

Verifying responsiveness for open systems by means of conformance checking

Citation for published version (APA):

Müller, R. (2014). *Verifying responsiveness for open systems by means of conformance checking*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science]. Technische Universiteit Eindhoven. <https://doi.org/10.6100/IR780046>

DOI:

[10.6100/IR780046](https://doi.org/10.6100/IR780046)

Document status and date:

Published: 01/01/2014

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

VERIFYING RESPONSIVENESS FOR OPEN SYSTEMS BY MEANS OF CONFORMANCE CHECKING

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
donderdag 28 augustus 2014 om 16.00 uur

door

Richard Müller

geboren te Dresden, Duitsland

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter: prof.dr. E.H.L. Aarts
1^e promotor: prof.dr.ir. W.M.P. van der Aalst
2^e promotor: Prof. Dr. W. Reisig (Humboldt-Universität zu Berlin)
copromotor: Dr. C. Stahl
leden: Prof. J.-C. Freytag, Ph.D. (Humboldt-Universität zu Berlin)
prof.dr.ir. J.F. Groote
Prof. Dr.-Ing. U. Nestmann (Technische Universität Berlin)
Prof. Dr. W. Vogler (Universität Augsburg)

VERIFYING RESPONSIVENESS FOR OPEN SYSTEMS BY MEANS OF CONFORMANCE CHECKING

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
Humboldt-Universität zu Berlin
im Rahmen einer Binationalen Promotion mit der
Technischen Universität Eindhoven, Niederlande

von
Herrn Diplom-Informatiker

Richard Müller

geboren am 18. Oktober 1984 in Dresden

Präsident der Humboldt-Universität zu Berlin
Prof. Dr. Jan-Hendrik Olbertz

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät II
Prof. Dr. Elmar Kulke

1. Gutachter	Prof. Dr. Wolfgang Reisig
2. Gutachter	prof.dr.ir. Wil M.P. van der Aalst
3. Gutachter	Prof. Dr. Walter Vogler

eingereicht am 09. April 2014

Tag der mündlichen Prüfung 28. August 2014

VERIFYING RESPONSIVENESS FOR OPEN SYSTEMS BY MEANS OF CONFORMANCE CHECKING

ABSTRACT

Best engineering practices suggest *specifying* a system before actually *implementing* it. Both the implementation as well as its specification exhibit *behavioral properties*. *Conformance checking* is deciding whether the implementation of a system preserves a certain behavioral property of its specification. This is the central scientific problem of this thesis.

Over the past years, there has been a shift in systems engineering from monolithic, closed systems to distributed systems, composed of *open systems*. Therefore, our research centers around conformance checking for open systems. An open system interacts with other open systems—that is, its *environment*. Of particular interest are *responsive* environments with which interaction or mutual termination is always possible. We refer to such an environment as a *partner*. For an open system, conformance checking translates to deciding whether each partner of its specification is a partner of the implementation.

We consider conformance checking for open systems in two distinct scenarios. In the first scenario, the *model-model scenario*, we assume the specification and the implementation of an open system to be given as formal models. We characterize conformance for two variants of responsiveness. For the first variant, conformance turns out to be undecidable. For the second variant however, we develop a decision algorithm for conformance and a finite characterization of all conforming open systems. In addition, two open systems can be composed, yielding again an (open) system. In general, we require conformance to respect *compositionality*; that is, we wish to infer the conformance of a composition from the conformance of the composed open systems. Therefore, we also study the above mentioned compositionality property of conformance for the two variants of responsiveness, and show its (un-)decidability.

In the second scenario, the *log-model scenario*, we assume the specification of an open system to be given as a formal model, but this time no formal model of the implementation is available. However, most implementations record their actual behavior. The observed behavior of an implementation can be recorded in an *event log*. This is a more realistic and practically relevant assumption because the implementation is often too complex to be formally modeled. The idea is to use an event log to check conformance of the unknown implementation to its known specification. To this end, we present a necessary condition for conformance: We analyze whether there exists a conforming implementation which can produce the event log. Furthermore, we study whether we can discover a formal model of the unknown implementation from the event log, assuming the implementation conforms to its specification.

We implement the decision algorithm from the first scenario and use it to develop algorithms for both questions in the second scenario. We evaluate the implemented algorithms using industrial-sized specifications and event logs.

KURZFASSUNG

Es ist gute ingenieurwissenschaftliche Praxis, ein System vor seiner Implementierung zu spezifizieren. Sowohl die *Implementierung* als auch die *Spezifikation* eines Systems zeigen *Verhaltenseigenschaften*. Eine Implementierung eines Systems ist *konform* zu einer Spezifikation, wenn die Implementierung bestimmte Verhaltenseigenschaften der Spezifikation bewahrt. Diese Konformanz zu entscheiden ist die zentrale wissenschaftliche Fragestellung dieser Arbeit.

In der Systementwicklung hat es in den letzten Jahren eine Verschiebung weg von monolithischen, geschlossenen Systemen hin zu verteilten, *offenen Systemen* gegeben. Daher beschäftigen wir uns in dieser Arbeit mit der Konformanz offener Systeme. Ein offenes System interagiert mit anderen offenen Systemen, d.h. mit seiner *Umgebung*. Von besonderem Interesse sind hierbei *responsive* Umgebungen, mit welchen Interaktion oder gemeinsame Terminierung immer möglich ist. Solch eine responsive Umgebung ist ein *Partner* eines offenen Systems. Die Frage nach der Konformanz eines offenen Systems lässt sich damit in die Frage übersetzen, ob jeder Partner der Spezifikation auch ein Partner der Implementierung ist.

In dieser Arbeit betrachten wir die Konformanz offener Systeme in zwei unterschiedlichen Szenarien. Im ersten Szenario, dem *Modell-Modell-Szenario*, ist sowohl die Spezifikation als auch die Implementierung eines offenen Systems als formales Modell gegeben. Wir charakterisieren Konformanz für zwei Varianten von Responsivität und zeigen deren (Un-)Entscheidbarkeit. Im Fall der Entscheidbarkeit entwickeln wir einen Entscheidungsalgorithmus und eine endliche Charakterisierung aller konformen offenen Systeme. Die Komposition zweier offener Systeme ist wieder ein offenes System. Eine wünschenswerte Eigenschaft von Konformanz ist daher *Kompositionalität*, d.h. wir wollen von der Konformanz der komponierten offenen Systeme auf die Konformanz der Komposition schließen. Dementsprechend untersuchen wir Konformanz für beide Varianten von Responsivität auch auf Kompositionalität und zeigen deren (Un-)Entscheidbarkeit.

Im zweiten Szenario, dem *Log-Modell Szenario*, ist nur die Spezifikation eines offenen Systems als formales Modell gegeben; ein formales Modell der Implementierung ist nicht bekannt. Jedoch stellen die meisten Implementierungen beobachtetes Verhalten von sich in Form eines *Logs* zur Verfügung. Diese Annahme ist weitaus realistischer und praktisch relevanter als das Modell-Modell-Szenario, da die Implementierung eines offenen Systems in der Regel zu komplex ist um formal modelliert zu werden. Die Idee ist nun, mit Hilfe des Logs auf die Konformanz der unbekanntes Implementierung und der bekannten Spezifikation zu schließen. Zu diesem Zweck präsentieren wir in dieser Arbeit eine notwendige Bedingung für Konformanz: Wir entscheiden, ob eine konforme Implementierung existiert, welche das gegebene Log erzeugen kann. Darüber hinaus entwickeln wir eine Methode zum automatischen Erstellen eines formalen Modells der unbekanntes Implementierung aus dem gegebenen Log unter der Annahme, dass die Implementierung konform zur Spezifikation ist.

Wir implementieren den Entscheidungsalgorithmus aus dem ersten Szenario und verwenden ihn, um Algorithmen für beide Fragen im zweiten Szenario zu entwickeln. Wir werten die implementierten Algorithmen mit industrienahen Spezifikationen und Logs aus.

CONTENTS

I	INTRODUCTION	1
1	ABOUT THIS THESIS	3
1.1	Background	3
1.1.1	Designing correct systems using conformance checking	3
1.1.2	Conformance checking for open systems	5
1.2	Problem statement and research questions	8
1.3	Contributions	11
1.4	Thesis overview	15
2	PRELIMINARIES	17
2.1	Basic mathematical notions	17
2.2	Labeled transition systems	18
2.3	Petri nets	21
2.4	Open nets and their composition	24
2.5	Open net environments and their composition	27
2.6	Relating open nets and open net environments	33
2.7	Conclusions and related work	35
2.7.1	Formalism based on process algebras	36
2.7.2	Formalism based on automata	37
2.7.3	Formalism based on Petri nets	37
2.7.4	Why do we chose open nets?	37
3	RESPONSIVENESS FOR OPEN SYSTEMS	39
3.1	Formalizing responsiveness and conformance	40
3.2	Formalizing b -responsiveness and b -conformance	44
3.3	Classifying both formalizations	49
3.3.1	Classifying responsiveness and b -responsiveness	49
3.3.2	Comparing conformance and b -conformance	50
3.4	Conclusions and related work	51
II	THE MODEL-MODEL SCENARIO	53
4	CONFORMANCE AND COMPOSITIONAL CONFORMANCE	55
4.1	Characterizing conformance	57
4.1.1	The <i>stopdead</i> -semantics for open nets	57
4.1.2	Refinement on the <i>stopdead</i> -semantics	61
4.2	Characterizing compositional conformance	64
4.2.1	The \mathcal{F}_{fin}^+ -semantics for open nets	65
4.2.2	Refinement on the \mathcal{F}_{fin}^+ -semantics	68
4.3	Undecidability of conformance and compositional conformance	75
4.3.1	Counter machines and their halting problem	76
4.3.2	Conformance is undecidable	79
4.3.3	Compositional conformance is undecidable	82
4.4	Conclusions	83
5	b -CONFORMANCE	85
5.1	Characterizing b -conformance	85
5.1.1	The b -bounded <i>stopdead</i> -semantics for open nets	86
5.1.2	The b -coverable <i>stopdead</i> -semantics for open nets	92
5.1.3	Refinement on the b -coverable <i>stopdead</i> -semantics	97

5.2	Deciding b -conformance	98
5.2.1	Deciding b -responsiveness using the LTS BSD_b	99
5.2.2	Deciding b -conformance using the LTS CSD_b	105
5.2.3	Analyzing the computational complexity	117
5.3	An alternative decision procedure for b -conformance	121
5.3.1	Deciding b -responsiveness using matching	123
5.3.2	Deciding b -conformance using matching	127
5.3.3	Analyzing the computational complexity	132
5.4	Implementation and experimental results	135
5.5	Conclusions	139
6	COMPOSITIONAL b -CONFORMANCE	141
6.1	Characterizing compositional b -conformance	141
6.1.1	The b -bounded \mathcal{F}_{fin}^+ -semantics for open nets	142
6.1.2	Refinement on the b -bounded \mathcal{F}_{fin}^+ -refinement	145
6.2	Deciding compositional b -conformance	149
6.2.1	Deciding \mathcal{F}_{fin}^+ -refinement for finite LTSs	150
6.2.2	Reducing the decision of compositional b -conformance to \mathcal{F}_{fin}^+ -refinement	153
6.3	Conclusions	157
7	CONCLUSIONS AND RELATED WORK	159
7.1	Overview of the results	159
7.2	Classifying compositional conformance and compositional b -conformance	160
7.3	Related work	162
7.3.1	Work based on process algebra and declarative models	162
7.3.2	Work based on automata	164
7.3.3	Work based on Petri nets	165
7.3.4	Work related to the undecidability results	167
III	THE LOG-MODEL SCENARIO	169
8	TESTING FOR b -CONFORMANCE	171
8.1	Formalizing observed behavior	173
8.1.1	Events, event traces, and event logs	173
8.1.2	Replaying an event log on a labeled net	174
8.1.3	Replaying an event log on an open net	177
8.2	The testing procedure	179
8.3	Implementation	183
8.4	Evaluation and experimental results	183
8.4.1	Preparing the evaluation process	184
8.4.2	Testing 1-conforming implementations	185
8.4.3	Testing non 1-conforming implementations	187
8.5	Conclusions	191
9	DISCOVERING A MODEL OF A b -CONFORMING SYSTEM	193
9.1	The discovery procedure	194
9.1.1	Discovering a b -conforming open net	194
9.1.2	Discovering a high-quality open net	194
9.2	Improving the discovery procedure with b -subnets	203
9.2.1	Impact on the fitness dimension	204
9.2.2	Impact on the simplicity dimension	205
9.2.3	Impact on the precision dimension	206
9.2.4	Impact on the generalization dimension	207
9.3	Implementation	208
9.4	Evaluation and experimental results	212

9.4.1	Preparing the evaluation process	212
9.4.2	Discovering 1-conforming open nets	215
9.5	Conclusions	221
10	CONCLUSIONS AND RELATED WORK	223
10.1	Overview of the results	223
10.2	Work related to conformance testing	224
10.3	Work related to open system discovery	225
IV	CLOSURE	227
11	APPLYING THE THESIS RESULTS	229
11.1	The emergency ward service in a stroke unit	229
11.1.1	An informal specification	229
11.1.2	A formal model of the specification	232
11.1.3	Two implementations in WS-BPEL	232
11.2	The model-model scenario	237
11.2.1	Step 1: Deriving formal models	237
11.2.2	Step 2: Checking for 1-conformance	241
11.3	The log-model scenario	241
11.3.1	Step 1: Deriving event logs	242
11.3.2	Step 2: Testing for 1-conformance	243
11.3.3	Step 3: Discovering a high-quality model of a 1-conforming implementation	245
11.4	Conclusions	246
12	THESIS CONCLUSIONS AND OUTLOOK	249
12.1	Summary of contributions	249
12.1.1	The model-model scenario	249
12.1.2	The log-model scenario	250
12.1.3	Tool support	251
12.2	Limitations and open questions	253
12.2.1	Incomplete or unsound specifications	253
12.2.2	Measuring quality is subjective	253
12.2.3	Abstraction only preserves fitness and simplicity	254
12.3	Future work	254
12.3.1	Refined conformance relations	254
12.3.2	Improved algorithms	255
12.3.3	Compositionality in the log-model scenario	255
12.3.4	Refined discovery	256
12.3.5	Introducing additional aspects	256
	BIBLIOGRAPHY	257
	INDEX	279
	ACKNOWLEDGEMENTS	283
	CURRICULUM VITÆ	285
	DECLARATION	289

Part I

INTRODUCTION

ABOUT THIS THESIS

THIS thesis contributes to a general theory of open systems. Open systems are, as opposed to closed systems, inherently made to interact with each other. Our goal is to verify responsiveness for open systems—that is, to ensure that mutual termination or interaction between two open systems is always possible. In this thesis, we aim to verify responsiveness for open systems by means of conformance checking: A conformance relation for responsiveness is a relation between two systems—based on their systems’ models—that preserves responsiveness; conformance checking is deciding whether this conformance relation for responsiveness holds.

We introduce the background of conformance checking and responsiveness for open systems in Sect. 1.1. In Sect. 1.2, we formulate our problem statement and derive five research questions. We give an overview over our contributions in Sect. 1.3 and conclude this chapter with an outline of the forthcoming chapters in Sect. 1.4.

1.1 BACKGROUND

In Sect. 1.1.1, we discuss conformance checking as a verification method for the behavior of systems. We reminisce the rise of interacting open systems from closed, monolithic systems in systems engineering in Sect. 1.1.2.

1.1.1 *Designing correct systems using conformance checking*

“Our civilization runs on software”, as Bjarne Stroustrup once said [229]. Software systems are everywhere nowadays, but only a small part of them is visible—for example, the operating system of a personal computer. Most software systems are omnipresent in form of embedded software [145], which is invisibly woven into artifacts like vehicles, communication systems, banking systems, consumer electronics, household applications, and medical systems. A failure in those software systems may have profound consequences. As an example, several cancer patients received deadly radiation overdoses between 1985 and 1987 at oncology clinics in the U.S. and Canada because of a software error in a certain linear accelerator model [262]. Another example is the August 2003 blackout in the northeastern U.S. and Ontario. This blackout was partly caused by a software error in General Electric’s XA/21 energy management system and is associated with costs between seven and ten billion U.S. dollar [262]. Beside such fatalities, software failures cause enormous economic cost associated with developing and distributing software patches, reinstalling and substituting systems, and lost productivity. A cost analysis from 2008 [197] estimates software bugs to cost alone the U.S. economy 59.5 billion U.S. dollar a year. In addition, the impact of faulty software systems is likely to rise, as the volume of embedded software systems is still increasing at 10 to 20 percent every year [88]. That is why there is a vital interest in the development of *correctly behaving* software systems. Thereby, correct behavior of a system refers to the absence of unwanted states like deadlocks or livelocks, for example.

Developing correctly behaving software systems is a complex and error-prone task [138]. Software systems are one of the most complex artifacts produced by humans [147]. Modern operating systems consists of hundred of millions lines of code (LOC)—for example, 45 Mio. LOC for Windows XP [176], 86 Mio. LOC for Mac OS 10.5 (Leopard) [127], and 323 Mio. LOC for Debian 5.0 (Lenny) [73]. Even embedded software systems, although specialized for particular hardware and not as universal as operating systems, reach these complexity dimensions: For example, a modern car is driven by software systems with up to 100 Mio. LOC [60] and the software systems of an Airbus A380 consist of several 100 Mio. LOC [255]. Lines of code are longly overhauled as a precise measure of software complexity [94]. Nevertheless, these numbers still give an impression of the complexity of software systems.

Because of the complexity of software systems, there exists a long-standing interest in techniques for checking the correctness of systems; already the computing pioneers Turing, Goldstine, and von Neumann presented workable methods on systems correctness [183]. In the sixties, a broad set of *formal methods* for correct systems has been established. *Testing* aims to identify errors by exercising tests on the running implementation [240]. However, testing can only show the presence of errors, but never their absence. *Verification* aims to prove the correctness of a system—based on a mathematical model of that system—with mathematical certainty [120]. Today, with the dissemination of embedded systems into our daily lives, the interest in computer-aided verification methods is as strong as ever. Consequently, their usage in industrial projects increased significantly in recent years [221, 260]. In this thesis, we investigate *conformance checking* [68] which can be seen as a particular verification method.

Conformance checking centers around a *relation* \sqsubseteq between two system models. Usually, a detailed model *Impl*—the *implementation*—and a more abstract model *Spec*—the *specification*—of the behavior of a system is assumed to be given. It is then checked whether *Impl* is behaviorally “contained” in *Spec*—that is, whether a certain behavioral correctness criterion ϕ that holds in the specification also holds in the implementation. In this thesis, we refer to \sqsubseteq as *conformance* relation that preserves ϕ . Other names found in literature are refinement relation [256, 37], implementation relation [119, 72, 143, 238], conformation relation [81], preorder relation [68], accordance relation [226, 11], and subcontract relation [141, 44]. A conformance relation \sqsubseteq is usually formalized as a preorder, defining an ordering among system models. If two system models *Impl* and *Spec* are related by a conformance relation \sqsubseteq , then intuitively *Impl* is less abstract than *Spec*. In other words, *Impl* implements *Spec* in a certain sense, for example, by resolving some nondeterminism of *Spec*.

