asynchronous processes are vulnerable).
Let I and O be two disjoint sets of message types. Let x and y be two asynchronous processes that only contain the synchronized actions $\checkmark$, $?m$ for $m \in I$, and $!m$ for $m \in O$, but that do not contain the action $\circledast$. Let B be the buffer that corresponds to I and O. Let x' be the result of pruning the (I,O)-buffered process $x \| B$, and let y' be the result of pruning the (I,O)-buffered process $y \| B$. If $x \| B$ and $y \| B$ are X_3-controllable, then

$$x' \sqsubseteq_{\boldsymbol{conf},X_3} y' \;\Leftrightarrow\; x' \sqsubseteq_{\boldsymbol{rt}} y' \quad .$$ ⌟

A.2. Relationship between conformance and failures

In this section, we study the relationship between the X_1-conformance preorder (recall that $X_1 = \{\text{deadlock freedom}\}$) and known process equivalences and preorders. The notion of a deadlock has been investigated in the *stable failures model* [BHR84, Hoa85, Ros98] in process algebra. We show that the stable failures preorder and the X_1-conformance preorder are equivalent.

We use the notion of processes as introduced in Section A.1.1. In particular, we use the synchronized action $\checkmark$ to model termination and ignore all states after the first $\checkmark$ has occurred.

In the stable failures model, a process x is represented by the set *Failures*(x) of its *failures*. A failure of x is a pair (σ, Y). With σ we denote a finite sequence of actions of x—that is, $x \overset{\sigma}{\Longrightarrow} x'$. The set $Y \subseteq Act$ denotes the *refusal set* of x'. A refusal set contains actions that are refused by process x after having executed σ. That is, for all actions $a \in Y \cup \{\tau\}$ holds: $x' \overset{a}{\not\longrightarrow}$. Note that the stable failures model only considers states x' of x that do not enable any τ-transition. Such states x' are called *stable*. Process x *failure refines* process y if and only if the failures of x are contained in the failures of y.

Definition A.2.1 (stable failures preorder).
Let x and y be processes. The set *Failures*(x) of *failures* of x is the least set satisfying $(\sigma, Y) \in Failures(x)$ if there is a process x' with $x \overset{\sigma}{\Longrightarrow} x'$ such that for all $a \in Y \cup \{\tau\}$ holds: $x' \overset{a}{\not\longrightarrow}$. Processes x and y are related in the *stable failures preorder*, denoted by $x \sqsubseteq_{\boldsymbol{f}} y$, iff $Failures(x) \subseteq Failures(y)$. ⌟

As an example, consider a process $x = a+b$ with $H = \{a, b\}$. The set of failures of x is defined by $Failures(x) = \{(\epsilon, \emptyset), (a, \{a, b\}), (b, \{a, b\})\}$.

The stable failures preorder $\sqsubseteq_{\boldsymbol{f}}$ is a precongruence with respect to composition by synchronization. Given a process $x \| r$, by the operational rules, we have $Failures(x \| r) = \{(\sigma, Y \cup Z) \mid (\sigma, Y) \in Failures(x) \wedge (\sigma, Z) \in Failures(r)\}$.

We want to compare the stable failures preorder $\sqsubseteq_{\boldsymbol{f}}$ with the X_1-conformance preorder $\sqsubseteq_{\boldsymbol{conf},X_1}$. The following theorem justifies that both preorders are equivalent.

Theorem A.2.2 (X_1-conformance coincides with stable failures).
For any two processes x and y holds:

$$x \ \sqsubseteq_{\boldsymbol{conf},X_1} \ y \ \Leftrightarrow \ x \ \sqsubseteq_{\boldsymbol{f}} \ y \quad .$$ ⌟

Proof.
Let $x \not\sqsubseteq_{\boldsymbol{f}} y$; that is, we have $Failures(x) \not\subseteq Failures(y)$. We will show that $x \not\sqsubseteq_{\boldsymbol{conf},X_1} y$.

Choose a refusal set Z and a finite sequence $\sigma = a_1 \cdot \ldots \cdot a_n$ of actions such that $(\sigma, Z) \in Failures(x) \setminus Failures(y)$. Let r be the process $\tau^* \cdot a_1 \cdot \tau^* \cdot \ldots \cdot a_n \cdot (\Sigma_{z \in Z} z) \cdot \tau^* \cdot \delta$. Clearly, $x \| r$ contains a deadlock and, by assumption, $y \| r$ does not. As r diverges (or in case of a synchronization on action $\checkmark$, a final state is reached), it is an X_1-strategy of y, but it is not an X_1-strategy of x. Consequently, $x \not\sqsubseteq_{\boldsymbol{conf},X_1} y$.

Let $x \sqsubseteq_{\boldsymbol{f}} y$; that is, we have $Failures(x) \subseteq Failures(y)$. We will show that $x \sqsubseteq_{\boldsymbol{conf},X_1} y$.

Choose an arbitrary process r. As $\sqsubseteq_{\boldsymbol{f}}$ is a precongruence, we have $x \| r \sqsubseteq_{\boldsymbol{f}} y \| r$. Suppose r is an X_1-strategy of y; that is, for all $\sigma \in Act^*$ holds, $(\sigma, Act) \notin Failures(y \| r)$. From $\sqsubseteq_{\boldsymbol{f}}$ being a precongruence we conclude that, for all $\sigma \in$

Act^* holds, $(\sigma, Act) \notin \mathit{Failures}(x \| r)$; that is, r is an X_1-strategy of x as well. Consequently, $x \sqsubseteq_{\boldsymbol{conf}, X_1} y$. □

Theorem A.2.2 shows that the X_1-conformance preorder is, in fact, not a novel preorder. The equivalence with the stable failures preorder, which is well-studied, simplifies the comparison of X_1-conformance with related preorders in the literature. As must testing is equivalent to stable failures—at least for finitely branching processes without divergences [Nic87]—we conclude that X_1-conformance takes the same position than musting (MT) in Figure A.1.