In system design, one can distinguish two approaches for designing correctly behaving systems: *correctness-by-construction* (i.e., a priori verification) and *correctness-by-verification* (i.e., a posteriori verification). Correctness-by-construction is a top-down approach. An implementation is constructed from a correct specification, and correctness of the implementation follows from the construction algorithm. Complementary to correctness-by-construction, correctness-by-verification is a bottom-up approach, where we conclude correctness of a given implementation by verifying the correctness of a specification. A conformance relation \sqsubseteq enables *both* approaches:

- In the top-down approach, we are given a behavioral correctness criterion ϕ and a specification *Spec* that satisfies ϕ . The goal is to construct

a correct implementation $Impl$ —that is, $Impl$ satisfies ϕ , too. To this end, we incrementally transform $Spec \sqsupseteq Impl_1 \sqsupseteq Impl_2 \sqsupseteq \dots \sqsupseteq Impl_k$ such that the conformance relation is preserved in each step. As a result, the implementation $Impl = Impl_k$ is correct by construction. This approach is also known as stepwise refinement [256], which is one of the main methods for the systematic construction of programs and systems [24].

- In the bottom-up approach, we are given a behavioral correctness criterion ϕ and an implementation $Impl$. The goal is to verify whether ϕ holds in $Impl$. To this end, we construct an abstraction $Spec$ from $Impl$ leaving out unnecessary details. Due to its more compact representation, $Spec$ is easier to verify than $Impl$. We can do so by applying verification techniques such as theorem proving [158, 83] and model checking [67, 28]. After establishing correctness of $Spec$, we can check whether $Impl$ is correct by checking whether $Impl \sqsubseteq Spec$.

Summing up, there exists a need for designing correctly behaving software systems. We can support their design with verification methods like conformance checking.

1.1.2 Conformance checking for open systems

We already detailed in Sect. 1.1.1 that software systems are highly complex. For handling this complexity, the principle of *modularization* is one of the most fundamental principles in system engineering. Thereby, we construct a large system by assembling smaller, interchangeable *parts* [169, 161]. *Compositionality* is one of the most desirable requirements for these interchangeable parts: An aggregate of properly assembled parts should behave as one part itself. Over the last forty years, parts have been developed in different forms like procedures, functions, modules [75], objects [175], components [230], and services [201]. They all differ in many types of properties like how they are assembled, whether they are stateless or not, or what kind of side effects they have. However, all these forms of parts share a common idea: Constructing a complex system by assembling less complex parts enables the reuse of existing parts and an “organic” growth of systems—that is, system evolution by redesigning and iteratively improving parts.

Consequently, there has been a constant shift in system engineering from monolithic, closed systems to distributed, *open systems*: Nowadays, typical complex systems are open and embedded in or connected with other open systems. These open systems execute concurrently on different machines and *interact* with each other through computer networks [22]. In other words, many complex systems have—despite their names—quite simple microscopic parts, and their complexity arises from local interactions [98]. Examples for such systems are service-oriented systems like web service applications [201], systems based on wireless network technologies like wireless sensor networks [17], online games [146], distributed transportation systems [117], medical systems [107], or a software system based on electronic control units in a car or plane [60]. Figure 1 motivates our choice to focus on conformance checking for open systems in this thesis.

The goal of this thesis is to contribute to a general theory of open systems. For open systems, *interaction* is a first-class citizen: A typical open system is not executed in isolation but interacts with other open systems—that is, its *environment*. An open system interacts with its environment through a

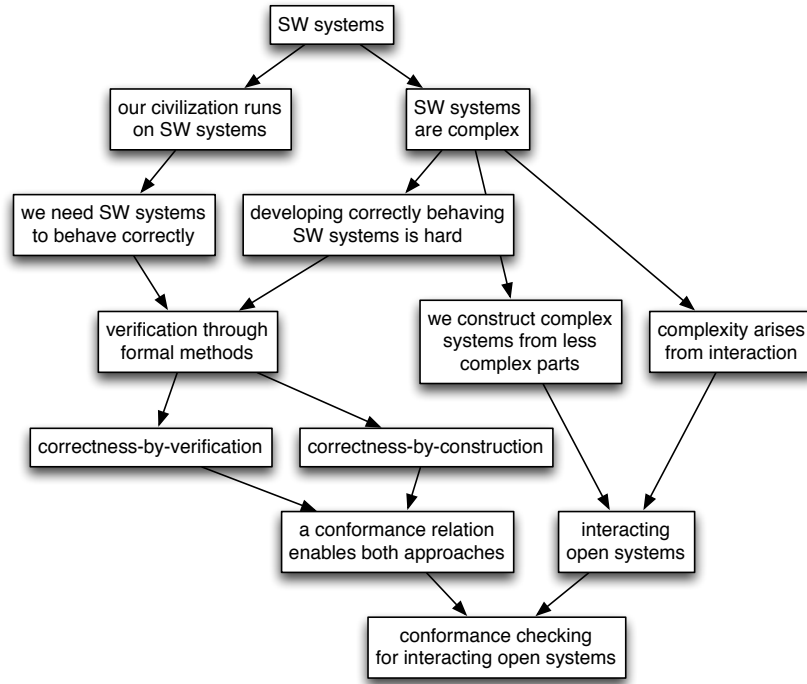


Figure 1: A summary of the arguments presented in Sect. 1.1.1 and Sect. 1.1.2 that motivate our study of conformance checking for interacting open systems.

well-defined *interface*. *Message-based communication* [29, 172, 55] emerged as a fundamental interaction mechanism, where open systems interact with each other by sending and receiving messages over the *message channels* of the interface. In this thesis, we study *asynchronous* communication: Each message channel is an unbounded, unordered, and lossless buffer [172, 134, 150]. We consider asynchronous communication because it naturally supports the distributed setting of interacting open systems [172]. On the downside, asynchronous communication between systems is more difficult to verify than synchronous communication. Every (complex) communicating open system has a *communication protocol* [174, 37] that describes the system's control flow (i.e., the order in which messages are exchanged) and the underlying communication model (i.e., the way messages are exchanged with the environment). The communication protocol of an asynchronously communicating open system is formulated in terms of visible actions (i.e., sending or receiving a message) and invisible actions (i.e., internal activities). In this thesis, we restrict ourselves to the communication protocol of an open system and abstract from details such as the location of the open system, the underlying middleware, or the content of messages. Figure 2 illustrates the open systems notion.

Next to interaction, *compositionality* is another first-class citizen for open systems: The composition of two open systems is again an open system. Composition allows open systems to be composed from smaller ones. Basic forms of composition are, for example, parallel composition, sequential composition, and recursion. In this thesis, we consider parallel composition with asynchronous communication because we consider it to be the natural form of composition for open systems.

Compositionality of open systems requires a compositional notion of conformance. A conformance relation is compositional if conformance between

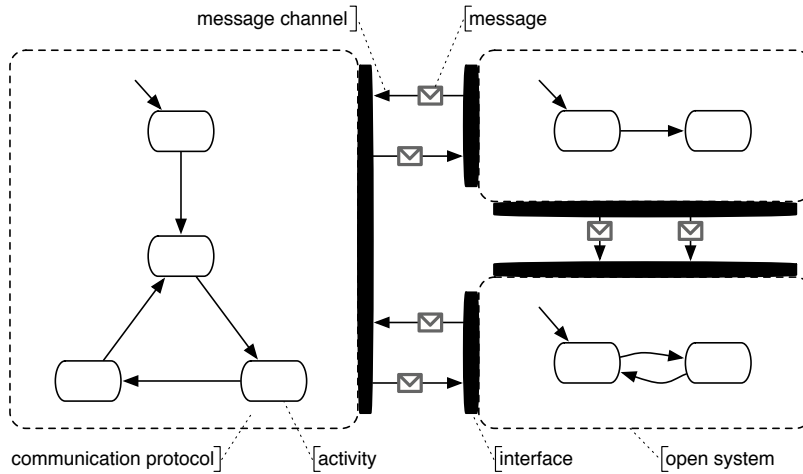


Figure 2: An illustration of the concepts of open systems.

two composed systems can be derived by showing conformance for their components. We illustrate the difference between a conformance relation for open systems and a conformance relation for closed systems using Fig. 3. For closed systems, there does not exist a notion of system composition. Hence, the conformance relation is a relation between two closed systems as depicted in Fig. 3a. For open systems, the conformance relation is a relation between two open systems. The implementation on the left-hand side in Fig. 3b is composed from three open systems $Impl_1$, $Impl_2$, and $Impl_3$. The specification on the right-hand side in Fig. 3b is composed from three open systems $Spec_1$, $Spec_2$, and $Spec_3$. Using a compositional conformance relation, we can infer that the implementation conforms to the specification from $Impl_1$ conforming to $Spec_1$, $Impl_2$ conforming to $Spec_2$, and $Impl_3$ conforming to $Spec_3$. Traditionally, compositional conformance notions for concurrent systems are considered hard to obtain [51].

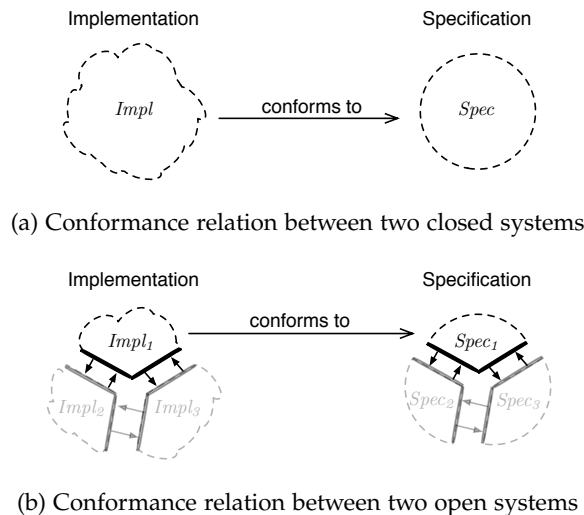


Figure 3: An illustration of the difference between a conformance relation for closed systems and a conformance relation for open systems.

The composition of two open systems may be correct in terms of its syntax, its semantics, its behavior, and its quality. *Syntactical correctness* ensures that

connected message channels have the same message type. *Semantical correctness* guarantees that messages and their content are correctly interpreted. *Behavioral correctness* expresses the absence of behavioral errors—that is, the absence of unwanted communication patterns. *Qualitative correctness* ensures quality parameters like reliability, costs, or security levels. As already motivated in Sect. 1.1.1, we restrict ourselves to the verification of *behavioral correctness*.

1.2 PROBLEM STATEMENT AND RESEARCH QUESTIONS

In this thesis, we aim to verify behavioral correctness of open systems by means of conformance checking. Thereby, the employed conformance relation always depends on a certain behavioral correctness criterion. In the following, we motivate *responsiveness* as a minimal behavioral correctness criterion for open systems.

Responsiveness ensures that *termination* of the composition of two interacting open systems or *communication* between these two open systems is always possible. In other words, responsiveness combines termination with interaction: Termination is an important correctness criterion to all kinds of systems, but usually too strict if considered in isolation. Interaction is fundamental to open systems. A nonterminating composition of two open systems that do not have the possibility to communicate is fundamentally ill-designed. An example for the importance of responsiveness is Microsoft’s asynchronous event driven programming language P [76]. P was used to implement and verify the core of the USB device driver stack that ships with Microsoft Windows 8. Thereby, P uses responsiveness for bounded message channels as a combination of termination and interaction while additionally requiring that no message in any channel is ignored forever. We aim at an even more general notion of responsiveness by focusing solely on the combination of termination and interaction. So the problem statement of this thesis is:

How can we verify responsiveness in open systems by means of conformance checking?

In the following, we state five research questions that arise from our problem statement.

For the first research question, recall that conformance checking operates on *models* of the behavior of open systems. Figure 4 illustrates the relation between open systems in reality and their models.

In this thesis, we assume that we can translate a given specification and implementation of an open system into formal models. We do not investigate how these models are derived; they may be created manually or automatically derived using existing techniques. Moreover, we focus on formal behavioral models that abstract from non-controlflow related aspects such as resource or timing information. The formal model of our choice will be *open nets* [246, 153]—a variant of Petri nets [216]—which we present and justify in Chap. 2. Figure 5 illustrates the relation between open systems’ behavior in reality and the translation of open systems’ behavior into open nets. As an example, consider an open system in the form of a service [201]. Services are often implemented in the Web Services Business Process Execution Language [130] (WS-BPEL). Those implementations can be translated into an open net using the compiler BPEL2OWFN [149]. In addition,

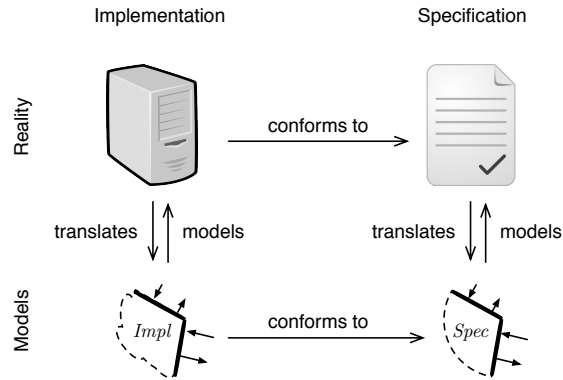


Figure 4: Verification using conformance checking operates on behavioral models of open systems.

there exist approaches that can translate a service description in PHP [208] or C [135] into an automata [223, 222] using techniques from the areas of model checking [67, 28] and static program analysis [198]. Automata, in turn, can be translated into Petri nets [25], e.g., using state-based [77, 215] or language-based regions [157].

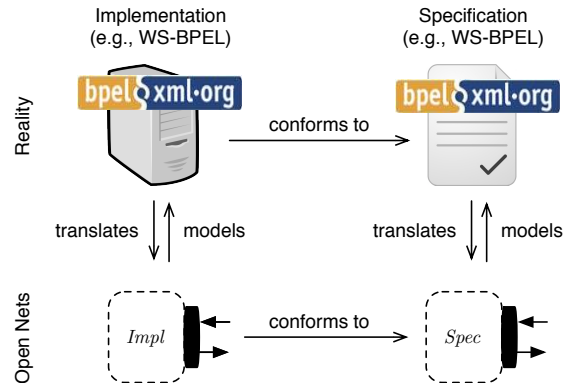


Figure 5: Employing existing translations of open systems' behavior to formal models.

Conformance checking on formal models requires a formal notion of responsiveness and of the corresponding conformance relation, both formulated in terms of these models. Therefore, our first research question is

1. How can we formalize responsiveness and the corresponding conformance relation on open nets?

For the remaining four research questions, we distinguish two scenarios for conformance checking, depending on the information we have about the implementation: In the first scenario, we assume that a formal model of the implementation is available. We refer to this scenario as the *model-model scenario*. In the second scenario, however, a formal model of the implementation is unavailable; instead, we are given an *event log* of the implementation. We refer to the second scenario as the *log-model scenario*.

In the model-model scenario, we assume that the specification and the implementation of an open system are given as formal models. This scenario corresponds to traditional conformance checking on formal models.

Hence, Fig. 5 also illustrates an instance of the model-model scenario. In the model-model scenario, the following two research questions arise:

2. Is the conformance relation arising from responsiveness compositional? If not, what is the compositional conformance relation that preserves responsiveness?
3. How can we decide the (compositional) conformance relation arising from responsiveness?

In the log-model scenario, we assume the specification of an open system to be given as a formal model, but no formal model of the implementation is available. In practice, often no formal model of the implementation is available because the implementation is too complex to be formally modeled. Even if there exists a formal model of the implemented system, it can differ significantly from the actual implementation: The formal model may have been implemented incorrectly, or the implementation may have been changed over time. However, most implementations can provide some kind of observed behavior, commonly referred to as event log. An event log may be extracted from databases, message logs, or audit trails [8]. The idea is to use a formalization *Log* of such an event log to investigate conformance of the unknown implementation to its known specification. Thereby, *Log* is a multiset of traces that abstracts from captured resource or timing information, for example. This is often a more realistic and practically relevant assumption than assuming the availability of a formal model of the implementation as we do in the model-model scenario. Figure 6 depicts our assumptions for the log-model scenario. By investigating the log-model scenario in addition to the model-model scenario, we move closer toward conformance checking in practice.

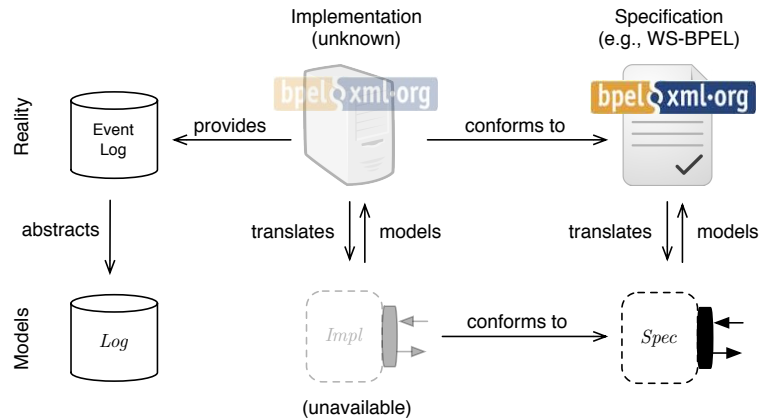


Figure 6: The log-model scenario

The idea for the log-model scenario is fueled by recent advances in the area of process mining [2]. Process mining techniques focus on extracting process models from event logs (“process discovery”), comparing normative models with the reality recorded in event logs (which is also called “conformance testing” [218] or “conformance checking” [12, 9, 219, 15]), and extending models based on event logs (“extension”). In other words, by investigating the log-model scenario, we pursue the goals of the verification method “conformance checking” under the assumptions of (and with techniques from) “conformance checking” in process mining.

In general, an event log captures only example behavior of an implementation; it is highly unrealistic to assume that a complex implementation exhibits every possible behavior while being observed only for a limited amount of time. Therefore, we need to assume an event log to be inherently *incomplete*. This incompleteness hinders the application of traditional verification techniques like conformance checking in the log-model scenario. Still, *testing* for conformance may be applicable. Testing for conformance means that if there is some deviating behavior captured by *Log*, we can conclude that the implementation does not conform to the specification. However, if there is no erroneous behavior captured by *Log*, we cannot make it precise whether the implementation conforms to the specification; we simply cannot say whether conformance holds based on *Log*.

Another approach to support the design of responsive open systems in the log-model scenario is to discover a formal model of the unknown implementation based on *Log*.

We summarize our research questions for the log-model scenario as follows:

4. How can we apply conformance testing using a specification and an event log?
5. How can we discover a formal model of the unknown implementation using a specification and an event log?

1.3 CONTRIBUTIONS

In the previous section, we elaborated five research questions from our problem statement. In this section, we summarize the contributions of this thesis regarding these research questions.

CONTRIBUTION 1: FORMALIZING RESPONSIVENESS We formalize two variants of responsiveness for open nets: responsiveness and *b*-responsiveness. Responsiveness guarantees that either communication between two open nets or termination of the nets is always possible; *b*-responsiveness additionally guarantees that the number of pending messages between two open nets never exceeds a previously known bound *b*. We classify the two variants of responsiveness into a spectrum of behavioral correctness criteria. We refer to two responsive open nets as partners and to two *b*-responsive open nets as *b*-partners. We define the inclusion of all partners of the specification in the set of the partners of the implementation as conformance relation: Intuitively, the implementation interacts desirably (i.e., responsively) with at least all environments of the specification—or even more. In other words, responsiveness (or more precisely, the set of responsive partners) is preserved. Based on partner inclusion, two conformance relations arise from the two variants of responsiveness: the conformance relation that preserves responsiveness (conformance for short) and the conformance relation that preserves *b*-responsiveness (*b*-conformance for short). Finally, we position both relations in the whole spectrum of conformance relations.

CONTRIBUTION 2: CHARACTERIZING THE COMPOSITIONAL CONFORMANCE RELATIONS In the model-model scenario, we assume that the specification and the implementation of an open system are given as open nets. We analyze the conformance relation and the *b*-conformance relation

for compositionality: Technically, conformance and b -conformance are classical preorders and compositionality means that they are precongruences with respect to open system composition as well. Compositional notions of conformance are traditionally considered hard to obtain [51]. Figure 7 illustrates our general approach: To this end, we provide open nets with a certain denotational semantics. Based upon this semantics we define a refinement relation that coincides with the conformance relation under investigation. Then, we investigate compositionality of the refinement relation. In general, this is less difficult than directly investigating the conformance relation, because the denotational semantics abstracts from irrelevant details.

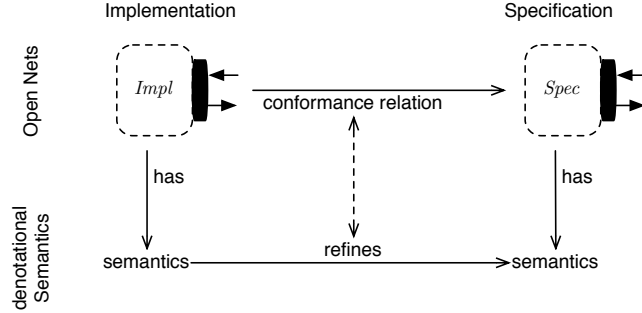


Figure 7: An illustration of how we characterize a conformance relations based upon a denotational semantics for open nets. A solid arc illustrates the relation described by the corresponding arc label. The dashed arc illustrates logical equivalence.

As a concrete example, Fig. 8 illustrates the relation between conformance and the provided denotational semantics, to which we refer as *stopdead*-semantics: We provide both the implementation and the specification with the *stopdead*-semantics and refinement on the *stopdead*-semantics coincides with the conformance relation. It turns out that the conformance relation is not a precongruence; that is, it is not compositional. Therefore, we proceed by characterizing its compositional core—that is, the coarsest precongruence that is contained in the conformance relation. We refer to this precongruence as compositional conformance. Again, we characterize the compositional conformance relation using a denotational semantics for open nets. We refer to this semantics as the \mathcal{F}_{fin}^+ -semantics.

We employ the same approach to investigate the compositionality of b -conformance. As for conformance, it turns out that the b -conformance relation is not a precongruence; that is, it is not compositional. Again, we proceed by characterizing the coarsest precongruence that is contained in the b -conformance relation—that is, compositional b -conformance.

CONTRIBUTION 3: SHOWING (UN-)DECIDABILITY OF THE CHARACTERIZED CONFORMANCE AND COMPOSITIONAL CONFORMANCE RELATIONS

We analyze the (compositional) conformance relation and the (compositional) b -conformance relation for decidability, thereby using their characterizations that we described in the previous contribution. It turns out that conformance and compositional conformance are undecidable, but b -conformance and b -compositional conformance are decidable. Thus, we elaborate a decision procedure for b -conformance and for compositional b -conformance. The tools Chloe [115] and Delain [78] implement the decision algorithm for b -conformance; both tools are free open source software that we develop in the course of this thesis. We evaluate the decision algorithm

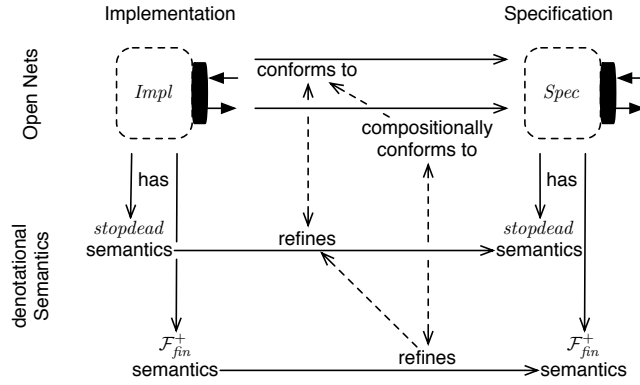


Figure 8: An illustration of how we characterize conformance and compositional conformance based upon denotational semantics for open nets. A solid arc illustrates the relation described by the corresponding arc label. Dashed arcs illustrate logical implication or logical equivalence, depending on their number of heads.

for b -conformance with industrial-sized open nets. For a given open net, we additionally develop a finite characterization of all b -partners and all b -conforming open nets. The finite characterization of all b -partners serves as an alternative decision procedure to decide whether two open nets are b -partners with a better computational worst-case complexity. The finite characterization of all b -conforming open nets serves as an alternative decision procedure for b -conformance. This alternative decision procedure might be more feasible in practice for an implementation with a very large state-space and a specification with a very small state-space.

CONTRIBUTION 4: CONFORMANCE TESTING IN THE LOG-MODEL SCENARIO In the log-model scenario, we assume that the specification of an open system is given as a formal model and the implementation is given as an event log, i.e., example behavior generated by the actual (not modeled) implementation. We consider conformance checking only for b -conformance, because the conformance relation turns out to be undecidable. To this end, we present a necessary condition for deciding b -conformance: We analyze whether there exists a b -conforming implementation which can replay the given event log. Thereby, we use the finite characterization of all b -conforming open nets that we developed in the model-model scenario. Figure 9 sketches this contribution. If there does not exist a b -conforming implementation that can replay the given event log, then the implementation which provided that event log is certainly not b -conforming to the specification. Thus, we provide an approach to test b -conformance in the log-model scenario. We use the implemented decision algorithm for b -conformance from the model-model scenario to develop an algorithm to test b -conformance. We evaluate the implemented algorithm using industrial-sized specifications and event logs.

CONTRIBUTION 5: DISCOVERING A FORMAL MODEL IN THE LOG-MODEL SCENARIO We present an approach to discover a high-quality formal model of the unknown implementation from the event log, assuming the implementation b -conforms to its specification. To judge the discovered model we consider two aspects: b -conformance (i.e., the discovered model b -conforms to the model of the specification) and quality (i.e., the ability of the

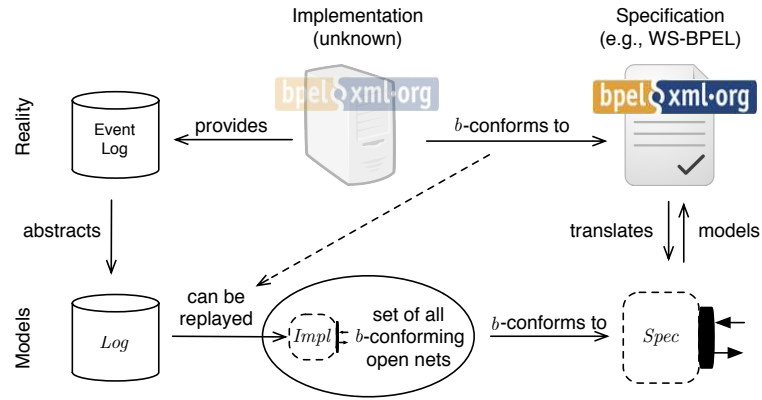


Figure 9: Conformance testing in the log-model scenario. A solid arc illustrates the relation described by the corresponding arc label. The dashed arc illustrates logical implication.

discovered model to describe the observed behavior in the event log well). Regarding quality, there exist four quality dimensions for general process models [2]: (1) *fitness* (i.e., the discovered model should allow for the behavior seen in the event log), (2) *precision* (i.e., the discovered model should not allow for behavior completely unrelated to what was seen in the event log), (3) *generalization* (i.e., the discovered model should generalize the example behavior seen in the event log), and (4) *simplicity* (i.e., the discovered model should be as simple as possible). These quality dimensions compete with each other, as visualized in Fig. 10. For example, to improve the fitness of a model one may end up with a substantially more complex model. A more general model usually means a less precise model. Clearly, the user needs to set priorities when balancing the four quality dimensions.

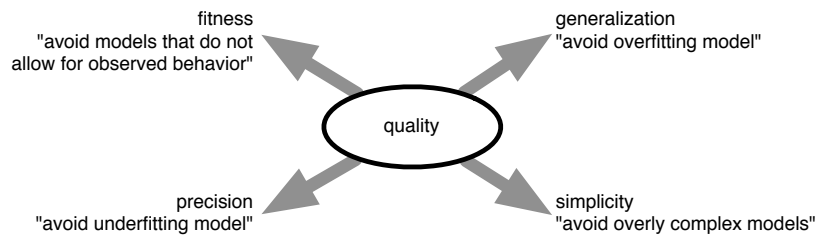


Figure 10: The different quality dimensions for model discovery.

We aim at discovering a formal model of the implementation that *b*-conforms to the given specification and, in addition, balances the four quality dimensions guided by user preferences. In other words, we search for a high-quality model in the set of all *b*-conforming open nets to the given specification. In general, this set is infinite and the competing four quality dimensions are nonlinear. However, we can employ the finite characterization of all *b*-conforming open nets that we developed in the model-model scenario. Based on this finite characterization, we employ a genetic discovery algorithm and a suitable abstraction technique to solve the problem. Figure 11 sketches our contribution. We use the implemented decision algorithm for *b*-conformance (implemented in the tools Chloe [115] and Delain [78] from the model-model scenario) to develop an algorithm to discover a high-quality model of the implementation. We implement the discovery algorithm in the “ServiceDiscovery” ProM plug-in [188], which we develop in the course of

this thesis. ProM [212] is an extensible framework that supports a wide variety of process mining techniques. We evaluate the implemented algorithm using industrial-sized specifications and event logs.

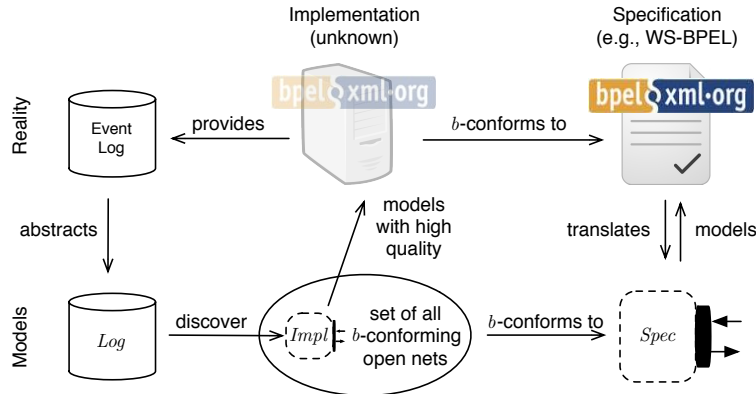


Figure 11: Discovering a high-quality model of the implementation in the log-model scenario. A solid arc illustrates the relation described by the corresponding arc label.

1.4 THESIS OVERVIEW

The thesis consists of four parts and 12 chapters. Figure 12 illustrates the structure of the thesis. An arrow from a chapter A to a chapter B indicates that the content of chapter A is required to understand the content of chapter B.

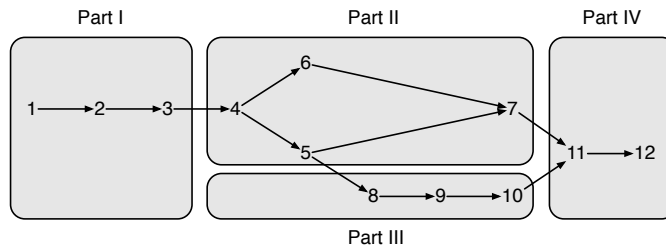


Figure 12: Illustration of the dependencies between the chapters of this thesis.

Part I covers Chap. 1 to Chap. 3. In Chap. 2, we introduce the basic notions needed in the remainder of this thesis. For example, we introduce open nets as the formalism in which we model (the behavior of) open systems and their composition. In Chap. 3, we formalize responsiveness and b -responsiveness and the corresponding conformance relations: conformance and b -conformance. We compare the two variants of responsiveness with two known behavioral correctness criteria for open nets: deadlock freedom and weak termination. In addition, we show that conformance and b -conformance are incomparable. The content of Chap. 3 refers to our first contribution from Sect. 1.3.

Part II covers Chap. 4 to Chap. 7. Here, we investigate conformance checking in the model-model scenario; that is, Part II presents the second and third contribution from Sect. 1.3. Figure 13 reflects how the model-model and the log-model scenario are reflected in the chapters of this thesis. We characterize conformance and compositional conformance in Chap. 4 and

show that both relations are undecidable. In Chap. 5, we investigate b -conformance. In particular, we show decidability of b -conformance and present two decision procedures. In Chap. 6, we characterize compositional b -conformance and show its decidability. Finally, Chap. 7 summarizes the results of Part II and reviews related work.

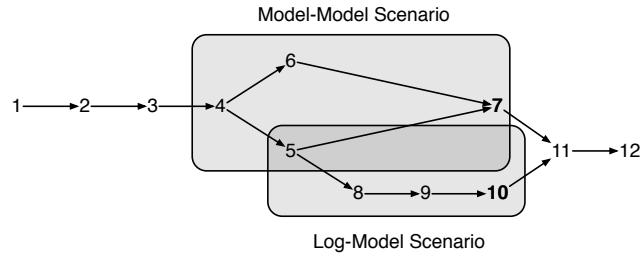


Figure 13: Illustration of the model-model scenario and the log-model scenario reflected in the chapters of this thesis. We highlighted Chap. 7 and Chap. 10 because they summarize the results of the respective parts.

Chapter 8 to Chap. 10 form Part III of this thesis, i.e., our study of the log-model scenario. All of them refer to our fourth and fifth contribution from Sect. 1.3. In Chap. 8, we recapitulate the formalization of an event log and how it is replayed on the model of an open system. Based on that formalization, we focus on the problem whether there exists a b -conforming implementation which can produce a given event log. In Chap. 9, we formalize the idea of model quality and discover a formal model of the unknown implementation, assuming that the implementation b -conforms to the given specification. Chapter 10 summarizes the results of the log-model scenario and reviews related work.

Finally, Part IV consists of Chap. 11 and Chap. 12. We demonstrate the applicability of our results in Chap. 11. Chapter 12 summarizes the contributions and limitations of this thesis and outlines future work.

As shown in Fig. 14, Chap. 4 focuses on responsiveness, whereas Chapters 5, 6, 8, and 9 focus on b -responsiveness. The undecidability of conformance and compositional conformance (i.e., both relations that arise from responsiveness) motivates our investigation of b -responsiveness and the arising b -conformance relation and compositional b -conformance relation. We show the undecidability of conformance and compositional conformance in one chapter—that is, Chap. 4—because both undecidability proofs are similarly structured.

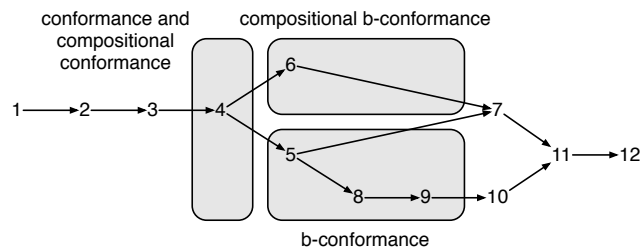


Figure 14: Illustration of the variant of conformance and b -conformance that we investigate in the different chapters.

IN this chapter, we introduce the basic notions from mathematics and computer science used in this thesis. We start by recapitulating sets and multisets, binary relations, and words and languages in Sect. 2.1. In Sect. 2.2, we introduce labeled transition systems, and we present Petri nets in Sect. 2.3. Then, we introduce open nets, a variant of Petri nets, in which we model (the behavior of) open systems and their composition in Sect. 2.4. We introduce an open net environment in Sect. 2.6 as a tool with whom we describe the semantics of an open net. We relate open nets to their environments in Sect. 2.6, and close this chapter with a discussion of the choice of open nets as our formal model in Sect. 2.7.

2.1 BASIC MATHEMATICAL NOTIONS

SETS We denote the set of natural numbers $\{0, 1, 2, 3, \dots\}$ with \mathbb{N} and the set of positive natural numbers (i.e., excluding 0) with \mathbb{N}^+ . As usual, we denote membership in a set by \in ; for example, we have $0 \in \mathbb{N}$ but $0 \notin \mathbb{N}^+$. We denote the empty set by \emptyset , set inclusion by \subseteq , set union by \cup , set intersection by \cap , and set difference by \setminus . For a set A , we denote its cardinality with $|A|$ and its powerset with $\mathcal{P}(A)$. For two sets A and B , let $A \times B$ denote their Cartesian product and $A \uplus B$ the disjoint union of A and B ; writing $A \uplus B$ implies that A and B are implicitly assumed to be disjoint.

MULTISSETS A multiset or bag M over a set A is a mapping $M : A \rightarrow \mathbb{N}$. We denote a multiset as a list; for example, we write $[x, y, y]$ for a multiset over a set A (with $x, y \in A$) that contains the element x once, the element y twice, and no other element of A . As usual, $[\]$ denotes the empty multiset, and we denote the set of all multisets over a set A with $\text{Bags}(A)$. We define $+$ for the sum and $-$ for the difference of two multisets and $=, <, >, \leq, \geq$ for the comparison of two multisets in the standard way (i.e., pointwise). By abuse of notation, we write $M \subseteq B$ for a multiset M over A and a set B if for all $x \in A$ with $M(x) > 0$, $x \in B$.