Bibliography

[Aal98] Wil M. P. van der Aalst. The application of Petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998. [35, 44, 126, 216]

[Aal03] Wil M. P. van der Aalst. Inheritance of interorganizational workflows: How to agree to disagree without loosing control? *Information Technology and Management Journal*, 4(4):345–389, 2003. [23]

[AB02] Wil M. P. van der Aalst and Twan Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science*, 270(1-2):125–203, 2002. [22, 175, 178, 179, 216, 221, 229]

[ABH$^+$07] Wil M. P. van der Aalst, Michael Beisiegel, Kees M. van Hee, Dieter König, and Christian Stahl. An SOA-based architecture framework. *International Journal of Business Process Integration and Management*, 2(2):91–101, 2007. [18, 200]

[AFFK05] Jesús Arias-Fisteus, Luis Sánchez Fernández, and Carlos Delgado Kloos. Applying model checking to BPEL4WS business collaborations. In Hisham Haddad, Lorie M. Liebrock, Andrea Omicini, and Roger L. Wainwright, editors, *Symposium on Applied Computing (SAC 2005)*, pages 826–830. ACM, 2005. [85]

[AGU72] Alfred V. Aho, M. R. Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1(2):131–137, 1972. [112, 120]

[AH01] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *ESEC / SIGSOFT FSE*, pages 109–120, 2001. [219]

[AH02] Wil M. P. van der Aalst and Kees M. van Hee. *Workflow Management: Models, Methods, and Systems.* MIT press, Cambridge, MA, 2002. [35]

[AHM$^+$09] Wil M. P. van der Aalst, Kees M. van Hee, Peter Massuthe, Natalia Sidorova, and Jan Martijn E. M. van der Werf. Compositional service trees. In Giuliana Franceschinis and Karsten Wolf, editors, *International Conference on Applications and Theory of Petri Nets (PETRI*

NETS 2009), volume 5606 of *Lecture Notes in Computer Science*, pages 283–302. Springer, 2009. [216, 217, 221, 222]

[AL08] Wil M. P. van der Aalst and Kristian Bisgaard Lassen. Translating unstructured workflow processes to readable BPEL: Theory and implementation. *Information & Software Technology*, 50(3):131–159, 2008. [55]

[ALM$^+$08] Wil M. P. van der Aalst, Niels Lohmann, Peter Massuthe, Christian Stahl, and Karsten Wolf. From public views to private views - correctness-by-design for services. In *International Workshop on Web Services and Formal Methods (WS-FM 2007)*, volume 4937 of *Lecture Notes in Computer Science*, pages 139–153. Springer, 2008. [27, 59, 61, 176, 178]

[ALM$^+$09] Wil M. P. van der Aalst, Niels Lohmann, Peter Massuthe, Christian Stahl, and Karsten Wolf. Multiparty Contracts: Agreeing and Implementing Interorganizational Processes. *The Computer Journal*, 2009. (Accepted for publication). [27, 41, 59, 61, 133, 135, 212]

[Alo08] Gustavo Alonso. Challenges and opportunities for formal specifications in service oriented architectures. In Kees M. van Hee and Rüdiger Valk, editors, *International Conference on Applications and Theory of Petri Nets (PETRI NETS 2008)*, volume 5062 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 2008. [21]

[Alv07] Alexandre Alves et al. Web Services Business Process Execution Language Version 2.0. OASIS Standard, 11 April 2007, OASIS, April 2007. [19, 54, 85, 200]

[AMSW09] Wil M. P. van der Aalst, Arjan J. Mooij, Christian Stahl, and Karsten Wolf. Service interaction: Patterns, formalization, and analysis. In Marco Bernardo, Luca Padovani, and Gianluigi Zavattaro, editors, *Formal Methods for Web Services (SFM 2009)*, volume 5569, pages 42–88. Springer, April 2009. [176, 229]

[And03] Tony Andrews et al. Business Process Execution Language for Web Services, Version 1.1. Specification, BEA Systems, IBM, Microsoft, May 2003. [55]

[AQR$^+$04] Tony Andrews, Shaz Qadeer, Sriram K. Rajamani, Jakob Rehof, and Yichen Xie. Zing: A Model Checker for Concurrent Software. In *International Conference on Computer Aided Verification (CAV 2004)*, volume 3114 of *Lecture Notes in Computer Science*, pages 484–487. Springer, 2004. [218]

[AW01] Wil M. P. van der Aalst and Mathias Weske. The P2P approach to interorganizational workflows. In *International Conference on Advanced Information Systems (CAiSE 2001)*, volume 2068 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2001. [23]

[BA01] Twan Basten and Wil M. P. van der Aalst. Inheritance of Behavior. *Journal of Logic and Algebraic Programming*, 47(2):47–145, 2001. [175, 178, 179, 216, 221]

[Bau88] Bernd Baumgarten. On Internal and External Characterizations of PT-net Building Block Behaviors. In *Advances in Petri Nets 1988*, volume 340 of *Lecture Notes in Computer Science*, pages 44–61. Springer, 1988. [59]

[BBCG08] Filippo Bonchi, Antonio Brogi, Sara Corfini, and Fabio Gadducci. On the use of behavioural equivalences for web services' development. *Fundam. Inform.*, 89(4):479–510, 2008. [216]

[BCH05] Dirk Beyer, Arindam Chakrabarti, and Thomas A. Henzinger. Web service interfaces. In Allan Ellis and Tatsuya Hagino, editors, *International World Wide Web Conference (WWW 2005)*, pages 148–159. ACM, 2005. [220]

[BCT06] B. Benatallah, F. Casati, and F. Toumani. Representing, Analysing and Managing Web Service Protocols. *Data Knowl. Eng.*, 58(3):327–357, 2006. [219, 220]

[BD98] Eric Badouel and Philippe Darondeau. Theory of Regions. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, pages 529–586. Springer, 1998. [52]

[Ber87] Gérard Berthelot. Transformations and decompositions of nets. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advances in Petri Nets 1986, Part I*, volume 254 of *Lecture Notes in Computer Science*, pages 359–376. Springer, 1987. [221]

[BHR84] Stephen D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984. [63, 241, 247]

[BNP99] Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Basic observables for processes. *Inf. Comput.*, 149(1):77–98, 1999. [237]

[Bre07] Jan Bretschneider. Produktbedienungsanleitungen zur Charakterisierung austauschbarer Services. Diplomarbeit, Humboldt-Universität zu Berlin, March 2007. [156]

[Bru04] Stefan D. Bruda. Preorder relations. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors, *Model-Based Testing of Reactive Systems, Advanced Lectures*, volume 3472 of *Lecture Notes in Computer Science*, pages 117–149, Dagstuhl, Germany, January 2004. Springer. [233]

[BRV95] Ed Brinksma, Arend Rensink, and Walter Vogler. Fair testing. In Insup Lee and Scott A. Smolka, editors, *International Conference on Concurrency Theory (CONCUR 1995)*, volume 962 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 1995. [63, 231, 233]

[BW90] J.C.M. Baeten and W. Weijland. *Process Algebra.* Cambridge University Press, 1990. [232]

[BZ07a] Mario Bravetti and Gianluigi Zavattaro. Contract Based Multi-party Service Composition. In Farhad Arbab and Marjan Sirjani, editors, *Symposium on Fundamentals of Software Engineering (FSEN 2007)*, volume 4767 of *Lecture Notes in Computer Science*, pages 207–222. Springer, 2007. [61, 64, 218, 219, 231, 234, 235, 236]

[BZ07b] Mario Bravetti and Gianluigi Zavattaro. A theory for strong service compliance. In *International Conference on Coordination Models and Languages (COORDINATION 2007)*, volume 4467 of *Lecture Notes in Computer Science*, pages 96–112, Paphos, Cyprus, 6-8 June 2007. Springer. [219]

[BZ08] Mario Bravetti and Gianluigi Zavattaro. A foundational theory of contracts for multi-party service composition. *Fundam. Inform.*, 89(4):451–478, 2008. [218, 219]

[BZ09] Mario Bravetti and Gianluigi Zavattaro. Contract compliance and choreography conformance in the presence of message queues. In *International Workshop on Web Services and Formal Methods (WS-FM 2008)*, volume 5387 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2009. [219, 245]

[CC77] Partick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Symposium on Principles of Programming Languages (POPL 1977)*, pages 238–252. ACM Press, New York, NY, jan 1977. [54]

[CE82] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logics of Programs 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1982. [43]

[CES86] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986. [43]

[CFJ93] Edmund M. Clarke, Thomas Filkorn, and Somesh Jha. Exploiting symmetry in temporal logic model checking. In Costas Courcoubetis, editor, *International Conference on Computer Aided Verification (CAV 1993)*, volume 697 of *Lecture Notes in Computer Science*, pages 450–462. Springer, 1993. [110]

[CGP00] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT, Cambridge, Massachusetts, 2000. [34, 55, 99, 109, 110]

[CGP08] Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A Theory of Contracts for Web Services. In George C. Necula and Philip Wadler, editors, *Symposium on Principles of Programming Languages (POPL 2008)*, pages 261–272, New York, NY, USA, 2008. ACM. [218, 229]

[Che93] Ghassan Chehaibar. Replacement of open interface subnets and stable state transformation equivalence. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1993*, volume 674 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 1993. [36]

[CMRW07] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Recommendation 26 June 2007, W3C, 2007. [18]

[CSS06] Edmund M. Clarke, Natasha Sharygina, and Nishant Sinha. Program compatibility approaches. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *International Symposium on Formal Methods for Components and Objects (FMCO 2005)*, volume 4111 of *Lecture Notes in Computer Science*, pages 243–258. Springer, 2006. [219]

[CVZ07] Ivana Cerná, Pavlina Vareková, and Barbora Zimmerová. Component Substitutability via Equivalencies of Component-Interaction Automata. In *International Workshop on Formal Aspects of Component Software (FACS 2006)*, pages 39–55. Elsevier ENTCS, 2007. [219]

[DA04] Juliane Dehnert and Wil M. P. van der Aalst. Bridging the gap between business models and workflow specifications. *Int. J. Cooperative Inf. Syst.*, 13(3):289–332, 2004. [99, 219]

[DDO08] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and analysis of business process models in BPMN. *Information and Software Technology*, 50(12):1281 – 1294, 2008. [26, 55]

[DK75] Frank DeRemer and Hans Kron. Programming-in-the large versus programming-in-the-small. In *International Conference on Reliable Software*, pages 114–121, New York, NY, USA, 1975. ACM. [17]

[DKLW07] Gero Decker, Oliver Kopp, Frank Leymann, and Mathias Weske. BPEL4Chor: Extending BPEL for modeling choreographies. In *International Conference on Web Services (ICWS 2007)*, pages 296–303. IEEE-CSP, 9-13 July 2007. [19, 55]

[DR98] Jörg Desel and Wolfgang Reisig. Place or transition petri nets. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, pages 122–173. Springer, 1998. [32]

[DW07] Gero Decker and Mathias Weske. Behavioral Consistency for B2B Process Integration. In John Krogstie, Andreas L. Opdahl, and Guttorm Sindre, editors, *International Conference on Advanced Information Systems Engineering (CAiSE 2007)*, volume 4495 of *Lecture Notes in Computer Science*, pages 81–95. Springer, 2007. [40]

[ES93] E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. In Costas Courcoubetis, editor, *International Conference on Computer Aided Verification(CAV 1993)*, volume 697 of *Lecture Notes in Computer Science*, pages 463–478. Springer, 1993. [110]

[FGK$^+$96] Jean-Claude Fernandez, Hubert Garavel, Alain Kerbrat, Laurent Mounier, Radu Mateescu, and Mihaela Sighireanu. CADP - A Protocol Validation and Verification Toolbox. In Rajeev Alur and Thomas A. Henzinger, editors, *International Conference on Computer Aided Verification (CAV 1996)*, volume 1102 of *Lecture Notes in Computer Science*, pages 437–440. Springer, 1996. [117]

[FHRR04] C. Fournet, C. A. R. Hoare, S. K. Rajamani, and J. Rehof. Stuck-Free Conformance. In R. Alur and D. Peled, editors, *International Conference on Computer Aided Verification (CAV 2004)*, volume 3114 of *Lecture Notes in Computer Science*, pages 242–254. Springer, 2004. [218, 245]

[Gie08] Christian Gierds. Strukturelle Reduktion von Bedienungsanleitungen. Diplomarbeit, Humboldt-Universität zu Berlin, January 2008. [89]

[Gla93] Rob J. van Glabbeek. The linear time - branching time spectrum ii. In Eike Best, editor, *International Conference on Concurrency Theory (CONCUR 1993)*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1993. [31, 63, 231]

[Gla01] Rob J. van Glabbeek. The Linear Time – Branching Time Spectrum I; The Semantics of Concrete, Sequential Processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier, Amsterdam, The Netherlands, 2001. [31, 63, 231, 233]

[GMW08] Christian Gierds, Arjan J. Mooij, and Karsten Wolf. Specifying and generating behavioral service adapter based on transformation rules. Preprint CS-02-08, Universität Rostock, Rostock, Germany, August 2008. [229]

[God91] Patrice Godefroid. Using partial orders to improve automatic verification methods. In Edmund M. Clarke and Robert P. Kurshan, editors, *International Workshop on Computer Aided Verification (CAV 1990)*, volume 531 of *Lecture Notes in Computer Science*, pages 176–185. Springer, 1991. [110]