We canonically extend the notion of a multiset over A to supersets $B \supseteq A$; that is, for a mapping $M : A \rightarrow \mathbb{N}$, we extend M to the multiset $M : B \rightarrow \mathbb{N}$ such that for all $x \in B \setminus A$, $M(x) = 0$. Conversely, a multiset over A can be restricted to a subset $B \subseteq A$. For a mapping $M : A \rightarrow \mathbb{N}$, the restriction of M to B is denoted by $M|_B : B \rightarrow \mathbb{N}$. For example, $[x, y, y, z]|_{\{y, z\}} = [y, y, z]$. We lift the extension and restriction of a multiset to a set of multisets in the standard way (i.e., element-wise). For example, $\{[x, y, y, z], [x, x]\}|_{\{y, z\}} = \{[y, y, z], [\]\}$.

BINARY RELATIONS A (binary) relation \leq over a set A is a subset of $A \times A$; \leq is a *preorder* if it is reflexive (i.e., for all $a \in A$: $a \leq a$) and transitive (i.e., for all $a, b, c \in A$: $a \leq b$ and $b \leq c$ implies $a \leq c$). A preorder \leq over a set A is a *partial order* if it is antisymmetric (i.e., for all $a, b \in A$: $a \leq b$ and $b \leq a$ implies $a = b$), and \leq is an *equivalence relation* if it is symmetric (i.e., for all $a, b \in A$: $a \leq b$ implies $b \leq a$). A preorder \leq over a set A is a *precongruence* with respect to a mapping $+$: $A \times A \rightarrow A$ if \leq

is preserved by $+$; formally, we have for all $a, b \in A$: $a \leq b$ implies for all $c \in A$: $a + c \leq b + c$.

WORDS AND LANGUAGES An *alphabet* is a set of symbols, a sequence of symbols over an alphabet is a *word*, and a set of words is a *language*. We denote the set of all finite words over an alphabet Σ with Σ^* . With $v \sqsubseteq w$ we denote that a word v is a *prefix* of a word w ; then, w is a *continuation* of v . As usual, ε denotes the *empty word*, and ε is a prefix of every word. We write $|w|$ for the length of a word w , and $|w|_x$ denotes how many times the symbol x occurs in the word w . Let Σ_1 and Σ_2 be two alphabets. For a word $w \in \Sigma_1^*$ and $\Sigma_2 \subseteq \Sigma_1$, $w|_{\Sigma_2}$ denotes the *projection* of w to the alphabet Σ_2 .

We introduce a few, more compact notations on languages.

Definition 1 [closures, remainder, complement]

Given a language $L \subseteq \Sigma^*$ over an alphabet Σ ,

- $\downarrow L = \{u \in \Sigma^* \mid \exists v \in L : u \sqsubseteq v\}$ is the *prefix closure* of L ,
- $\uparrow L = \{u \in \Sigma^* \mid \exists v \in L : v \sqsubseteq u\}$ is the *suffix closure* of L ,
- $v^{-1}L = \{u \in \Sigma^* \mid vu \in L\}$ is the *remainder* of $v \in \Sigma^*$ in L , and
- $co-L = \Sigma^* \setminus L$ is the *complement* of L .

For the suffix closure, we can show the following properties.

Lemma 2 [suffix closure]

Let $X, Y \in \mathcal{P}(\Sigma^*)$. Then the following properties hold:

1. $\uparrow (X \cup Y) = \uparrow X \cup \uparrow Y$
2. $x^{-1}(X \cup Y) = x^{-1}X \cup x^{-1}Y$ for any $x \in \Sigma$
3. $y \notin \uparrow Y$ implies $y^{-1}(X \cup \uparrow Y) \subseteq \uparrow (y^{-1}(X \cup Y))$

Proof. Items (1) and (2) are trivial. For (3) observe that $y^{-1}(X \cup \uparrow Y) = y^{-1}X \cup y^{-1}\uparrow Y \subseteq \uparrow (y^{-1}X) \cup \uparrow (y^{-1}Y) = \uparrow (y^{-1}(X \cup Y))$. \square

2.2 LABELED TRANSITION SYSTEMS

Labeled transition systems (LTSs) are a uniform formalism for modeling the behavior of systems. They are widely used in the theory of computation [224] and for the verification of distributed systems [28, 49]. An LTS is an abstract machine consisting of a set of states and labeled transitions between states; a usual extension of the definition includes a fixed initial state and a state labeling function [28]. For merely technical reasons, we additionally distinguish the transition labels between input-, output- and internal actions.

Definition 3 [labeled transition system]

A *labeled transition system* (LTS) $S = (Q, \delta, q_S, \Sigma^{in}, \Sigma^{out}, \lambda)$ consists of

- a set Q of *states*,
- a *labeled transition relation* $\delta \subseteq Q \times (\Sigma^{in} \uplus \Sigma^{out} \uplus \{\tau\}) \times Q$,
- an *initial state* $q_S \in Q$,

- an *alphabet* $\Sigma = \Sigma^{in} \uplus \Sigma^{out}$ of disjoint *input actions* Σ^{in} and *output actions* Σ^{out} , and
- a *state labeling function* $\lambda : Q \rightarrow \mathbb{N}$.

$\Sigma \uplus \{\tau\}$ is the set of *labels* of S , and the label τ denotes an *internal action*.

Whenever the state labeling function of an LTS $S = (Q, \delta, q_S, \Sigma^{in}, \Sigma^{out}, \lambda)$ distinguishes only two kinds of states, we introduce *final states* as a separate notion for readability reasons: We employ a set of final states $\Omega \subseteq Q$ instead of the state labeling function λ , i.e., $S = (Q, \delta, q_S, \Sigma^{in}, \Sigma^{out}, \Omega)$.

Convention 1 Introducing an LTS S also implicitly introduces its components $Q, \delta, q_S, \Sigma^{in}, \Sigma^{out}, \lambda$ or Ω ; the same applies to LTS S', S_1 , etc. and their components $Q', \delta', q_{S'}, \Sigma^{in'}, \Sigma^{out'}, \lambda'$ or Ω' , and $Q_1, \delta_1, q_{S_1}, \Sigma_1^{in}, \Sigma_1^{out}, \lambda_1$ or Ω_1 , respectively—and it also applies to other structures later on. \diamond

An LTS S is *finite* if both Q and Σ are finite; it is τ -*free* if no transition is labeled with τ ; and it is *deterministic* if for all $q, q', q'' \in Q, x \in \Sigma: (q, \tau, q') \in \delta$ implies $q = q'$, and $(q, x, q'), (q, x, q'') \in \delta$ implies $q' = q''$. Two LTSs are *action-equivalent* if they have the same sets of input and output actions. An LTS $S' = (Q', \delta', q_{S'}, \Sigma^{in}, \Sigma^{out}, \lambda')$ is an (initialized) *subsystem* of an LTS $S = (Q, \delta, q_S, \Sigma^{in}, \Sigma^{out}, \lambda)$, denoted by $S' \subseteq S$, if $Q' \subseteq Q, \delta' \subseteq \delta$, and, for all $q \in Q', \lambda'(q) = \lambda(q)$.

A transition $(q, x, q') \in \delta$ is an incoming transition of q' and an outgoing transition of q . We write $q \xrightarrow{x} q'$ for $(q, x, q') \in \delta$ and $q \xrightarrow{x}$ if there exists a state q' such that $q \xrightarrow{x} q'$. We extend this to sequences: A transition sequence $q_1 \xrightarrow{v_1} q_2 \xrightarrow{v_2} \dots \xrightarrow{v_k} q_{k+1}$ is a *run* of S from q_1 to q_{k+1} if for all $1 \leq i \leq k, q_i \xrightarrow{v_i} q_{i+1}$; for $v = v_1 v_2 \dots v_k$, we also write $q_1 \xrightarrow{v} q_{k+1}$ instead of $q_1 \xrightarrow{v_1} q_2 \xrightarrow{v_2} \dots \xrightarrow{v_k} q_{k+1}$ and $q_1 \xrightarrow{v}$ instead of $q_1 \xrightarrow{v_1} q_2 \xrightarrow{v_2} \dots \xrightarrow{v_{k-1}} q_k \xrightarrow{v_k}$. If $q \xrightarrow{v} q'$, we say that q' is *reachable* from q with v ; a state q is *reachable* in S if q is reachable from the initial state q_S . If $q \xrightarrow{v} q'$ and $w \in \Sigma^*$ is obtained from v by removing all τ labels, then we write $q \xrightarrow{w} q'$. Similarly, if $q \xrightarrow{v}$ and $w \in \Sigma^*$ is obtained from v by removing all τ labels, then we write $q \xrightarrow{w}$. A *trace* of S is a word $w \in \Sigma^*$ such that $q_S \xrightarrow{w}$; the *language* $L(S)$ of S is the set of all traces of S . We define $L_i(S) = \{w \in \Sigma^* \mid q_S \xrightarrow{w} q \wedge \lambda(q) = i\}$ as the language of S restricted to traces leading to states labeled with $i \in \mathbb{N}$, and $L_\Omega(S) = \{w \in \Sigma^* \mid q_S \xrightarrow{w} q \wedge q \in \Omega\}$ in the case of final states.

Convention 2 In the remainder of this thesis, we implicitly assume any LTS to have only reachable states. In other words, we do not consider LTSs with unreachable states. \diamond

In this thesis, we strive for readability and understandability of the presented concepts and, hence, seek to avoid switching between different formalism, whenever possible. Therefore, we introduced the notion of a language on labeled transition systems and not on the frequently used formalism of finite automata [224, 123]. In contrast to an LTS, a finite automaton has a set of accepting states that determines the automaton's language; accepting states are in general unrelated to labeled states or final states of any LTS. We can understand a finite LTS as a finite automaton if we regard the transition label τ as the empty word ε and consider all states as accepting states. That way, our notions of τ -freeness, determinism, and language co-

incide with the automata-theoretic notions of the same name. In case we consider all i -labeled states ($i \in \mathbb{N}$) as accepting states, then the automata-theoretic notion of language coincides with the language $L_i(S)$. That way, a family of finite automata that characterizes a family $(L_i)_{i \in \mathbb{N}}$ of languages can be represented by one finite LTS.

Graphically, a rounded rectangle represents a state, and a directed arc between two states represents a transition. We depict the identity of a state inside the rounded rectangle and a transition's label next to the directed arc; two transitions between the same states but with different transition labels are shown as one directed arc with the according transition labels separated by a comma. We indicate the initial state by an unlabeled arc without source.

Example 4 Consider the four LTSs S_1 , S_2 , S_3 , and S_4 in Fig. 15 and let $\Sigma^{in} = \{a\}$ and $\Sigma^{out} = \{b, c\}$. We have $S_1 = (\{q_0, q_1, q_2, q_3, q_4\}, \{(q_0, a, q_1), (q_1, b, q_3), (q_0, a, q_2), (q_2, c, q_4)\}, q_0, \Sigma^{in}, \Sigma^{out}, \lambda_1)$, $S_2 = (\{r_0, r_1, r_2, r_3, r_4\}, \{(r_0, a, r_1), (r_1, \tau, r_1), (r_1, b, r_3), (r_0, a, r_2), (r_2, c, r_4)\}, r_0, \Sigma^{in}, \Sigma^{out}, \lambda_2)$, $S_3 = (\{s_0, s_1, s_2, s_3\}, \{(s_0, a, s_1), (s_1, b, s_2), (s_1, c, s_3)\}, s_0, \Sigma^{out}, \Sigma^{in}, \lambda_3)$, and $S_4 = (\{t_0, t_1, t_2, t_3, t_4\}, \{(t_0, a, t_1), (t_1, b, t_2), (t_1, c, t_3), (t_1, c, t_4)\}, t_0, \Sigma^{out}, \Sigma^{in}, \lambda_4)$. All four LTS are finite. S_1 , S_2 , and S_4 are nondeterministic (S_1 and S_2 because of their initial states, and S_4 because of state t_1), but S_3 is deterministic. S_1 , S_3 , and S_4 are τ -free, whereas S_2 is not because of transition $r_1 \xrightarrow{\tau} r_1$. In addition, S_1 and S_2 as well as S_3 and S_4 are action-equivalent.

Although different in structure, S_1 , S_2 , S_3 , and S_4 have identical languages—that is, $L(S_1) = L(S_2) = L(S_3) = L(S_4) = \{\varepsilon, a, ab, ac\}$. In general, the state labeling function λ of an LTS S induces a family of languages $(L_i)_{i \in \mathbb{N}}$ that are subsets of $L(S)$: For example, let us assume that the state labeling function λ_4 of S_4 is defined by $\lambda_4(t_0) = 0$, $\lambda_4(t_1) = 1$, $\lambda_4(t_2) = 2$, $\lambda_4(t_3) = 3$, and $\lambda_4(t_4) = 4$. Then, we have $L_0(S_4) = \{\varepsilon\}$, $L_1(S_4) = \{a\}$, $L_2(S_4) = \{ab\}$, and $L_3(S_4) = L_4(S_4) = \{ac\}$. Regardless of the actual definition of λ_4 , we always have $L_i(S_4) \subseteq L(S_4)$ for $i \in \mathbb{N}$. \diamond

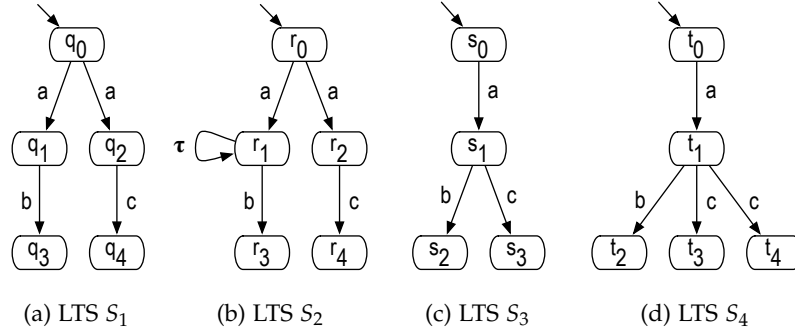


Figure 15: Four labeled transition systems. In addition to the figures, we have $\Sigma_{S_1}^{in} = \Sigma_{S_2}^{in} = \Sigma_{S_3}^{in} = \Sigma_{S_4}^{in} = \{a\}$ and $\Sigma_{S_1}^{out} = \Sigma_{S_2}^{out} = \Sigma_{S_3}^{out} = \Sigma_{S_4}^{out} = \{b, c\}$.

We frequently compare the structure of two given LTS using a *(weak) simulation relation* or *(weak) bisimulation* [202, 177].

Definition 5 [(weak) simulation, (weak) bisimulation]

Let S_1 and S_2 be two action-equivalent LTSs. A binary relation $\rho \subseteq Q_1 \times Q_2$ is a

- *simulation* relation if for all $(q_1, q_2) \in \varrho$, for all $x \in \Sigma_1 \uplus \{\tau\}$ and for all states $q'_1 \in Q_1$ such that $q_1 \xrightarrow{x} q'_1$, there exists a state $q'_2 \in Q_2$ such that $q_2 \xrightarrow{x} q'_2$ and $(q'_1, q'_2) \in \varrho$.
- *weak simulation* relation if for all $(q_1, q_2) \in \varrho$, for all $x \in \Sigma_1 \uplus \{\tau\}$ and for all states $q'_1 \in Q_1$ such that $q_1 \xrightarrow{x} q'_1$, there exists a state $q'_2 \in Q_2$ such that $q_2 \xrightarrow{x} q'_2$ and $(q'_1, q'_2) \in \varrho$.

S_1 is simulated (weakly simulated) by S_2 if there exists a simulation (weak simulation) relation relating their initial states q_{S_1} and q_{S_2} .

If ϱ and ϱ^{-1} are simulations (weak simulations), then ϱ is a *bisimulation* (weak bisimulation) relation. S_1 and S_2 are *bisimilar* (weakly bisimilar) if there exists a bisimulation (weak bisimulation) relation relating their initial states q_{S_1} and q_{S_2} .

If the LTS S_2 in Def. 5 is deterministic (and we only consider LTSs with reachable states as stated in Conv. 2), then the least (weak) simulation and the least (weak) bisimulation of S_1 and S_2 is uniquely defined.

Example 6 Consider again the four LTSs S_1 , S_2 , S_3 , and S_4 in Fig. 15. S_1 is simulated by S_2 with the simulation relation $\varrho = \{(q_0, r_0), (q_1, r_1), (q_2, r_2), (q_3, r_3), (q_4, r_4)\}$. The simulation relation ϱ is also a weak simulation relation of S_1 by S_2 . In contrast, S_2 is not simulated by S_1 because of transition (r_1, τ, r_1) , but weakly simulated by S_1 with the weak simulation relation $\{(r_0, q_0), (r_1, q_1), (r_2, q_2), (r_3, q_3), (r_4, q_4)\}$.

The LTSs S_3 and S_4 are bisimilar with, for example, the bisimulation relation $\pi = \{(s_0, t_0), (s_1, t_1), (s_2, t_2), (s_3, t_3), (s_3, t_4)\}$. In other words, π is a simulation relation of S_3 by S_4 and the inverse $\pi^{-1} = \{(t_0, s_0), (t_1, s_1), (t_2, s_2), (t_3, s_3), (t_3, s_4)\}$ of π is a simulation relation of S_4 by S_3 . Because S_4 is not deterministic, the simulation relation π is not unique: for example, $\{(s_0, t_0), (s_1, t_1), (s_2, t_2), (s_3, t_3)\}$ is another simulation relation of S_3 by S_4 . Observe that there does not exist any (weak) simulation relation between S_1 or S_2 and S_3 or S_4 by Def. 5 because S_1 or S_2 and S_3 or S_4 are not action-equivalent. \diamond

2.3 PETRI NETS

In this section, we introduce place/transition Petri nets [216] extended with a set of final markings and transition labels.

Definition 7 [net]

A net $N = (P, T, F, m_N, \Omega)$ consists of

- a set P of *places*,
- a set T of *transitions* such that P and T are disjoint,
- a *flow relation* $F \subseteq (P \times T) \uplus (T \times P)$,
- an *initial marking* m_N , where a marking is a multiset over P , and
- a set Ω of *final markings*.

Usually, we are interested in finite nets—that is, nets with finite sets P and T —but we shall also make use of infinite nets. Graphically, a circle represents a place, a box represents a transition, and the directed arcs between places and transitions represent the flow relation. A marking m is a distribution of tokens over the places, and a place p is *marked* at m if $m(p) > 0$. Graphically, a black dot represents a token.

Let $x \in P \uplus T$ be a node of a net N . As usual, $\bullet x = \{y \mid (y, x) \in F\}$ denotes the *preset* of x and $x^\bullet = \{y \mid (x, y) \in F\}$ the *postset* of x . We canonically extend the notion of presets and postsets to sets of nodes of N , and interpret presets and postsets as multisets when used in operations also involving multisets.