[GS97] Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with pvs. In *International Conference on Computer Aided Verification (CAV 1997)*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 1997. [54]

[GV06] Rob J. van Glabbeek and Marc Voorhoeve. Liveness, fairness and impossible futures. In Christel Baier and Holger Hermanns, editors, *International Conference on Concurrency Theory (CONCUR 2006)*, volume 4137 of *Lecture Notes in Computer Science*, pages 126–141. Springer, 2006. [233, 234]

[GW96] Rob J. van Glabbeek and W. P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996. [31, 178]

[HAM08] Thomas S. Heinze, Wolfram Amme, and Simon Moser. Generic CSSA-Based Pattern over Boolean Data for an Improved WS-BPEL to Petri Net Mappping. In Abdelhamid Mellouk, Jun Bi, Guadalupe Ortiz, Dickson K. W. Chiu, and Manuela Popescu, editors, *International Conference on Internet and Web Applications and Services (ICIW 2008)*, pages 590–595. IEEE Computer Society, 2008. [208]

[Hoa85] C. A. R. Hoare. *Communicating sequential processes.* Prentice-Hall International series in computing science. Prentice-Hall International, Englewood Cliffs, N J London, 1985. [59, 63, 231, 247]

[HSV03] Kees M. van Hee, Natalia Sidorova, and Marc Voorhoeve. Soundness and separability of workflow nets in the stepwise refinement approach. In Wil M. P. van der Aalst and Eike Best, editors, *International Conference on Applications and Theory of Petri Nets (ICATPN 2003)*, volume 2679 of *Lecture Notes in Computer Science*, pages 337–356. Springer, 2003. [42, 43]

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979. [143, 157]

[Jen97] Kurt Jensen. *Coloured Petri Nets*, volume 1–3 of *EATCS Monographs on Theoretical Computer Science.* Springer, Berlin, Heidelberg, New York, 2nd edition, 1997. [54]

[JTM98] E. Y. T. Juan, J. J. P. Tsai, and Tadao Murata. Compositional Verification of Concurrent Systems Using Petri-Net-Based Condensation Rules. *ACM Trans. on Programming Languages and Systems*, 20(5):917–979, 1998. [111]

[KBR$^+$05] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, and Charlton Barreto. Web Services Choreography Description Language Version 1.0. W3C Candidate Recommendation 9 November 2005, W3C, Cambridge, Massachusetts, USA, 2005. [19]

[Kin97] Ekkart Kindler. A compositional partial order semantics for Petri net components. In Pierre Azéma and Gianfranco Balbo, editors, *International Conference on Application and Theory of Petri Nets (ICATPN 1997)*, volume 1248 of *Lecture Notes in Computer Science*, pages 235–252. Springer, June 1997. [36]

[KLM$^+$08] Dieter König, Niels Lohmann, Simon Moser, Christian Stahl, and Karsten Wolf. Extending the compatibility notion for abstract WS-BPEL processes. In Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *International World Wide Web Conference (WWW 2008)*, pages 785–794. ACM, April 2008. [27, 199]

[KMW07] Kathrin Kaschner, Peter Massuthe, and Karsten Wolf. Symbolic Representation of Operating Guidelines for Services. *Petri Net Newsletter*, 72:21–28, April 2007. [89]

[KV06] Orna Kupferman and Moshe Y. Vardi. *Interactive Computation - The New Paradigm*, chapter Verification of Open Systems, pages 97–118. Springer, September 2006. [228, 229]

[KVW00] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, 2000. [228]

[KW09] Kathrin Kaschner and Karsten Wolf. Set algebra for service behavior: Applications and constructions. In Umeshwar Dayal, Johann Eder, Jana Koehler, and Hajo Reijers, editors, *International Conference on Business Process Management (BPM 2009)*, volume 5701 of *Lecture Notes in Computer Science*, pages 193–210. Springer, September 2009. [98, 151, 152, 160, 172, 228]

[LK08] Niels Lohmann and Jens Kleine. Fully-automatic Translation of Open Workflow Net Models into Human-readable Abstract BPEL Processes. In Thomas Kühne, Wolfgang Reisig, and Friedrich Steimann, editors, *Modellierung 2008*, volume P-127 of *Lecture Notes in Informatics*, pages 57–72. GI, March 2008. [55]

[LKLR08] Niels Lohmann, Oliver Kopp, Frank Leymann, and Wolfgang Reisig. Analyzing BPEL4Chor: Verification and participant synthesis. In Marlon Dumas and Reiko Heckel, editors, *International Workshop on Web Services and Formal Methods (WS-FM 2007)*, volume 4937 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2008. [26, 55]

[LMSW06] Niels Lohmann, Peter Massuthe, Christian Stahl, and Daniela Weinberg. Analyzing Interacting BPEL Processes. In Schahram Dustdar, Jos Luiz Fiadeiro, and Amit Sheth, editors, *International Conference on Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*, pages 17–32. Springer, September 2006. [85]

[LMSW08] Niels Lohmann, Peter Massuthe, Christian Stahl, and Daniela Weinberg. Analyzing interacting WS-BPEL processes using flexible model generation. *Data Knowledge Engineering*, 64(1):38–54, 2008. [85]

[LMW07a] Niels Lohmann, Peter Massuthe, and Karsten Wolf. Behavioral constraints for services. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 271–287. Springer, September 2007. [170]

[LMW07b] Niels Lohmann, Peter Massuthe, and Karsten Wolf. Operating guidelines for finite-state services. In Jetty Kleijn and Alex Yakovlev, editors, *International Conference on Applications and Theory of Petri Nets (ICATPN 2007)*, volume 4546 of *Lecture Notes in Computer Science*, pages 321–341. Springer, 2007. [40, 68, 85, 88]

[Loh08] Niels Lohmann. A feature-complete Petri net semantics for WS-BPEL 2.0. In Marlon Dumas and Reiko Heckel, editors, *International Workshop on Web Services and Formal Methods (WS-FM 2007)*, volume 4937 of *Lecture Notes in Computer Science*, pages 77–91. Springer, 2008. [26, 55, 85, 208]

[LP07] Cosimo Laneve and Luca Padovani. The must preorder revisited. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *International Conference on Concurrency Theory (CONCUR 2007)*, volume 4703 of *Lecture Notes in Computer Science*, pages 212–225. Springer, 2007. [218]

[LVD09] Niels Lohmann, Eric Verbeek, and Remco M. Dijkman. Petri net transformations for business processes - a survey. *Transactions on Petri Nets and Other Models of Concurrency II, Special Issue on Concurrency in Process-Aware Information Systems*, 2:46–63, 2009. [26]