The *behavior* of a net N relies on changing a marking of N by the firing of transitions of N . A transition $t \in T$ is *enabled* at a marking m , denoted by $m \xrightarrow{t}$, if for all $p \in \bullet t$, $m(p) > 0$. If t is enabled at m , it can *fire*, thereby changing the current marking m to a marking $m' = m - \bullet t + t^\bullet$. The firing of t is denoted by $m \xrightarrow{t} m'$; that is, t is enabled at m and firing t results in m' . We extend this to firing sequences: $m_1 \xrightarrow{t_1} \dots \xrightarrow{t_k} m_{k+1}$ is a *run* of N from m_1 to m_{k+1} , if for all $1 \leq i \leq k$, $m_i \xrightarrow{t_i} m_{i+1}$; for $v = t_1 \dots t_k$, we also write $m_1 \xrightarrow{v} m_{k+1}$ instead of $m_1 \xrightarrow{t_1} \dots \xrightarrow{t_k} m_{k+1}$. A marking m' is *reachable* from a marking m if there exists a (possibly empty) run from m to m' , and m' is *reachable* in N if it is reachable from the initial marking m_N . The set M_N represents the set of all reachable markings of N . The *reachability graph* $RG(N)$ of N is a labeled transition system with the reachable markings M_N as its states and a t -labeled transition from m to m' whenever $m \xrightarrow{t} m'$ in N . The set of final states of $RG(N)$ is the set of reachable final markings of N .

Finally, we introduce with boundedness, deadlock freedom, and weak termination three behavioral properties of a net. A marking m of a net N is *b-bounded* for a bound $b \in \mathbb{N}$ if $m(p) \leq b$ for all $p \in P$. The net N is *b-bounded* if every reachable marking is *b-bounded*; it is *bounded* if it is *b-bounded* for some $b \in \mathbb{N}$. A marking $m \notin \Omega$ of a net N is a *deadlock* if no transition of N is enabled at m ; N is *deadlock-free* if no deadlock is reachable in N . Compared to the standard definition of a deadlock [216]—that is, a marking that does not enable any transition—we additionally distinguish between final and nonfinal markings: Only a nonfinal marking may be a deadlock, because final markings model successful termination. A marking m of a net N is *weakly terminating* if there is a marking $m' \in \Omega$ reachable from m ; N is *weakly terminating* if every reachable marking of N is weakly terminating.

Convention 3 Throughout this thesis, b denotes a bound—a positive natural number. \diamond

Example 8 Figure 16 depicts the nets N_1 and N_2 , both consisting of two places p_0 and p_1 , three transitions t_0 , t_1 , and t_2 , and the initial marking $[p_0]$. The net N_1 is unbounded, because transition t_0 has an empty preset (and, thus, is always enabled) and may produce any number of tokens on p_0 , thereby violating any bound b . Thus, the set of reachable markings M_{N_1} of N_1 and, also, the reachability graph $RG(N_1)$ are infinite. In contrast, the net N_2 is 2-bounded and, therefore, bounded; the set of reachable markings of is $M_{N_2} = \{[], [p_0], [p_1], [p_0, p_0], [p_0, p_1], [p_1, p_1]\}$. Both nets are deadlock-free: N_1 is deadlock-free because transition t_0 is enabled at every

reachable marking of N_1 , and N_2 is deadlock-free because the only reachable marking $[\]$ that does not enable any transition is a final marking of N_2 . If we assume $[\]$ is not a final marking of N_2 , then $[\]$ is a reachable deadlock of N_2 and, thus, N_2 is not deadlock-free. The net N_1 is not weakly terminating: If transition t_0 fires—leading to the marking $[p_0, p_0]$ —then the only final marking $[p_0]$ is no longer reachable. In contrast, the net N_2 is weakly terminating, because the final marking $[\]$ is reachable from every reachable marking of N_2 . \diamond

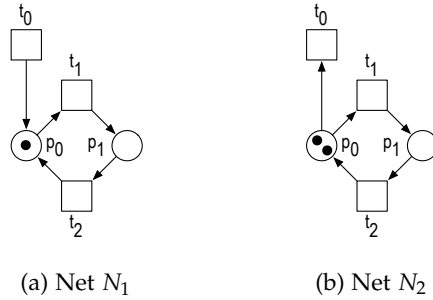


Figure 16: Two nets. In addition to the figures, we have $\Omega_{N_1} = \{p_0\}$ and $\Omega_{N_2} = \{[\]\}$.

A labeled net extends a net by transition labels. Transition labels model visible and invisible actions like the transition labels of an LTS. Using the notion of a net's reachability graph, we can, therefore, model an (but not necessarily any) infinite LTS as a finite labeled net. In a sense, a labeled net serves as a more compact model of (the behavior of) a system that is initially modeled as an LTS.

Definition 9 [labeled net]

A labeled net $N = (P, T, F, m_N, \Omega, \Sigma^{in}, \Sigma^{out}, l)$ is a net (P, T, F, m_N, Ω) together with

- an alphabet $\Sigma = \Sigma^{in} \uplus \Sigma^{out}$ of disjoint input actions Σ^{in} and output actions Σ^{out} , and
- a labeling function $l : T \rightarrow \Sigma \uplus \{\tau\}$, where τ represents an invisible, internal action.

Graphically, we represent a labeled net like a net. In addition, we depict a transition label inside the transition with bold font to distinguish it from the transition's identity.

We canonically lift the notions of τ -freeness, action-equivalence, traces, and language from LTS to labeled nets: A labeled net N is τ -free if no transition of N is labeled with τ . Two labeled nets are *action-equivalent* if they have the same sets of input and output actions. If $m \xrightarrow{v} m'$ and w is obtained from v by replacing each transition with its label and removing all τ labels, we write $m \xrightarrow{w} m'$. If $m_N \xrightarrow{w}$, then w is a *trace* of the labeled net N and the *language* $L(N)$ of N is the set of all traces of N . For a trace w , its *Parikh vector* $Parikh(w) : \Sigma \rightarrow \mathbb{N}$ maps every action $a \in \Sigma$ to the number of occurrences of a in w . For the reachability graph $RG(N)$ of a labeled net N , we replace each transition label t with $l(t)$, and the sets of input and output actions of N and $RG(N)$ coincide, respectively. Finally, we canonically lift the notions of (weak) simulation and (weak) bisimulation from two action-equivalent

LTSs in Def. 5 to two action-equivalent labeled nets N_1 and N_2 by relating their reachability graphs $RG(N_1)$ and $RG(N_2)$.

Example 10 Figure 17 depicts the labeled net N_3 and its reachability graph $RG(N_3)$. Structurally, N_3 coincides with the net N_2 from Fig. 16b, thus N_3 is bounded and weakly terminating. Because N_3 is bounded, its reachability graph is finite. The language of N_3 is the set $L(N_3) = \{w \in \{a, b\}^* \mid |w|_a \leq 1\} \uplus \{wa \in \{a, b\}^* \mid |wa|_a = 2\}$. It coincides with the language $L(RG(N_3))$ of the reachability graph of N_3 . \diamond

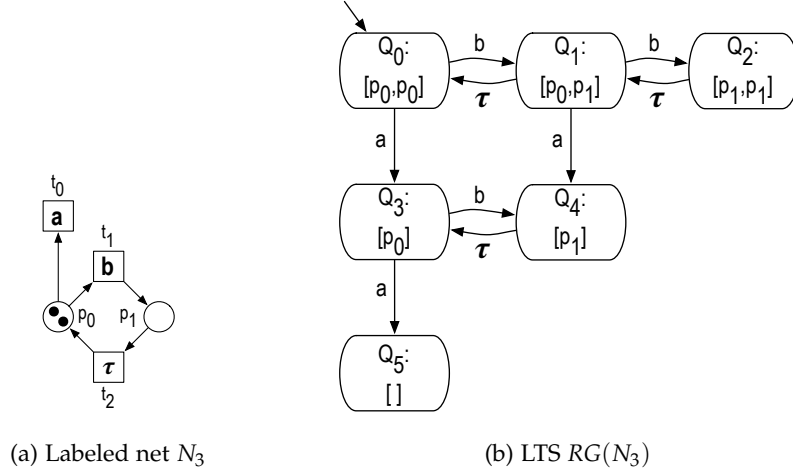


Figure 17: A labeled net and its reachability graph. In addition to the figures, we have $\Omega_{N_3} = \{[]\}$ and $\Omega_{RG(N_3)} = \{Q_5\}$.

2.4 OPEN NETS AND THEIR COMPOSITION

We model open systems as *open nets* [246, 153], thereby restricting ourselves to the communication protocol of an open system. An open net extends a net by an interface. An interface consists of two disjoint sets of input and output places corresponding to asynchronous input and output channels. In the model, we abstract from data and represent each message by a token on the respective interface place. In the initial marking and the final markings, interface places are not marked. An input place has an empty preset, and an output place has an empty postset. For merely technical reasons, we consider only open nets that have either at least one input and one output place or no input and output places.

Definition 11 [open net]

An *open net* $N = (P, T, F, m_N, \Omega, I, O)$ is a net $(P \uplus I \uplus O, T, F, m_N, \Omega)$ where

- the set I of *input places* satisfies for all $p \in I$, $\bullet p = \emptyset$;
- the set O of *output places* satisfies for all $p \in O$, $p^\bullet = \emptyset$;
- for all $p \in I \uplus O$, $m_N(p) = 0$ and for all $m \in \Omega$, $m(p) = 0$; and
- set $I = \emptyset$ if and only if set $O = \emptyset$.

The set P is the set of *internal* places of N and the set $I \uplus O$ is the set of *interface* places of N . If $I = O = \emptyset$, then N is a *closed net*. If every transition of N is connected to at most one interface place, then N is *sequentially communicating*. Two open nets are *interface-equivalent* if they have the same sets of input and of output places.

Graphically, we represent an open net like a net with a dashed frame around it. An interface place p is positioned on the frame; an additional arrow indicates whether p is an input or an output place.

Example 12 Figure 18 shows two open systems, each modeled as an open net. The open net S in Fig. 18a models an unreliable time server that sends its timing information (output place t) to some client and processes its responses (input place r). Anytime before sending the next timing information, an error may happen (output place e) and the server shuts down (and final marking $[\]$ can be reached). As a typical time server, S is originally intended to be always running, thus $[\]$ is the only final marking. The interface places of S are e , t , and r ; its internal places are p_0 and p_1 . S is not a closed net and sequentially communicating.

The open net C in Fig. 18b models a client of the time server S . It repeatedly updates its system time by the timing information sent by the server (input place t) and responds with a response packet (output place r). If the client receives an error message from the server (input place e), it continuously tries to reset the time server (output place r). If resetting the server was successful—that is, the client receives timing information again— C may recover and respond with a response packet like before. The output places of C are the input places of S and vice versa. The internal places of C are p_2 , p_3 , and p_4 . Like S , the open net C is not a closed net and sequentially communicating. \diamond

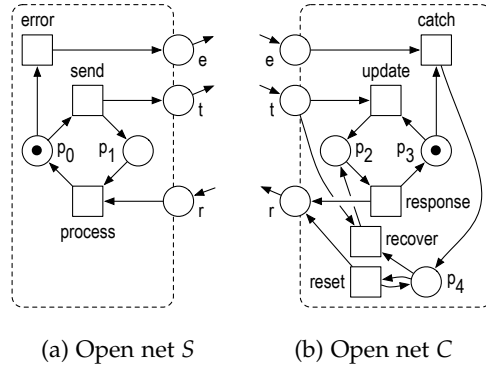


Figure 18: Two open nets modeling an unreliable time server and one of its clients. In addition to the figures, we have $\Omega_S = \{[\]\}$ and $\Omega_C = \{[p_3]\}$.

For the composition of open nets, we assume that the sets of transitions are pairwise disjoint and that no internal place of an open net is a place of any other open net. In contrast, the interfaces intentionally overlap. We require that all communication is *bilateral* and *directed*; that is, every shared place p has only one open net that sends into p and one open net that receives from p . In addition, we require that either (1) all interface places are shared or (2) there is at least one input and one output place which are not shared. We refer to open nets that fulfill these conditions as *composable*. We compose two composable open nets N_1 and N_2 by merging shared interface places and turning these places into internal places. The definition of *com-*

possible thereby guarantees that an open net composition is again an open net (possibly a closed net).

Definition 13 [open net composition]

Two open nets N_1 and N_2 are *composable* if $(P_1 \uplus T_1 \uplus I_1 \uplus O_1) \cap (P_2 \uplus T_2 \uplus I_2 \uplus O_2) = (I_1 \cap O_2) \uplus (I_2 \cap O_1)$, and $(I_1 \uplus I_2) \setminus (O_1 \uplus O_2)$ and $(O_1 \uplus O_2) \setminus (I_1 \uplus I_2)$ are both either empty or nonempty. The *composition* of such open nets is the open net $N_1 \oplus N_2 = (P, T, F, m_N, \Omega, I, O)$, where

- $P = P_1 \uplus P_2 \uplus (I_1 \cap O_2) \uplus (I_2 \cap O_1)$,
- $T = T_1 \uplus T_2$,
- $F = F_1 \uplus F_2$,
- $m_N = m_{N_1} + m_{N_2}$,
- $\Omega = \{m_1 + m_2 \mid m_1 \in \Omega_1, m_2 \in \Omega_2\}$,
- $I = (I_1 \uplus I_2) \setminus (O_1 \uplus O_2)$, and
- $O = (O_1 \uplus O_2) \setminus (I_1 \uplus I_2)$.

Recall that an interface place of an open net N is empty at the initial marking of N and at every final marking of N by Def. 11. Thus, the composition $N_1 \oplus N_2$ of two composable open nets N_1 and N_2 is well-defined—it is an open net again. In general, $N_1 \oplus N_2$ is not a closed net. Figure 19 gives a schematic example for open net composition. The open nets N_1 and N_2 are composable because only their sets of interface places overlap: x and y are interface places in both N_1 and N_2 . Therefore, the places x and y turn into internal places in the composition $N_1 \oplus N_2$, which is again an open net.

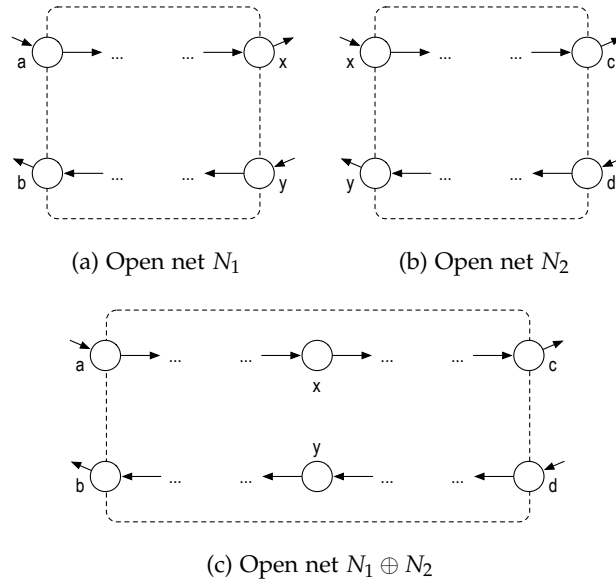


Figure 19: Schematic example of two open nets N_1 , N_2 and their composition $N_1 \oplus N_2$.

Example 14 The open nets S and C in Fig. 18 are composable: Every input place of one open net is an output place of the other, and vice versa. More-

over, no other places are shared. In contrast to our schematic example in Fig. 19c, the composition $S \oplus C$ is a closed net, which we depict in Fig. 20. \diamond

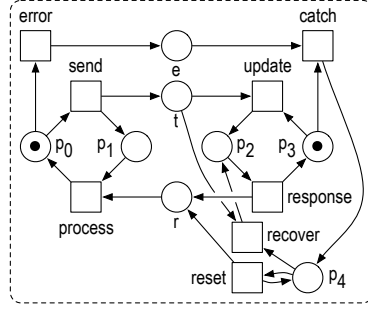


Figure 20: The composition $S \oplus C$ of the open nets S and C from Fig. 18. In addition to the figure, we have $\Omega_{S \oplus C} = \{[p_3]\}$.

We frequently reason about the inner structure of an open net N . To this end, we turn N into a labeled net $inner(N)$ —the *inner net* of N —by removing all interface places and by labeling every transition that is connected to an interface place p in N with the label p .

Definition 15 [inner net]

The *inner net* of an open net N is the labeled net $inner(N) = (P, T, F \setminus ((I \times T) \uplus (T \times O)), m_N|_P, \Omega|_P, I, O, l)$, where

$$l(t) = \begin{cases} i, & i \in I \wedge (i, t) \in F \\ o, & o \in O \wedge (t, o) \in F \\ \tau, & \text{otherwise.} \end{cases}$$

In this thesis, we consider only sequentially communicating open nets. Thus, the inner net of an open net in Def. 15 is well-defined and unique.

Convention 4 Throughout this thesis, we implicitly assume every open net to be sequentially communicating. This is not a restriction, as every open net can be transformed into a sequentially communicating open net [153] such that the language of its inner net is preserved. \diamond

Example 16 Figure 21 shows the inner nets of the open nets S and C in Fig. 18. Both inner nets are τ -free and—as S and C —bounded. Boundedness is not always preserved. Assume the open net S with an additional internal place x and an additional arc $(process, x)$, yielding the open net N in Fig. 22a. The open net N is bounded because the input place r is never marked and transition $process$ can never fire. In contrast, its inner net $inner(N)$, which we depict in Fig. 22b, is unbounded because place r is removed and, therefore, $process$ may fire infinitely many times. \diamond

2.5 OPEN NET ENVIRONMENTS AND THEIR COMPOSITION

In the first part of this thesis, we shall define different denotational semantics for open nets to characterize certain relations between open nets. We define these semantics upon the notion of traces of labeled nets—that is,

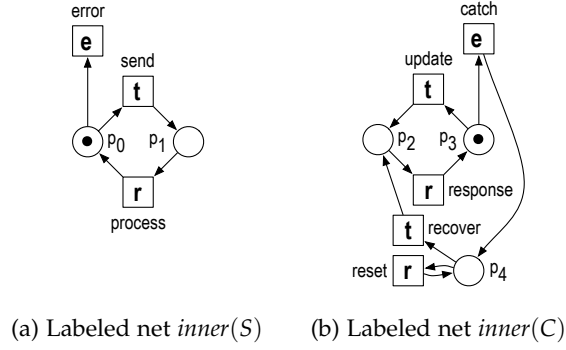


Figure 21: The inner nets of the two open nets S and C from Fig. 18. In addition to the figures, we have $\Omega_{inner(S)} = \{\{\}\}$ and $\Omega_{inner(C)} = \{[p_3]\}$.