[LVOS08] Niels Lohmann, Eric Verbeek, Chun Ouyang, and Christian Stahl. Comparing and evaluating Petri net semantics for BPEL. *International Journal of Business Process Integration and Management*, 2008. (Accepted for publication). [55]

[LW09] Niels Lohmann and Karsten Wolf. Petrifying operating guidelines for services. In Stephen Edwards and Walter Vogler, editors, *International Conference on Application of Concurrency to System Design (ACSD 2009).* IEEE Computer Society, July 2009. [89]

[Lyn96] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996. [51]

[Mar05] Axel Martens. Analyzing Web Service based Business Processes. In Maura Cerioli, editor, *International Conference on Fundamental Approaches to Software Engineering (FASE 2005)*, volume 3442 of *Lecture Notes in Computer Science*, Edinburgh, Scotland, April 2005. Springer. [36, 40, 216, 217]

[Mas09] Peter Massuthe. *Operating Guidelines for Services.* PhD thesis, Technische Universiteit Eindhoven, The Netherlands, April 2009. [51, 53, 68, 73, 80, 81, 84, 89, 135, 138, 160]

[McI68] M. D. McIlroy. Mass Produced Software Components. In P. Naur and B. Randell, editors, *NATO Software Engineering Conference*, volume 1, pages 138–150, Garmisch, Germany, October 1968. [17]

[Mil89] Robin Milner. *Communication and Concurrency.* Prentice-Hall, Inc., 1989. [31]

[ML07] Nilo Mitra and Yves Lafon. SOAP Version 1.2 Part 0: Primer (Second Edition). W3C Recommendation 17 April 2007, W3C, 2007. [18]

[MMG$^+$07] Simon Moser, Axel Martens, Katharina Görlach, Wolfram Amme, and Artur Godlinski. Advanced Verification of Distributed WS-BPEL Business Processes Incorporating CSSA-based Data Flow Analysis. In *International Conference on Services Computing (SCC 2007)*, pages 98–105. IEEE Computer Society, 2007. [208]

[MRS05] Peter Massuthe, Wolfgang Reisig, and Karsten Schmidt. An Operating Guideline Approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics*, 1(3):35–43, 2005. [25, 35]

[MS05] Peter Massuthe and Karsten Schmidt. Operating Guidelines - an Automata-Theoretic Foundation for the Service-Oriented Architecture. In Kai-Yuan Cai, Atsushi Ohnishi, and M.F. Lau, editors, *International Conference on Quality Software (QSIC 2005)*, pages 452–457. IEEE Computer Society, September 2005. [50, 68]

[MSR06] Robi Malik, David Streader, and Steve Reeves. Conflicts and fair testing. *Int. J. Found. Comput. Sci.*, 17(4):797–814, 2006. [61, 64, 218, 231, 232, 234, 235, 236, 237, 244]

[MSSW08] Peter Massuthe, Alexander Serebrenik, Natalia Sidorova, and Karsten Wolf. Can I find a partner? Undecidablity of partner existence for open nets. *Information Processing Letters*, 108(6):374–378, November 2008. [47]

[MSV09] Arjan J. Mooij, Christian Stahl, and Marc Voorhoeve. Relating fair testing and accordance for service replaceability. *submitted to JLAP*, 2009. [26, 59]

[Mur89] Tadao Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989. [32, 49, 120, 221]

[MW07] Peter Massuthe and Karsten Wolf. An Algorithm for Matching Nondeterministic Services with Operating Guidelines. *International Journal of Business Process Integration and Management*, 2(2):81–90, 2007. [87, 88]

[MW08] Peter Massuthe and Daniela Weinberg. Fiona: A tool to analyze interacting open nets. In Niels Lohmann and Karsten Wolf, editors, *Workshop on Algorithms and Tools for Petri Nets (AWPN 2008)*, volume 380 of *CEUR Workshop Proceedings*, pages 99–104. CEUR-WS.org, September 2008. [28, 85, 138]

[NC95] V. Natarajan and Rance Cleaveland. Divergence and fair testing. In Zoltán Fülöp and Ferenc Gécseg, editors, *International Colloquium on Automata, Languages and Programming (ICALP 1995)*, volume 944 of *Lecture Notes in Computer Science*, pages 648–659. Springer, 1995. [63, 231, 233]

[NH84] Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theor. Comput. Sci.*, 34:83–133, 1984. [218, 231]

[Nic87] Rocco De Nicola. Extensional equivalences for transition systems. *Acta Inf.*, 24(2):211–237, 1987. [218, 249]

[NNH05] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis.* Springer, Berlin, 2nd edition, 2005. [55]

[OVA$^+$07] Chun Ouyang, Eric Verbeek, Wil M. P. van der Aalst, Stephan Breutel, Marlon Dumas, and Arthur H. M. ter Hofstede. Formal semantics and analysis of control flow in WS-BPEL. *Sci. Comput. Program.*, 67(2-3):162–198, 2007. [26, 55]

[Pap07] Mike P. Papazoglou. *Web Services: Principles and Technology.* Pearson - Prentice Hall, Essex, July 2007. [17, 18, 53]

[Pap08] Mike P. Papazoglou. The challenges of service evolution. In Zohra Bellahsene and Michel Léonard, editors, *International Conference on Advanced Information Systems Engineering (CAiSE 2008)*, volume 5074 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2008. [21]

[Par81] David Park. Concurrency and Automata on Infinite Sequences. In *GI-Conference on Theoretical Computer Science*, pages 167–183, London, UK, 1981. Springer. [31]

[PBH07] Jyotishman Pathak, Samik Basu, and Vasant Honavar. On Context-Specific Substitutability of Web Services. In *International Conference on Web Services (ICWS 2007)*, pages 192–199. IEEE Computer Society, 2007. [220]

[Pel93] Doron Peled. All from one, one for all: on model checking using representatives. In Costas Courcoubetis, editor, *International Conference on Computer Aided Verification (CAV 1993)*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 1993. [110]

[Pel03] Chris Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003. [19]

[Phi87] Iain Phillips. Refusal testing. *Theor. Comput. Sci.*, 50:241–284, 1987. [218]

[PTDL08] Mike P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: a research roadmap. *Int. J. Cooperative Inf. Syst.*, 17(2):223–255, 2008. [17, 20, 21]

[PV99] Antti Puhakka and Antti Valmari. Weakest-Congruence Results for Livelock-Preserving Equivalences. In Jos C. M. Baeten and Sjouke Mauw, editors, *International Conference on Concurrency Theory (CONCUR 1999)*, volume 1664 of *Lecture Notes in Computer Science*, pages 510–524. Springer, 1999. [117]

[Rei85] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985. [32]

[Ros98] A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall Series in Computer Science. Prentice Hall, 1998. [63, 218, 247]

[RRD04] Stefanie Rinderle, Manfred Reichert, and Peter Dadam. Correctness criteria for dynamic changes in workflow systems - a survey. *Data Knowl. Eng.*, 50(1):9–34, 2004. [229]

[RRMD09] Manfred Reichert, Stefanie Rinderle-Ma, and Peter Dadam. *Transactions on Petri Nets and Other Models of Concurrency II, Special Issue on Concurrency in Process-Aware Information Systems*, 2:115–135, 2009. [229]

[RV07] Arend Rensink and Walter Vogler. Fair testing. *Inf. Comput.*, 205(2):125–198, 2007. [59, 63, 231, 233]

[RW87] 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. [46]

[SCCS05] Natasha Sharygina, Sagar Chaki, Edmund M. Clarke, and Nishant Sinha. Dynamic Component Substitutability Analysis. In John Fitzgerald, Ian J. Hayes, and Andrzej Tarlecki, editors, *International Symposium of Formal Methods Europe (FM 2005)*, volume 3582 of *Lecture Notes in Computer Science*, pages 512–528. Springer, 2005. [55, 219]

[Sch05] Karsten Schmidt. Controllability of open workflow nets. In Jörg Desel and Ulrich Frank, editors, *Workshop on Enterprise Modelling and Information Systems Architectures (EMISA 2005)*, volume P-75 of *Lecture Notes in Informatics*, pages 236–249. GI, 2005. [46]

[SK07] Natasha Sharygina and Daniel Kröning. Model checking with abstraction for web services. In Luciano Baresi and Elisabetta Di Nitto, editors, *Test and Analysis of Web Services*, pages 121–145. 2007. [55]

[SM83] Ichiro Suzuki and Tadao Murata. A Method for Stepwise Refinement and Abstraction of Petri Nets. *J. Comput. Syst. Sci.*, 27(1):51–76, 1983. [221]

[SMB09] Christian Stahl, Peter Massuthe, and Jan Bretschneider. Deciding substitutability of services with operating guidelines. *Transactions on Petri Nets and Other Models of Concurrency II, Special Issue on Concurrency in Process-Aware Information Systems*, 2:172–191, 2009. [26, 27, 59, 61, 65, 133, 135, 155, 156]

[SW08] Christian Stahl and Karsten Wolf. Covering places and transitions in open nets. In Marlon Dumas and Manfred Reichert, editors, *International Conference on Business Process Management (BPM 2008)*, volume 5240 of *Lecture Notes in Computer Science*, pages 116–131. Springer, September 2008. [27, 89]

[SW09a] Christian Stahl and Karsten Wolf. Deciding service composition and substitutability using extended operating guidelines. *Data Knowl. Eng.*, 68(9):819–833, 2009. [27, 89, 133, 149]

[SW09b] Jan Sürmeli and Daniela Weinberg. Creating message profiles of open nets. In Oliver Kopp and Niels Lohmann, editors, *Workshop on Services and their Composition (ZEUS 2009)*, volume 438 of *CEUR Workshop Proceedings*, pages 74–80. CEUR-WS.org, 2009. [89]

[Szy98] Clemens Szyperski. *Component Software–Beyond Object-Oriented Programming.* Addison-Wesley and ACM Press, 1998. [17]

[Val79] Robert Valette. Analysis of Petri Nets by Stepwise Refinements. *J. Comput. Syst. Sci.*, 18(1):35–46, 1979. [221]

[Val91] Antti Valmari. A stubborn attack on state explosion. In Edmund M. Clarke and Robert P. Kurshan, editors, *International Workshop on Computer Aided Verification (CAV 1990)*, volume 531 of *Lecture Notes in Computer Science*, pages 156–165. Springer, 1991. [110]

[Val95] Antti Valmari. The Weakest Deadlock-Preserving Congruence. *Inf. Process. Lett.*, 53(6):341–346, 1995. [117]

[Val98] Antti Valmari. The state explosion problem. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528. Springer, 1998. [110]

[Vog92] Walter Vogler. *Modular Construction and Partial Order Semantics of Petri Nets*, volume 625 of *Lecture Notes in Computer Science.* Springer, 1992. [36, 59, 180, 215, 216, 221]

[WFMN04] A. Wombacher, P. Fankhauser, B. Mahleko, and E. J. Neuhold. Matchmaking for business processes based on choreographies. *International Journal of Web Service Research*, 1(4):14–32, 2004. [71]

[Wol09] Karsten Wolf. Does my service have partners? *Transactions on Petri Nets and Other Models of Concurrency II, Special Issue on Concurrency in Process-Aware Information Systems*, 2:152–171, 2009. [47, 101, 128, 129, 172, 227, 228]

[WSOD09] Karsten Wolf, Christian Stahl, Janine Ott, and Robert Danitz. Verifying Livelock Freedom in an SOA Scenario. In Stephen Edwards and Walter Vogler, editors, *International Conference on Application of Concurrency to System Design (ACSD'09)*, Augsburg, Germany, July 2009. IEEE Computer Society. [27, 99]

[ZBDH06] Johannes Maria Zaha, Alistair P. Barros, Marlon Dumas, and Arthur H. M. ter Hofstede. Let's Dance: A language for service behavior modeling. In *Proceedings of OTM Confederated International Conferences, Part I*, volume 4275 of *Lecture Notes in Computer Science*, pages 145–162, Montpellier, France, 29 October-3 November 2006. Springer. [19]

Index

Acknowledgements

It is now five and a half year ago when Wolfgang Reisig asked me whether I would like to work on a PhD in his group in Berlin. At this time, he had a vacant position on a project on design and verification of asynchronous hardware. So I became a PhD student. After two years, the funding of this project was not extended. So we worked on a new research proposal—this time on service substitution—and I had to bridge the time until the reviewers had decided about this proposal.

Thanks to the B.E.S.T. (Berlin-Rostock-Eindhoven Service Technology) cooperation I got the opportunity to work in the group of Wil van der Aalst and Kees van Hee in Eindhoven for four months. The time in Eindhoven was a great experience. I learned many things and could develop my personality. In particular, this time was the starting point for a successful collaboration from which I benefitted so much.