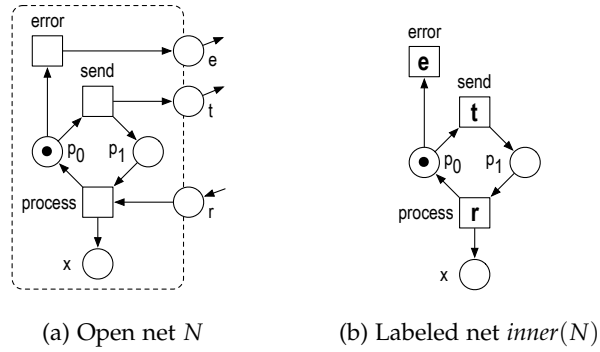


Figure 22: The open net N as a modification of the open net S from Fig. 18a, and its inner net $inner(N)$. In addition to the figures, we have $\Omega_N = \Omega_{inner(N)} = \{\{\}\}$.

we define trace-based semantics. To give an open net N a trace-based semantics, we consider its environment $env(N)$, which we define similarly to Vogler [246]. The net $env(N)$ can be constructed from N by adding to each interface place $p \in I$ ($p \in O$) a p -labeled transition p in $env(N)$ and renaming the place p to p^i (p^o). The net $env(N)$ shows the possible behavior of an environment of N —that is, which inputs it can send to N and which outputs it can receive from N . It is just a tool to define our characterizations and prove our results. But intuitively, one can understand the construction as translating the asynchronous interface p of N into a buffered synchronous interface (with unbounded buffers p^i or p^o) described by the transition labels of $env(N)$.

Definition 17 [open net environment]

The *environment* of an open net N is the labeled net $env(N) = (P \uplus P^I \uplus P^O, T \uplus I \uplus O, F', m_N, \Omega, I, O, l)$, where

- $P^I = \{p^i \mid p \in I\}$,
- $P^O = \{p^o \mid p \in O\}$,
- $F' = ((P \uplus T) \times (T \uplus P)) \cap F$
 $\uplus \{(p^i, t) \mid p \in I, t \in T, (p, t) \in F\}$
 $\uplus \{(t, p^o) \mid p \in O, t \in T, (t, p) \in F\}$
 $\uplus \{(p^o, p) \mid p \in O\}$
 $\uplus \{(p, p^i) \mid p \in I\}$, and

$$\bullet l(t) = \begin{cases} t, & t \in I \uplus O \\ \tau, & \text{otherwise.} \end{cases}$$

Comparing N and $env(N)$ we see that the construction of $env(N)$ changes the set of places of N . Every interface place p of N is renamed to a place p^i or p^o . To keep things simple, we abstract from this difference and do not distinguish between markings of N and $env(N)$.

Convention 5 Throughout this thesis, we implicitly identify a marking of N with the corresponding marking of $env(N)$, and vice versa. \diamond

As already mentioned at the beginning of this section, the environment of an open net N is a tool to give N a semantics based on traces. To shorten our notation, we shall define this semantics for labeled nets and implicitly extend it to open nets using their environment.

Convention 6 Throughout this thesis, each trace set and semantics for labeled nets are implicitly extended to any open net N via $env(N)$ —for example, the *language* of N is defined as $L(N) = L(env(N))$. \diamond

Example 18 Figure 23 shows the environments of the open nets S and C in Fig. 18. Each interface place of the open nets S and C is now a transition of the labeled nets $env(S)$ and $env(C)$, respectively. In contrast to the inner nets $inner(S)$ and $inner(C)$ in Fig. 21, the labeled nets $env(S)$ and $env(C)$ are neither τ -free nor bounded: For example, place r^i is unbounded in $env(S)$ and place e^i is unbounded in $env(C)$. Clearly, we can relate the firing $m \xrightarrow{t} m'$ of a transition t of $inner(S)$ from the same marking m in $env(S)$: If t is connected to an output place p in S , then $m \xrightarrow{t} m + [p^o] \xrightarrow{p} m'$ in $env(S)$, if t is connected to an input place p in S , then $m \xrightarrow{p} m + [p^i] \xrightarrow{t} m'$ in $env(S)$. For example, we have $[p_0] \xrightarrow{send} [p_1]$ in $inner(S)$ and $[p_0] \xrightarrow{send} [p_1, t^o] \xrightarrow{t} [p_1]$ in $env(S)$. Therefore, every trace of $inner(S)$ is also a trace of $env(S)$: For example, we can simulate the trace tr of $inner(S)$ with its underlying run $[p_0] \xrightarrow{send} [p_1] \xrightarrow{process} [p_0]$ by the run $[p_0] \xrightarrow{send} [p_1, t^o] \xrightarrow{t} [p_1] \xrightarrow{r} [p_1, r^o] \xrightarrow{process} [p_0]$ in $env(S)$, yielding the trace tr of $env(S)$. However, the converse does not hold: For example, the trace r is a trace of $env(S)$ but not a trace of $inner(S)$. The language of $env(S)$ (and, by Conv. 6, of S) is

$$L(env(S)) = \{w \in \{r, t\}^* \mid \forall v \sqsubseteq w : |v|_t \leq |v|_r + 1\} \\ \uplus \{wez \mid w, z \in \{r, t\}^* \wedge \forall v \sqsubseteq w : |v|_t \leq |v|_r \wedge |wz|_t \leq |w|_r\},$$

and the language of $env(C)$ (and, by Conv. 6, of C) is

$$L(env(C)) = \{w \in \{r, t\}^* \mid \forall v \sqsubseteq w : |v|_t \geq |v|_r\} \\ \uplus \{wez \mid w \in \{r, t\}^* \wedge z \in \{e, r, t\}^* \wedge \forall v \sqsubseteq w : |v|_t \geq |v|_r\}.$$

By the argumentation above, we have $L(inner(S)) \subseteq L(env(S))$ and $L(inner(C)) \subseteq L(env(C))$. \diamond

We can generalize the observations from Ex. 18 as follows: If an open net N has at least one transition, then the environment $env(N)$ is not τ -free. If an open net N is not a closed net, then the environment $env(N)$ is unbounded

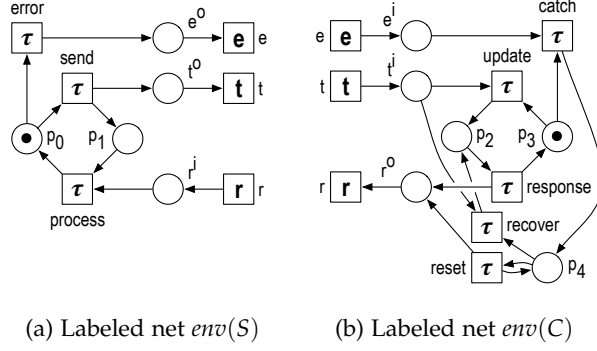


Figure 23: The environments of the open nets S and C from Fig. 18. In addition to the figures, we have $\Omega_{env(S)} = \{\{\}\}$ and $\Omega_{env(C)} = \{\{p_3\}\}$.

because all former input places of N are unbounded and there exists at least one such input place. By comparing the inner net with the environment of an open net N , we state another fact: Every trace of $inner(N)$ is a trace of $env(N)$, thus $L(inner(N)) \subseteq L(env(N))$.

To compose environments of composable open nets in particular and labeled nets in general, we define a parallel composition operator \parallel where, for each action a that the components have in common, each a -labeled transition of one component is synchronized with each a -labeled transition of the other. In addition, we define a second parallel composition operator \uparrow . This operator works as operator \parallel and, in addition, hides all common actions—that is, changes the respective labels to τ . Hiding and \parallel are defined as in Vogler [246]. For merely technical reasons, we consider transition labels of transitions that are synchronized by \parallel as output actions.

Definition 19 [parallel composition and hiding]

Two labeled nets N_1 and N_2 are *composable* if $P_1 \cap P_2 = \Sigma_1^{in} \cap \Sigma_2^{in} = \Sigma_1^{out} \cap \Sigma_2^{out} = \emptyset$. The *parallel composition* of two composable labeled nets is the labeled net $N_1 \parallel N_2 = (P, T, F, m_N, \Omega, \Sigma^{in}, \Sigma^{out}, l)$, where

- $P = P_1 \uplus P_2$,
- $T = \{(t_1, t_2) \in T_1 \times T_2 \mid l_1(t_1) = l_2(t_2) \neq \tau\} \uplus \{(t_1, \tau) \in T_1 \times \{\tau\} \mid \forall t_2 \in T_2 : l_2(t_2) \neq l_1(t_1)\} \uplus \{(\tau, t_2) \in \{\tau\} \times T_2 \mid \forall t_1 \in T_1 : l_1(t_1) \neq l_2(t_2)\}$,
- $F = \{(p, (t_1, t_2)) \in P \times T \mid (p, t_1) \in F_1 \vee (p, t_2) \in F_2\} \uplus \{((t_1, t_2), p) \in T \times P \mid (t_1, p) \in F_1 \vee (t_2, p) \in F_2\}$,
- $m_N = m_{N_1} + m_{N_2}$,
- $\Omega = \{m_1 + m_2 \mid m_1 \in \Omega_1, m_2 \in \Omega_2\}$,
- $\Sigma^{in} = (\Sigma_1^{in} \uplus \Sigma_2^{in}) \setminus (\Sigma_1^{out} \uplus \Sigma_2^{out})$,
- $\Sigma^{out} = \Sigma_1^{out} \uplus \Sigma_2^{out}$, and
- $l(t_1, t_2) = \begin{cases} l_1(t_1), & t_1 \in T_1 \\ l_2(t_2), & \text{otherwise.} \end{cases}$

For a labeled net N and a set $A \subseteq \Sigma$, we obtain N/A from N by *hiding* all actions of A , meaning we replace the respective labels in A with τ and

remove A from the alphabet of N/A . For N/A and a word $w \in \Sigma^*$, $\phi(w)$ denotes $w|_{\Sigma \setminus A}$. We canonically extend the notion of $\phi(w)$ pointwise to sets of words.

We define the *parallel composition with hiding* as the labeled net $N_1 \uparrow N_2 = (N_1 \parallel N_2) / (\Sigma_1 \cap \Sigma_2)$.

Figure 24 extends Fig. 19 and gives a schematic example for the parallel composition of two labeled nets with and without hiding. The labeled nets $env(N_1)$ and $env(N_2)$ are composable because they coincide only on the transitions x and y . The compositions $env(N_1) \parallel env(N_2)$ and $env(N_1) \uparrow env(N_2)$ are again labeled nets. In $env(N_1) \parallel env(N_2)$, the transitions x and y are still labeled with x and y , respectively, but in $env(N_1) \uparrow env(N_2)$ the label of both transitions has been replaced by τ .

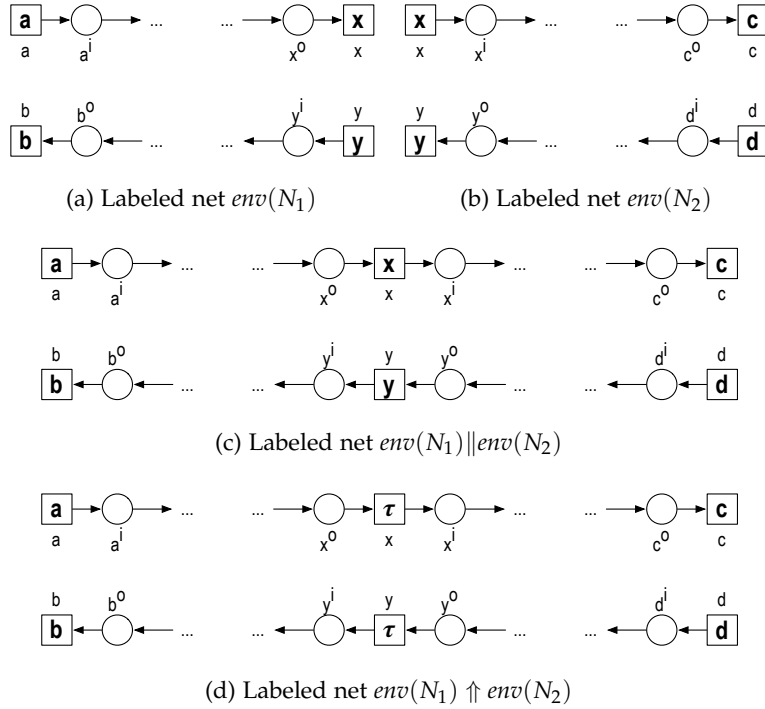


Figure 24: Schematic example of the environments $env(N_1)$ and $env(N_2)$ of two open nets N_1 and N_2 (as sketched in Fig. 19), and their parallel composition with and without hiding.

Example 20 Figure 25 illustrates the parallel composition with and without hiding of the environments $env(S)$ and $env(C)$ of the open nets S and C from Fig. 23. The language of $env(S) \parallel env(C)$ is

$$\begin{aligned}
 L(env(S) \parallel env(C)) &= \{tr\}^* \\
 &\quad \uplus \{wt \mid w \in \{tr\}^*\} \\
 &\quad \uplus \{wez \mid w \in \{tr\}^* \wedge z \in \{r\}^*\}.
 \end{aligned}$$

In contrast, the language of $env(S) \uparrow env(C)$ is $L(env(S) \uparrow env(C)) = \{\varepsilon\}$, because $S \oplus C$ is a closed net and, thus, all transitions of $env(S) \uparrow env(C)$ are labeled with τ . \diamond

To describe the behavior of compositions, we define parallel compositions of words and languages; operator \parallel synchronizes common actions, operator

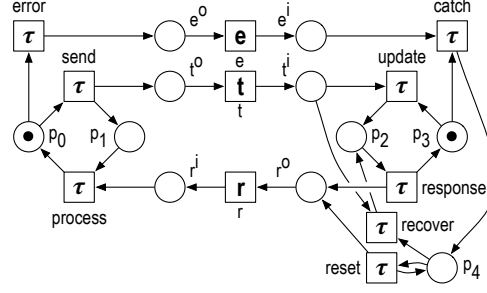
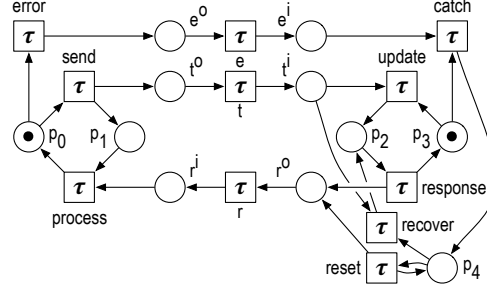
(a) Labeled net $env(S) \parallel env(C)$ (b) Labeled net $env(S) \uparrow env(C)$

Figure 25: The parallel composition with and without hiding of the environments of the two open nets S and C from Fig. 23. In addition to the figures, we have $\Omega_{env(S) \parallel env(C)} = \Omega_{env(S) \uparrow env(C)} = \{[p_3]\}$.

\uparrow also hides them. Observe that in $env(N_1) \uparrow env(N_2)$ only common transitions are merged; operator \parallel is needed to relate the respective transition sequences.

Definition 21

Let Σ_1, Σ_2 be alphabets and $\Sigma = (\Sigma_1 \cup \Sigma_2) \setminus (\Sigma_1 \cap \Sigma_2)$. Let $w_1 \in \Sigma_1^*$ and $w_2 \in \Sigma_2^*$ be words, and let $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be languages. We define

- $w_1 \parallel w_2 = \{w \in (\Sigma_1 \cup \Sigma_2)^* \mid w|_{\Sigma_1} = w_1, w|_{\Sigma_2} = w_2\}$,
- $w_1 \uparrow w_2 = \{w|_{\Sigma} \mid w \in w_1 \parallel w_2\}$,
- $L_1 \parallel L_2 = \cup\{w_1 \parallel w_2 \mid w_1 \in L_1, w_2 \in L_2\}$, and
- $L_1 \uparrow L_2 = \cup\{w_1 \uparrow w_2 \mid w_1 \in L_1, w_2 \in L_2\}$.

Convention 7 To simplify the notation in Def. 21, we do not add the alphabets of w_1 and w_2 to the operators \parallel and \uparrow ; the alphabets will be always clear from the context. \diamond

Example 22 Consider again the environments $env(S)$ and $env(C)$ in Fig. 23. We stated the languages of $env(S)$ and $env(C)$ in Ex. 18. For example, we have $A = \{w \in \{r, t\}^* \mid \forall v \sqsubseteq w : |v|_t \leq |v|_r + 1\} \subseteq L(env(S))$ and $B = \{w \in \{r, t\}^* \mid \forall v \sqsubseteq w : |v|_t \geq |v|_r\} \subseteq L(env(C))$, thus $A \parallel B = \{w \in \{r, t\}^* \mid \forall v \sqsubseteq w : |v|_r \leq |v|_t \leq |v|_r + 1\} \subseteq L(env(S) \parallel env(C))$. Because

$\Sigma_{env(S)} = \Sigma_{env(C)}$, we have $A \uparrow B = \{\varepsilon\}$. In other words, for two languages A and B over the same alphabet, $A \parallel B$ collapses to $A \cap B$, and $A \uparrow B = \{\varepsilon\}$ if $A \cap B \neq \emptyset$; otherwise, $A \uparrow B = \emptyset$. Therefore,

$$\begin{aligned} L(env(S)) \parallel L(env(C)) &= \{tr\}^* \\ &\quad \uplus \{wt \mid w \in \{tr\}^*\} \\ &\quad \uplus \{wez \mid w \in \{tr\}^* \wedge z \in \{r\}^*\} \end{aligned}$$

and $L(env(S)) \uparrow L(env(C)) = \{\varepsilon\}$. \diamond

2.6 RELATING OPEN NETS AND OPEN NET ENVIRONMENTS

As already mentioned at the beginning of Sect. 2.5, we shall define certain trace-based semantics for open nets in the first part of this thesis. Here, it is vital to infer the semantics of the composition $N_1 \oplus N_2$ of two composable open nets N_1 and N_2 from the semantics of N_1 and N_2 . In this section, we lay the foundation for this: We relate traces of the environment $env(N_1 \oplus N_2)$ of N_1 and N_2 to traces of the individual environments $env(N_1)$ and $env(N_2)$. To this end, we first relate traces of the individual environments $env(N_1)$ and $env(N_2)$ to traces of the parallel composition $env(N_1) \parallel env(N_2)$. In a second step, we relate traces of the parallel composition $env(N_1) \parallel env(N_2)$ (and $env(N_1) \uparrow env(N_2)$) to traces of $env(N_1 \oplus N_2)$. The second step is particularly elegant, because $env(N_1) \parallel env(N_2)$ differs from $env(N_1 \oplus N_2)$ in its hidden common actions and transitions that connect former overlapping interface places of N_1 and N_2 .

We start with the first step by recalling [246, Theorem 3.1.7(4)], which relates a trace of a composed labeled net to traces of its components.

Proposition 23 [[246]]

For two markings m_1 and m_2 of composable labeled nets N_1 and N_2 , respectively, we have $m_1 + m_2 \xrightarrow{w} m'_1 + m'_2$ in $N_1 \parallel N_2$ iff there exist $w_1 \in \Sigma_{N_1}^*$, $w_2 \in \Sigma_{N_2}^*$ such that $w \in w_1 \parallel w_2$, $m_1 \xrightarrow{w_1} m'_1$ in N_1 , and $m_2 \xrightarrow{w_2} m'_2$ in N_2 .