The proposal on service substitution got finally accepted. So after returning from Eindhoven to Berlin I started a PhD project on this topic. After two years of doing research on service substitution—from which I spent several months in Eindhoven—the funding stopped. However, this time I had enough results, and Wil van der Aalst made it possible to finance six additional months giving me the opportunity to finish my thesis.

I would not have finished this thesis without the help of various people. In the following I will express my gratitude to them. The first person I would like to thank is my copromotor Karsten Wolf. Most of the results of this thesis are joint work with him. When nobody in Berlin understood my (admittedly, often unordered) ideas, he did. I always enjoyed the hospitality in his group, and every time I was surprised how much progress we made within a few hours. Karsten also gave me a lot support during the time of writing this thesis.

Also, I would like to thank my promotors Kees van Hee and Wolfgang Reisig. I am grateful to Kees for supervising the writing of this thesis and for his enthusiasm for my work. Kees carefully read the first version of this thesis—at this stage an incredibly hard job—and gave valuable comments. I thank Wolfgang Reisig for offering me a position in his research group and for giving me free hand in my project. In addition, I thank him for initiating B.E.S.T. and for establishing the contact to Eindhoven.

Wil van der Aalst initiated the work on multiparty contracts and transformation rules. I always enjoyed the efficient way of working with him. I also thank Wil for his detailed comments and suggestions on this thesis. They helped to improve the presentation of this thesis. Furthermore, I want to express my gratitude to Wil for making it possible to hire me such that I could finish my thesis.

I am grateful to Jos Baeten, Joachim Fischer, Natalia Sidorova, and Walter Vogler for their service as committee members. Especially, I would like to thank Walter Vogler for carefully reading this thesis and for pointing out some flaws.

Within Berlin and Rostock, I would like to thank Niels Lohmann, Peter Massuthe, and Daniela Weinberg for their support during the last years. They read parts of this thesis and gave constructive suggestions for improvement. Special thanks go to Peter Massuthe who had the idea of writing a research proposal on service substitution and who spent much time with me in working it out. I am also grateful to Dirk Fahland and Andreas Glausch, my officemates in Berlin. The discussions with them and their complementary view on my work were very helpful. I specifically thank all developers of Fiona and related tools. Especially, I would like to thank the student assistants at my project, Robert Danitz, Leonard Kern and Janine Ott, for their implementation work. Without their work, it would have been hard to publish the results on operating guidelines. Christian Gierds gave me support to get the tools running and thanks to the effort of Dirk Fahland, I got some industrial service models which deserve the predicate challenging.

Within Eindhoven I had an intense cooperation with Arjan Mooij and Marc Voorhoeve. The work on relating conformance to preorders in process theory had never been possible without them. I learned a lot and enjoyed the discussions we had and not to forget the Dutch lessons. Particularly, I would like to express my gratitude to Arjan for his support during the last months. I am grateful to Natalia Sidorova, Eric Verbeek, and Jan Martijn van der Werf for fruitful discussions and the work on joint papers.

I also would like to thank all other colleagues of the “Kaffeerunde” in Berlin, of the AIS and the DH group in Eindhoven, and from Rostock for their discussions and hints concerning my work and for the pleasant and familiar atmosphere at these groups.

I am grateful to Simon Moser and Dieter König from IBM Böblingen for sharing their knowledge about BPEL and SOA. Their valuable feedback on my work helped me to apply my theoretical results to problems in practice. The work on extending the equivalence notion for BPEL is a result of this collaboration.

Many thanks I would like to address to the secretaries in Berlin and Eindhoven, Birgit Heene, Riet van Buul and Ine van der Ligt, for their help and support of so many administrative things.

Of course, this work would never be completed without constant support from my family and friends. I thank all my friends for their friendship and their understanding that I often also had to work in the evenings and at the weekends. I always enjoyed our joint social activities and holidays. Finally, I want to thank my family for their love, patience, and support during the last years even in times when we have not seen us for several months. Thank you for always being there when I need you!

Christian Stahl
Eindhoven, October, 2009

Erklärung

Ich erkläre hiermit, dass

- ich die vorliegende Dissertationsschrift
  "Service Substitution—A Behavioral Approach Based on Petri Nets"
  selbständig und ohne unerlaubte Hilfe angefertigt sowie nur die angegebene Literatur verwendet habe,
- ich mich nicht bereits anderwärts um einen Doktorgrad beworben habe oder ich einen solchen besitze und
- mir die Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät II der Humboldt-Universität zu Berlin (veröffentlicht im Amtlichen Mitteilungsblatt Nr. 34/2006) bekannt ist.

Christian Stahl
Eindhoven, den 9. Oktober 2009

Curriculum Vitae

Christian Stahl was born on June 15, 1978 in Berlin, Germany. From 1992 until 1998, he attended the grammar school "Gerhart-Hauptmann-Oberschule" in Berlin. After he finished the grammar school, he served 13 month his community service. In October 1999, Christian started his studies in Computer Science at Humboldt-Universität zu Berlin. In April 2004, he graduated under the supervision of Prof. Dr. Wolfgang Reisig. In his master thesis, Christian developed a Petri net semantics for the Business Process Execution Language for Web Services (nowadays known as WS-BPEL).

During his studies, Christian worked as a programmer at Aucoteam Berlin GmbH and as a student assistant at the Theory of Programming group headed by Prof. Dr. Wolfgang Reisig.

After graduation, Christian became a research associate at the Theory of Programming group, where he worked for two years on a project on modeling and verification of globally asynchronous locally synchronous (GALS) circuits. From August 2006 until November 2006, Christian joined the Architecture of Information Systems group, Technische Universiteit Eindhoven, The Netherlands headed by Prof. Dr. Wil van der Aalst and Prof. Dr. Kees van Hee. During his stay in Eindhoven, he worked on an SOA-based architecture framework.

In December 2006, Christian moved back to Berlin and started working on a project on service substitution at the Theory of Programming group in Berlin. In this time, Christian became a member of the B.E.S.T. (Berlin-Rostock-Eindhoven Service Technology) program and in particular of the dual PhD program. For that reason, his work was supervised by Prof. Dr. Kees van Hee, Prof. Dr. Wolfgang Reisig, and Prof. Dr. Karsten Wolf (University of Rostock). From December 2008 until July 2009, Christian moved again to Eindhoven, where he finished his thesis.

Since September 2009, Christian is working as a university researcher at the Architecture of Information Systems group, Technische Universiteit Eindhoven, The Netherlands. In his research, he is focused on modeling and analysis of service interaction.

SIKS Dissertatiereeks

====
1998
====

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations
within the Language/Action Perspective

1998-4 Dennis Breuker (UM)
Memory versus Search in Games

1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting

====
1999
====

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling;
Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets

1999-3 Don Beal (UM)
The Nature of Minimax Search

1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven
Specification of Network Information Systems

1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems

1999-7 David Spelt (UT)
Verification support for object database design

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent
Mechanism for Discrete Reallocation.

====
2000
====

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie;
een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval.

2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication

2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coup (EUR)
Sensitivity Analyis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations,
Algorithms and Architecture

2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

====
2001
====

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA)
Learning as problem solving

2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with
Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style

2001-6 Martijn van Welie (VU)
Task-based User Interface Design

2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization

2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.

2001-9 Pieter Jan 't Hoen (RUL)

Towards Distributed Development of Large Object-Oriented Models,
Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice
BRAHMS: a multiagent modeling and simulation language
for work practice analysis and design

2001-11 Tom M. van Engers (VUA)
Knowledge Management:
The Role of Mental Models in Business Systems Design

====
2002
====

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments
inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL)
Applied legal epistemology;
Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative
E-Commerce Ideas

2002-09 Willem-Jan van den Heuvel(KUB)
Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (Uva)
Processing XML in Database Systems

2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and
Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)

Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UVA)
Understanding, Modeling, and Improving Main-Memory Database Performance

====
2003
====

2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)
Repair Based Scheduling

2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction
between medium, innovation context and culture

2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to
Digital Media Warehouses

2003-17 David Jansen (UT)

Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
Learning Search Decisions

====
2004
====

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar
abstract denken, vooral voor meisjes

2004-08 Joop Verbeek(UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale
politiële gegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining

2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning

2004-17 Mark Winands (UM)
Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA)

Supporting the Construction of Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval

2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams

====
2005
====

2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications

2005-02 Erik van der Werf (UM))
AI techniques for the game of Go

2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language

2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data

2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing

2005-06 Pieter Spronck (UM)
Adaptive Game AI

2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications

2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages

2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments

2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry

2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen

2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics

2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU)
Usability of XML Query Languages

2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components

2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks

2005-19 Michel van Dartel (UM)
Situated Representation

2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives

2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by
Exploiting Application Semantics

====
2006
====

2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting

2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations

2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems

2006-04 Marta Sabou (VU)
Building Web Service Ontologies

2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines

2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools
for Graphical Service Modeling

2006-07 Marko Smiljanic (UT)
XML schema matching – balancing efficiency and effectiveness by means of clustering

2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web

2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion

2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems

2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types

2006-12 Bert Bongers (VU)
Interactivation - Towards an e-cology of people, our technological environment, and the arts

2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents

2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements
Change

2006-15 Rainer Malik (UU)

CONAN: Text Mining in the Biomedical Domain

2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks

2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device

2006-18 Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing

2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach

2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining

2006-21 Bas van Gils (RUN)
Aptness on the Web

2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation

2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web

2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources

2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC

2006-26 Vojkan Mihajlovic (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval

2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories

2006-28 Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval

====
2007
====

2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures

2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach

2007-03 Peter Mika (VU)
Social Networks and the Semantic Web

2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach

2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance

2007-06 Gilad Mishne (UVA)
Applied Text Analytics for Blogs

2007-07 Natasa Jovanovic' (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings

2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations

2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation

2007-10 Huib Aldewereld (UU)
Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols

2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System

2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty

2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology

2007-14 Niek Bergboer (UM)
Context-Based Image Analysis

2007-15 Joyca Lacroix (UM)
NIM: a Situated Computational Memory Model

2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems

2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice

2007-18 Bart Orriens (UvT)
On the development an management of adaptive business collaborations

2007-19 David Levy (UM)
Intimate relationships with artificial partners

2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network

2007-21 Karianne Vermaas (UU)
Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005

2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns

2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems

2007-24 Georgina Ramrez Camps (CWI)
Structural Features in XML Retrieval

2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement

====
2008
====

2008-01 Katalin Boer-Sorbn (EUR)
Agent-Based Simulation of Financial Markets: A modular,continuous-time approach

2008-02 Alexei Sharpanskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations

2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach

2008-04 Ander de Keijzer (UT)
Management of Uncertain Data - towards unattended integration

2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective

2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines,
an Artificial Intelligence Perspective

2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning

2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference

2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective

2008-10 Wauter Bosma (UT)
Discourse oriented summarization

2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach

2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation

2008-13 Caterina Carraciolo (UVA)
Topic Driven Access to Scientific Handbooks

2008-14 Arthur van Bunningen (UT)
Context-Aware Querying; Better Answers with Less Effort

2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains.

2008-16 Henriette van Vugt (VU)
Embodied agents from a user's perspective

2008-17 Martin Op 't Land (TUD)
Applying Architecture and Ontology to the Splitting and Allying of Enterprises

2008-18 Guido de Croon (UM)
Adaptive Active Vision

2008-19 Henning Rode (UT)
From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search

2008-20 Rex Arendsen (UVA)
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven

2008-21 Krisztian Balog (UVA)
People Search in the Enterprise

2008-22 Henk Koning (UU)
Communication of IT-Architecture

2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia

2008-24 Zharko Aleksovski (VU)
Using background knowledge in ontology matching

2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency

2008-26 Marijn Huijbregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled

2008-27 Hubert Vogten (OU)
Design and Implementation Strategies for IMS Learning Design

2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks

2008-29 Dennis Reidsma (UT)
Annotations and Subjective Machines -
Of Annotators, Embodied Agents, Users, and Other Humans

2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis:
Techniques for Extracting, Representing and Querying Media Content

2008-31 Loes Braun (UM)
Pro-Active Medical Information Retrieval

2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes

2008-33 Frank Terpstra (UVA)
Scientific Workflow Design; theoretical and practical issues

2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining

2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure

====
2009
====

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU)
Understanding Classification

2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Massuthe (TUE, Humboldt-Universität zu Berlin)
Operating Guidelines for Services

2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems

2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies
(making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess

2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data

2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System

2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in
Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI)
"RAM: Array Database Management through Relational Mapping"

2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services

2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models

2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage

2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text

2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors

2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method
Engineering Approach

2009-35 Wouter Koelewijn (UL)
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling

2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks

2009-37 Hendrik Drachsler (OUN)
Navigation Support for Learners in Informal Learning Networks

2009-38 Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a
multilingual context