Example 24 Consider the labeled net $env(S) \parallel env(C)$ in Fig. 25a. We have $\Sigma_{env(S)} = \Sigma_{env(C)}$, thus, Prop. 23 implies $L(env(S) \parallel env(C)) = L(env(S)) \parallel L(env(C))$. Therefore, we can confirm our characterization of $L(env(S) \parallel env(C)) = L(env(S)) \parallel L(env(C)) = \{tr\}^* \uplus \{wt \mid w \in \{tr\}^*\} \uplus \{wez \mid w \in \{tr\}^* \wedge z \in \{r\}^*\}$ from Ex. 20 and Ex. 22. \diamond

With Prop. 23, we can compute the language of the parallel composition $env(N_1) \parallel env(N_2)$ of two composable labeled nets N_1 and N_2 from the languages of $env(N_1)$ and $env(N_2)$. Next, we relate markings of the environment of the composition $env(N_1 \oplus N_2)$ to markings of the parallel composition of the environments $env(N_1) \parallel env(N_2)$ of N_1 and N_2 (Def. 25). Then, we use this relation to relate transition firings (Prop. 27), transition sequences (Prop. 28), and traces (Lem. 30).

For the remainder of this section, we fix two composable open nets N_1 and N_2 , and we put $C = env(N_1 \oplus N_2)$, $E = env(N_1) \parallel env(N_2)$, and $\bar{E} = env(N_1) \uparrow env(N_2)$. The labeled nets \bar{E} and E differ only in their labelings; C and \bar{E} (E) have the same places, except for places $p \in (I_1 \cap O_2) \uplus (I_2 \cap O_1)$ in C and the corresponding places p^i , p^o in \bar{E} (E). We study the relation

between reachable markings of different compositions of N_1 and N_2 . To this end, we define *agreement* between markings.

Definition 25 [agreement]

A marking m of C and a marking \bar{m} of \bar{E} (of E) *agree* if they coincide on the common places and if for each $p \in (I_1 \cap O_2) \uplus (I_2 \cap O_1)$, $\bar{m}(p^o) + \bar{m}(p^i) = m(p)$. They *strongly agree* if, additionally, $\bar{m}(p^o) = 0$.

Example 26 Because $S \oplus C$ is a closed net, Fig. 20 depicts the labeled net $env(S \oplus C)$ if we ignore the dashed frame and assume every transition to be labeled by τ . Comparing $env(S \oplus C)$ and the labeled net $env(S) \parallel env(C)$ in Fig. 25a, we see that the markings $[p_1, t, p_3]$ of $env(S \oplus C)$ and $[p_1, t^o, p_3]$ of $env(S) \parallel env(C)$ agree, and the markings $[p_1, t, p_3]$ of $env(S \oplus C)$ and $[p_1, t^i, p_3]$ of $env(S) \parallel env(C)$ even strongly agree. \diamond

Agreeing markings of C and \bar{E} permit the firing of a transition of C in both nets. We relate the firing of a transition of C and \bar{E} (E) by recalling [228, Lemma 15].

Proposition 27 [[228]]

Let the markings m of C and \bar{m} of \bar{E} agree, and let $t \in T_C$. Then

1. If $\bar{m} \xrightarrow{t} \bar{m}'$, then $m \xrightarrow{t} m'$ such that m' and \bar{m}' agree.
2. If m and \bar{m} strongly agree, all additional transitions of \bar{E} are disabled at \bar{m} , and further $m \xrightarrow{t} m'$ implies $\bar{m} \xrightarrow{t} \bar{m}'$ such that m' and \bar{m}' agree.

As \bar{E} and E differ only in their labelings, (1) and (2) also hold for E in place of \bar{E} .

The next proposition recalls [228, Lemma 16(1),(2)] and [228, Lemma 15(4)]. It states facts about sequences of transitions of C and \bar{E} and relates final markings. The intuitive idea is to extend Prop. 27 from a transition firing to firing sequences, always enforcing (strong) agreement between markings firing the “intermediate” transitions of \bar{E} in between.

Proposition 28 [[228]]

Let m be a marking of C and \bar{m} be one of \bar{E} . Then

1. If m and \bar{m} strongly agree and $m \xrightarrow{v} m'$ in C , then it is possible to insert transitions from $(I_1 \cap O_2) \uplus (I_2 \cap O_1)$ of \bar{E} into v such that for the resulting v' : $\bar{m} \xrightarrow{v'} \bar{m}'$ in \bar{E} and also m' and \bar{m}' strongly agree.
2. If m and \bar{m} agree and $\bar{m} \xrightarrow{v'} \bar{m}'$ in \bar{E} , then it is possible to delete transitions from $(I_1 \cap O_2) \uplus (I_2 \cap O_1)$ of \bar{E} in v' such that for the resulting v : $m \xrightarrow{v} m'$ in C and also m' and \bar{m}' agree.
3. $m \in \Omega_C$ iff $\bar{m} \in \Omega_{\bar{E}}$ iff for $i = 1, 2$: $\bar{m}|_{p_{env(N_i)}} \in \Omega_{env(N_i)}$.

As \bar{E} and E differ only in their labelings, (1), (2), and (3) also hold for E in place of \bar{E} .

Example 29 To illustrate Prop. 28(1), we consider the strongly agreeing markings $[p_1, t, p_3]$ of $\text{env}(S \oplus C)$ in Fig. 20 and $[p_1, t^i, p_3]$ of $\text{env}(S) \parallel \text{env}(C)$ in Fig. 25a. We have $[p_1, t, p_3] \xrightarrow{\text{update}} [p_1, p_2] \xrightarrow{\text{response}} [p_1, r, p_3]$ in $\text{env}(S \oplus C)$. We can insert transition r (which is an interface place of S and C) and obtain $[p_1, t^i, p_3] \xrightarrow{\text{update}} [p_1, p_2] \xrightarrow{\text{response}} [p_1, r^o, p_3] \xrightarrow{r} [p_1, r^i, p_3]$ in $\text{env}(S) \parallel \text{env}(C)$, and $[p_1, r, p_3]$ and $[p_1, r^i, p_3]$ strongly agree. For Prop. 28(2), we consider the agreeing markings $[p_1, t, p_3]$ and $[p_1, t^o, p_3]$ and $[p_1, t^o, p_3] \xrightarrow{t} [p_1, t^i, p_3] \xrightarrow{\text{update}} [p_1, p_2] \xrightarrow{\text{response}} [p_1, r^o, p_3]$ in $\text{env}(S) \parallel \text{env}(C)$. We can delete transition t (which is an interface place of S and C) and obtain $[p_1, t, p_3] \xrightarrow{\text{update}} [p_1, p_2] \xrightarrow{\text{response}} [p_1, r, p_3]$ in $\text{env}(S \oplus C)$, and $[p_1, r^o, p_3]$ and $[p_1, r, p_3]$ agree. \diamond

We have seen that agreement closely relates markings of C and \bar{E} . The following lemma justifies this by showing that agreement between markings of C and \bar{E} is a weak bisimulation.

Lemma 30 [agreement is a weak bisimulation]

The labeled nets C and \bar{E} are weakly bisimilar due to the agreement relation.

Proof. First, we show that if we label each transition of C with itself in C and \bar{E} and every transition $t \in (I_1 \setminus O_2) \uplus (I_2 \setminus O_1)$ in \bar{E} with τ , then agreement between the markings of C and \bar{E} is a weak bisimulation.

The initial markings m_C and $m_{\bar{E}}$ strongly agree by Def. 13 and Def. 17. Writing ϱ for the agreement relation, we now assume that $(m, \bar{m}) \in \varrho$. To prove that ϱ is a weak bisimulation, we have to show that

1. If $m \xrightarrow{t} m'$, then there exists \bar{m}' such that $\bar{m} \xRightarrow{t} \bar{m}'$ and $(m', \bar{m}') \in \varrho$; and
2. If $\bar{m} \xrightarrow{t} \bar{m}'$, then there exists m' such that $m \xRightarrow{t} m'$ and $(m', \bar{m}') \in \varrho$.

Consider the first item. By firing all τ -labeled transitions of \bar{E} that are enabled at \bar{m} , we can empty each place p^o while shifting the tokens to the respective place p^i . Let \bar{m}'' be the resulting marking of \bar{E} . Then $\bar{m} \xRightarrow{\varepsilon} \bar{m}''$ and \bar{m}'' strongly agrees with m , because firing a τ -labeled transition does not change the marking on the common places and no place p^o is marked now. By Prop. 27(2), we have $\bar{m}'' \xrightarrow{t} \bar{m}'$ such that m' and \bar{m}' agree.

For the second item, we can set $m = m'$ if t is τ -labeled and, clearly, m' and \bar{m}' agree then. Otherwise, we can conclude from Prop. 27(1) that marking m' exists such that m' and \bar{m}' agree. Thus, $m \xRightarrow{t} m'$ and ϱ is a weak bisimulation.

Because the original labelings can be obtained by the same relabeling from the labelings considered in the first part of the proof, agreement is also a weak bisimulation for C and \bar{E} . \square

2.7 CONCLUSIONS AND RELATED WORK

In this section, we provided basic mathematical notions that we use throughout this thesis. We introduced LTSs as a uniform formalism for modeling the behavior of distributed and open systems. We introduced open nets—a variant of Petri nets—to model the communication protocol of open systems.

Using the notion of (the reachability graph of) an open net environment, we can easily translate an open net into an LTS. In other words, the environment of an open net links that net to LTSs.

As already detailed in Chap. 1, we investigate interacting open systems that are executed concurrently on different machines. There exist various formalisms to model the communication protocol [174, 37] of an open system. We can roughly distinguish these formalisms by the employed communication model [172, 134, 150] that primarily distinguishes between *synchronous* and *asynchronous* communication. In the case of synchronous communication, messages between systems are neither pending nor buffered, because message exchange is assumed to be instantaneous. This type of communication is sometimes called “handshaking” [34]. In the case of asynchronous communication, a message sent from one system is buffered until it is received by another system. We can, additionally, distinguish asynchronous communication models by the ordering, capacity, quantity, and lossiness of the employed buffer(s).

In this thesis, we consider asynchronous communication between open systems over unbounded, unordered, and lossless buffers. This communication model is most general except for lossiness. Asynchronous communication allows for a more autonomous execution of the composed systems than synchronous communication, because the sending and receiving of a message are decoupled. Therefore, asynchronous communication naturally supports the distributed setting of interacting open systems [172]. On the downside, asynchronous communication requires the modeling of intermediate states of a composition, yielding an increased complexity. Bultan et al. [56] show that many behavioral correctness notions are much harder to decide in the case of asynchronous communication models. Fu et al. [99] investigate necessary conditions for synchronizability—that is, conditions for the possibility to abstract from message channels in asynchronously communicating systems without effecting their behavior, effectively translating them into synchronously communicating systems. Basu et al. [31] identify a subclass of open systems for which an asynchronous communication schema over unbounded First-In-First-Out queues can be replaced by synchronous communication while preserving reachability properties over output actions and states with no pending inputs. Such techniques may lead to faster decision procedures for subclasses of open systems. The converse of synchronizability—that is, translating synchronous to asynchronous communication—is for example studied by Decker et al. [74].

In the following, we briefly review formalism that are used to reason about the behavior of interacting open systems.

2.7.1 Formalism based on process algebras

Many articles on the behavior of concurrent systems are based on process-algebraic concepts [244]. Process algebra is the study of the behavior of parallel or distributed systems by algebraic means [27]; well-known process algebras include CCS [177], CSP [119], and ACP [26]. The process algebras CCS, CSP, and ACP assume that processes interact by means of synchronous communication. However, there also exist process algebras using asynchronous communication. For example, Bergstra et al. [34] and de Boer et al. [39] investigate process algebras with asynchronous communication using ordered (i.e., queues) and unordered (i.e., bags) message channels. Their un-

derlying semantics is that of process graphs (i.e., labeled transition systems). Fournet et al. [97] consider CCS processes of asynchronous message passing software components with unbounded message channels. Again, their operational semantics is that of labeled transition systems. Bravetti and Zavattaro [46] model the behavior of an asynchronously communicating open system with an extension of basic CCS and one unbounded but ordered message queue. Another process algebra for asynchronous communication is the asynchronous variant of the π -calculus [206], where processes interact by sending communication links to each other. The asynchronous variant was first proposed by Honda and Tokoro [121, 122] and, independently, by Boudol [41].

2.7.2 Formalism based on automata

I/O automata [159] and interface automata [18] are automaton models [224] for the behavior and the composition of systems based on synchronous communication. For asynchronous communication, communicating finite-state machines [42] model systems that exchange messages via unbounded queues. Here, each message channel between two systems is represented by one unbounded buffer that preserves the order of sent messages. Berardi et al. [33] and Calvanese et al. [57] model open systems as Mealy finite state machines [170] but do not take any concept of message queuing into account. Hull et al. [124] propose to model services with Mealy machines that communicate asynchronously over bounded or unbounded queues very similar to [42].

Alur et al. [20] use concurrent automata that communicate asynchronously over unbounded unordered buffers to model the behavior of components. Service automata [166] are a simplification of I/O automata which have been used to model service behavior. In contrast to I/O automata, service automata communicate asynchronously and do not require the explicit modeling of the states of message channels. A detailed comparison of service automata with other automata models can be found in [165]. Both concurrent automata and service automata make no assumptions about the infrastructure other than messages do not get lost.

2.7.3 Formalism based on Petri nets

The formalism of open nets [167, 226] is a variant of Petri nets [216, 194], whose modular construction traditionally supports asynchronous communication over unbounded unordered buffers by fusing places [246]. Open nets have been introduced as open workflow nets in [166, 153] and generalize workflow nets from Van der Aalst [1]. Workflow nets have been proven successful for the modeling of business processes and workflows; for distributed business processes, workflow nets have been enriched by interface places to asynchronously communicating workflow modules [163]. Open nets can be composed, yielding a new open net (possibly a closed net) that models the composition of the represented open systems. This idea is based on the module concept for Petri nets which was proposed by Kindler [136].

2.7.4 Why do we chose open nets?

As can be seen from the above, there exist a plethora of formalisms to model the behavior of asynchronously communicating open systems. The

actual choice of a formalism is negligible as long as it supports the most general communication model—that is, asynchronous communication over unbounded unordered buffers.

We have chosen open nets as a formal model for (the behavior of) open systems because they adequately model asynchronous communication over unbounded unordered buffers [246, 153, 226]. They—like service automata and concurrent automata—make no assumptions about the infrastructure or underlying middleware other than messages do not get lost. Open nets have already been proven useful for modeling and verifying the behavior of open systems [166, 151]. In contrast to the approaches of Massuthe et al. [166, 165], Lohmann [151], and Parnjai [203], we do not translate an open net into an LTS for verification purposes, but operate directly on open nets using the notion of their environment. All our notions for open nets in the subsequent chapters, for example a denotational semantics, are essentially grounded in LTSs. However, we still use open nets instead of the underlying LTSs because open nets allow, in general, for more compact models and an easier notion of composition by fusing interface places.

In practices, open systems are usually not modeled as open nets. However, open systems that are specified in industrial languages, such as WS-BPEL [130] or BPMN [63], can be translated into our formal model and then be analyzed [149, 154]. Lohmann [149] presents a feature-complete open net semantics for WS-BPEL and the compiler BPEL2OWFN that automatically translates a WS-BPEL process into an open net. The translation of BPMN into Petri nets [79] is also supported by tools. In addition, there exist approaches for open systems in the form of services: A service description in PHP [208] or C [135] can be translated into an automata [223, 222] using techniques from the areas of model checking [28] and static program analysis [198]. Automata, in turn, can be translated into Petri nets [25], e.g., using state-based [77, 215] or language-based regions [157].

IN this thesis, we aim to verify responsiveness for open systems. Responsiveness is a behavioral correctness criterion for two interacting open systems. It ensures that *termination* of their composition or *communication* between the two open systems is always possible. In other words, responsiveness combines termination (which is important for all systems) with interaction (which is especially important for interacting open systems), as both termination and interaction are too strict for open systems in isolation.

In this chapter, we formalize responsiveness in our modeling formalism. We formalize two variants of responsiveness: The basic variant is responsiveness as the perpetual possibility to either terminate or communicate in the composition of two open nets. As a second variant, we introduce *b*-responsiveness that combines responsiveness and *b*-boundedness. The introduction of *b*-responsiveness is motivated by some undecidability results for responsiveness, which we present in more detail in Chap. 4.

The goal of this thesis is to develop algorithms to verify responsiveness for open systems by means of conformance checking. A conformance relation between two formal models is the central theme in conformance checking. In this chapter, we formalize the conformance relation for responsiveness, called conformance, and the conformance relation for *b*-responsiveness, called *b*-conformance, for open nets. To this end, we define two open nets to be *partners* if they behave desirably (i.e., responsively or *b*-responsively) in composition. Intuitively, an implementation conforms to the specification if the first interacts desirably with at least all partners of the latter—or even more. Therefore, we define conformance as the set of partners of the implementation superseding the set of partners of the specification. Figure 26 illustrates the formalization of conformance as partner inclusion.

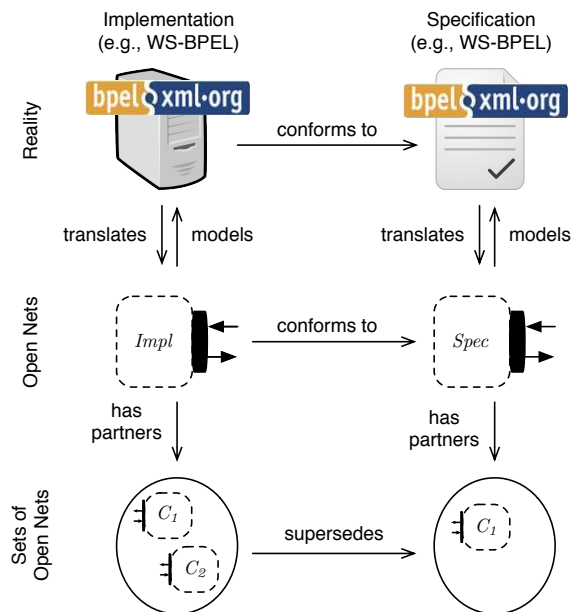


Figure 26: Formalizing conformance as partner inclusion.

Compositionality for open systems allows open systems to be composed from smaller ones; compositional conformance checking allows to infer conformance of a composition by checking conformance of the composed systems. For compositional conformance checking, we additionally require the conformance relation to be compositional. Technically, a compositional conformance notion extends the respective conformance relation (i.e., a pre-order) to a precongruence with respect to the open nets composition operator. We require separate notions for a compositional conformance and a compositional b -conformance relation because conformance and b -conformance are not compositional, which we elaborate in more detail in Part II. Therefore, we also introduce the largest (i.e., coarsest) precongruence that is contained in the conformance relation—that is, compositional conformance—and the coarsest precongruence that is contained in the b -conformance relation.

The remainder of this chapter is structured as follows: We formalize responsiveness for open nets and introduce the corresponding relations conformance and compositional conformance in Sect. 3.1. Then, we formalize b -responsiveness and introduce the b -conformance relation and the compositional b -conformance in Sect. 3.2. We classify responsiveness and b -responsiveness into a spectrum of behavioral correctness criteria for open nets in Sect. 3.3. In addition, we relate conformance and b -conformance by showing that they are incomparable. We finish this chapter in Sect. 3.4 with a conclusion and a review of related work on the notion of responsiveness.

3.1 FORMALIZING RESPONSIVENESS AND CONFORMANCE

Responsiveness is the perpetual possibility to either terminate or communicate. Because the behavior of a composition of interacting open systems derives from the interaction between the composed systems, we define responsiveness depending on two open systems in combination: In the composition, each of them will usually not reach all states it could reach in the composition with other open systems. Thus, an open system may be responsive with one open system, but not responsive with another open system.

For our formalization of responsiveness, we interpret communication between two open systems in the same way as (successful) communication between two human beings: One person talks (i.e., sends a message) while the other person does not talk (i.e., receives or ignores this message). Which person talks and which person does not may change at any time, but this change is *not* inevitable. Therefore, even for two nonterminating open systems, responsiveness does not imply a mutual sending of messages: It suffices that just one open system can perpetually send a message to the other open system. This most basic view on communication allows for a finite but unbounded number of send and pending (i.e., unreceived or ignored) messages between two systems, which renders conformance for responsiveness undecidable. We shall define another variant of responsiveness in Sect. 3.2, which—in contrast to conformance in this section—actually implies a mutual sending of messages.

In terms of open nets, sending a message is modeled by firing a transition that puts a token on an output place; communication in turn is always possible if it is always possible to enable such a transition in the composition of two open nets.

Definition 31 [responsiveness]

Let N_1 and N_2 be two composable open nets. A marking m of $N_1 \oplus N_2$ is *responsive* if we can reach from m a marking that enables a transition t with $t^\bullet \cap (O_1 \uplus O_2) \neq \emptyset$, or we can reach a final marking of $N_1 \oplus N_2$ from m . The open nets N_1 and N_2 are *responsive* if their composition $N_1 \oplus N_2$ is a closed net and every reachable marking of $N_1 \oplus N_2$ is responsive.

In the following, we give an example for two responsive open nets.

Example 32 Throughout this thesis, we consider the unreliable time server S and its recovering client C from Fig. 18 as a running example for everything concerning responsiveness. For convenience, we depict them again in Fig. 27. Recall that the open nets S and C are composable (cf. Ex. 14). Their composition $S \oplus C$, which we depict in Fig. 27c, is a closed net, and S and C are responsive: Either they can mutually send a message over the interface places t and r or C repeatedly produces a token on place r after consuming a token from e . The place r in $S \oplus C$ is unbounded; thus, the composition is unbounded, too. In addition, no final marking of $S \oplus C$ is reachable in $S \oplus C$ —that is, termination of $S \oplus C$ is impossible. \diamond

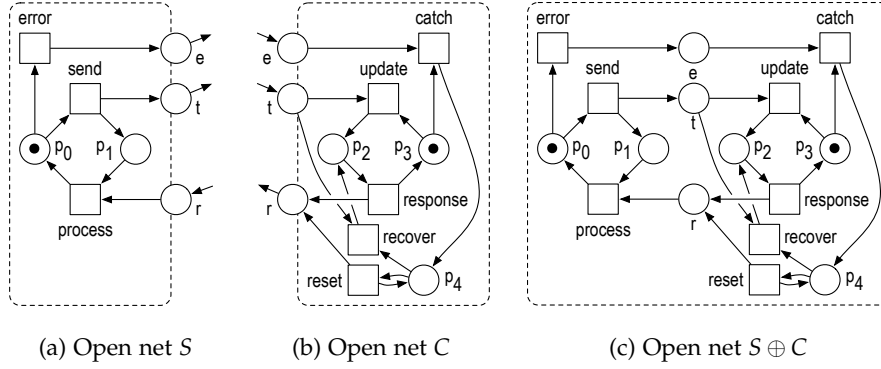


Figure 27: Three open nets modeling an unreliable time server, a client, and their composition. In addition to the figures, we have $\Omega_S = \{\{\}\}$ and $\Omega_C = \Omega_{S \oplus C} = \{[p_3]\}$.

Based on the correctness criterion responsiveness, we define a *partner* of an open net N as an open net C such that N and C are responsive.

Definition 33 [partner]

An open net C is a *partner* of an open net N if N and C are responsive.

For every “truly” (i.e., not closed) open net N , there exists a partner—the latter just has to continuously send a message (like C in Fig. 27b after receiving message e). Continuously sending a message to N is possible for any open net composable with N , because N has at least one input place by Def. 11.

In the following, we give two examples for partners.

Example 34 As already explained in Ex. 32, the open nets S and C from Fig. 27 are composable and responsive. Thus, S is a partner of C , and vice versa. \diamond

Example 35 The open net C' in Fig. 28a represents another client for the unreliable time server S in Fig. 27a. The client C' repeatedly updates its system time and responds with a response packet. However, if the time server sends an error message, C' receives this message (input place e) and—in contrast to the client C in Fig. 27b—terminates (final marking $[p_4]$). The open nets S and C' are composable; their composition $S \oplus C'$ is a closed net and depicted in Fig. 28b.

The open net C' is also a partner of S : The marking $[p_4]$ is the only reachable marking in their composition $S \oplus C'$ that does not enable future communication. However, $[p_4]$ is a final marking of $S \oplus C'$, thus S and C' are responsive. \diamond

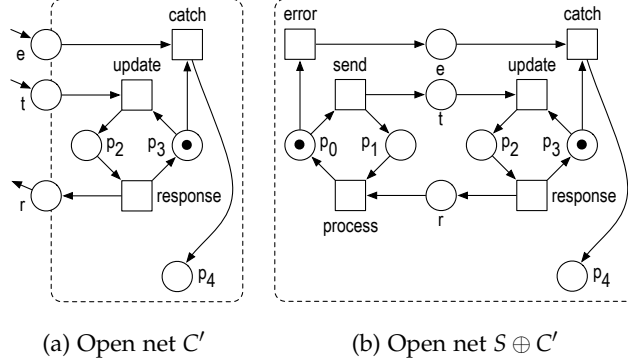


Figure 28: The open net C' modeling a terminating client of the open net S in Fig. 27a, and their composition $S \oplus C'$. In addition to the figures, we have $\Omega_{C'} = \Omega_{S \oplus C'} = \{[p_4]\}$.

If the partners of an open net $Impl$ are a superset of the partners of another open net $Spec$, then $Impl$ can be seen as “more correct” than $Spec$; intuitively, $Impl$ interacts desirably (i.e., responsively) with at least all environments of $Spec$ —or even more. This relation based on partner inclusion between $Impl$ and $Spec$ models the idea of a conformance relation that preserves a given behavioral correctness criterion; an idea we already described in Sect. 1.1.1. We refer to the resulting relation between open nets as *conformance*. Technically, conformance is a preorder over the set of all open systems.

Definition 36 [conformance]

For two interface-equivalent open nets $Impl$ and $Spec$, $Impl$ conforms to $Spec$, denoted by $Impl \sqsubseteq_{conf} Spec$, if for all open nets C the following holds: If C is a partner of $Spec$, then C is also a partner of $Impl$.

For modular reasoning—that is, compositional conformance checking—a conformance relation should be a precongruence with respect to the open net composition operator \oplus (see also Sect. 1.1.2). Because conformance shall turn out not to be a precongruence, we will make it stricter (smaller) as far as needed to obtain such a precongruence, and we already introduce a notation for this largest (i.e., coarsest) precongruence. We refer to the coarsest precongruence that is contained in the conformance relation as *compositional conformance*.

Definition 37 [compositional conformance]

We denote by \sqsubseteq_{conf}^c the largest subset of \sqsubseteq_{conf} such that \sqsubseteq_{conf}^c is a *precongruence* with respect to \oplus . For two interface-equivalent open nets $Impl$ and $Spec$, $Impl$ *compositionally conforms* to $Spec$, if $Impl \sqsubseteq_{conf}^c Spec$.

In the following, we give an example and a counter-example for two conforming open nets.

Example 38 The open net S' in Fig. 29 models a patched time server. It has the same functionality as the open net S in Fig. 27a, but it never sends an error message. In contrast to S , S' never shuts down and is intentionally always running: the only final marking $[\]$ is unreachable.

The open net S' conforms to the open net S : Every partner of S must expect an error from S (i.e., a token on interface place e) and must reach a final marking after catching the error (i.e., consuming the token from e). Additionally, every partner of S must provide a token on r for each token on t ; otherwise, S can get stuck in a nonfinal marking with a token on p_1 . Thus, every partner of S is also a partner of S' , where an error may never happen. \diamond

Example 39 Although the open net S' conforms to the open net S , S does not conform to S' . Assume the open net C' in Fig. 28a, but this time with the empty set of final markings. Clearly, C' is a partner of S' where an error never happens, perpetually communicating over the places t and r . However, in contrast to Ex. 35, C' is not a partner of S because of the changed set of final markings: If S sends a message e , then transition *catch* in C' may fire, yielding the nonresponsive marking $[p_4]$ of their composition $S \oplus C'$. \diamond

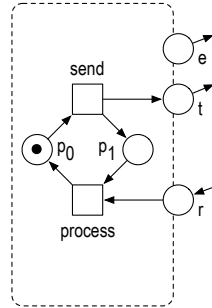


Figure 29: The open net S' models a patched time server. We have $\Omega_{S'} = \{[\]\}$.

We show with the following example that conformance as defined in Def. 36 does not guarantee compositionality, i.e., the preorder \sqsubseteq_{conf} is strictly larger than the precongruence \sqsubseteq_{conf}^c .

Example 40 Consider the open nets X and Y in Fig. 30. The open net X models a client that uses a time server as long as it does not catch an error; if X catches an error, then it immediately switches to another time server (i.e., open net Y).

Although S' conforms to S , as detailed in Ex. 38, $S' \oplus X$ does not conform to $S \oplus X$: The open net Y is a partner of $S \oplus X$ but not of $S' \oplus X$, because the transition *catch* in $S' \oplus X$ can never fire and, thus, firing transition t_2 in Y leads to nonresponsive markings of $(S' \oplus X) \oplus Y$. The only final

marking $[p_5, p_6]$ is not reachable in $(S' \oplus X) \oplus Y$. Therefore, $S' \sqsubseteq_{\text{conf}} S$ but $S' \not\sqsubseteq_{\text{conf}}^c S$. \diamond

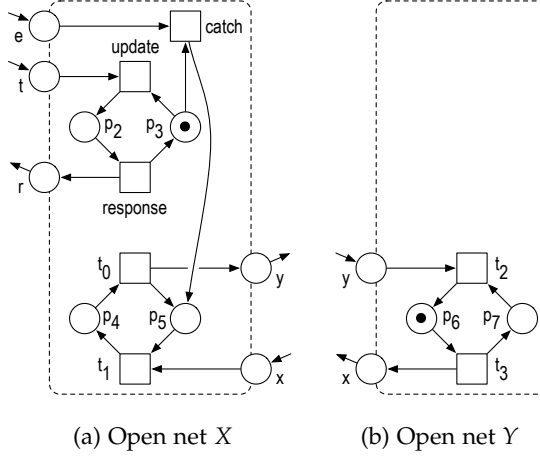


Figure 30: Two open nets proving that conformance is not a precongruence with regard to open net composition \oplus . In addition to the figures, we have $\Omega_X = \{[p_5]\}$ and $\Omega_Y = \{[p_6]\}$.

3.2 FORMALIZING b -RESPONSIVENESS AND b -CONFORMANCE

In the previous section, we formalized the notion of responsiveness and its corresponding conformance and compositional conformance relation. As it turns out in Sect. 4.3, conformance and compositional conformance are undecidable and, thus, not applicable for the verification of open systems. The composition of two responsive open nets may be unbounded, which endangers decidability: Intuitively, we can never know whether the set of partners of an open net $Impl$ supersedes the set of partners of an open net $Spec$ by just examining all partners C of $Spec$ such that $Spec \oplus C$ is b -bounded. There may always exist a partner C' of $Spec$ such that $C' \oplus Spec$ is not b -bounded and C' is no partner of $Impl$, thereby violating conformance of $Impl$ and $Spec$. In other words, checking conformance of $Impl$ and $Spec$ requires to solve a kind of halting problem.

As conformance and compositional conformance turn out to be undecidable, we already introduce another variant of responsiveness in this section: We require the composition of two open nets to be responsive and, additionally, to be b -bounded. We refer to this new notion of responsiveness as b -responsiveness. Recall that throughout this thesis, b denotes a bound (see Conv. 3). The bound b can be determined beforehand by static analysis of the open system's underlying middleware or of the communication behavior of one of the open system, for instance. Therefore, using b -responsiveness instead of responsiveness does not restrict the verification process in practice. As it turns out in Chap. 5 and Chap. 6, the conformance and compositional conformance relations corresponding to b -responsiveness become decidable.

Two open nets are b -responsive if they can terminate, or at least one net can repeatedly interact with the other net while respecting the message bound b . As for responsiveness, also b -responsiveness depends on two open nets in combination: In the composition, each of them will usually not reach

all states it could reach in the composition with another open net. Therefore, two open nets may be b -responsive although they are unbounded in isolation, or the composition with other open nets is unbounded.

Definition 41 [b -responsiveness]

Let N_1 and N_2 be two composable open nets. A marking of $N_1 \oplus N_2$ is b -responsive if it is b -bounded and responsive. The open nets N_1 and N_2 are b -responsive if their composition $N_1 \oplus N_2$ is a closed net and every reachable marking of $N_1 \oplus N_2$ is b -responsive.

Technically, Def. 41 defines a family of behavioral correctness criteria for two open nets: one criterion for each b -value. If two open nets N_1 and N_2 are b -responsive, then N_1 and N_2 are b' -responsive for all $b' \geq b$. To keep things simple (i.e., for smaller state spaces), we usually consider only 1-responsiveness in our examples.

In the following, we give an example for two b -responsive open nets.

Example 42 Figure 31 shows three open systems, each modeled as an open net. The open net D models a database server. After processing a query (input place q), it responds with the retrieved data (output place d). A user may shut down D by sending a shutdown message (input place s). D has the capability to forward a received shutdown message (output place f), which erroneously interferes with its termination (the final marking $[p_0]$ becomes unreachable).

The open net U models a user of the database. The user repeatedly queries the database and analyzes the returned data. U never sends a shutdown message and ignores any forwarded message from D . Throughout this thesis, the open nets D and U serve as a running example for everything concerning b -responsiveness.

The open nets D and U are composable. Their composition $D \oplus U$ is a closed net, which is depicted in Fig. 31c. The open nets D and U are b -responsive because $[p_1, p_4]$, $[p_1, p_3, q]$, $[p_2, p_3]$, and $[p_1, p_3, d]$ are the only reachable markings in $D \oplus U$; all of them are 1-bounded. Observe that this statement holds for any bound b because $D \oplus U$ is 1-bounded. \diamond

Recall that two open nets are b -responsive if they are responsive and their composition is b -bounded. In contrast to responsiveness in Def. 31, we can prove that, due to b -responsiveness, each net always has the chance to send a message (possibly after some messages from the other net), or their composition terminates. In other words, b -responsiveness implies a choice between mutual interaction between the nets, or termination. Thus, the word “responsive” is really justified here.

Proposition 43 [mutual interaction]

Let N_1 and N_2 be two composable open nets such that N_1 and N_2 are b -responsive. Then, from any reachable marking m in $N_1 \oplus N_2$,

1. markings m_1 and m_2 are reachable in $N_1 \oplus N_2$ such that $m_1 \xrightarrow{t_1}$ with $t_1^\bullet \cap O_1 \neq \emptyset$ and $m_2 \xrightarrow{t_2}$ with $t_2^\bullet \cap O_2 \neq \emptyset$, or
2. a final marking of $N_1 \oplus N_2$ is reachable.

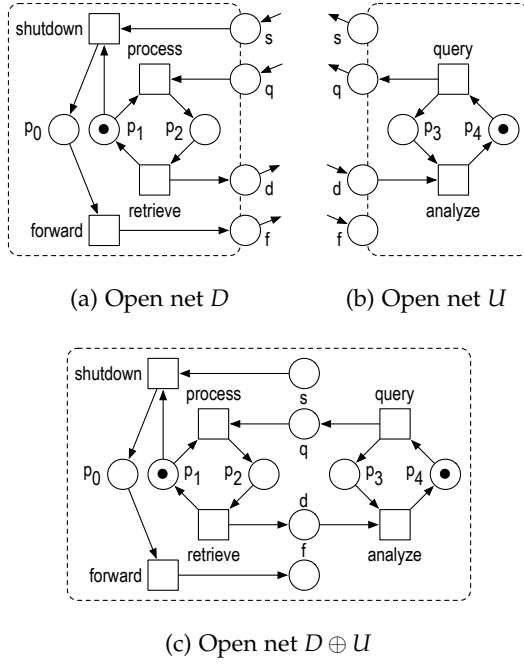


Figure 31: Three open nets modeling a database server, a user, and their composition. In addition to the figures, we have $\Omega_D = \{[p_0]\}$, $\Omega_U = \{[]\}$, and $\Omega_{D \oplus U} = \{[p_0]\}$.

Proof. Assume that there exists a reachable marking m from which no suitable markings m_1 and m_2 are reachable. Then there is a final marking of $N_1 \oplus N_2$ reachable from m by Def. 31.

Now assume that there exists an m from which no final marking of $N_1 \oplus N_2$ is reachable and from which—w.l.o.g.—only a marking m_1 but no m_2 is reachable. Then, in $N_1 \oplus N_2$ there exists a run from m to a marking m_1 enabling some t_1 . No tokens are put onto $I_1 = O_2$ in this run; otherwise, we would have found an m_2 just before such a firing. Hence, no transitions of N_2 are needed to enable t_1 , and we can assume that all transitions of the run belong to N_1 . Consequently, no token is removed from $O_1 = I_2$. Now, we fire t_1 and reach some m' with at least one token more on O_1 . If m' has an m_2 as claimed in the lemma, this can also serve for m as m_2 . Hence, m_2 does not exist, but some m'_1 must, as argued previously. We repeat this argument, and each time the token count on O_1 increases until bound b is violated. However, this contradicts Def. 41, stating that $N_1 \oplus N_2$ is b -bounded. As a consequence, a marking m_2 must be reachable from m . \square

We redefine the notion of a partner from Def. 33 for b -responsiveness to b -partner. As for b -responsiveness in Def. 41, the notion of a b -partner is technically a family of partners between two open nets.

Definition 44 [b -partner]

An open net C is a b -partner of an open net N if N and C are b -responsive.

In the following, we give an example and a counter-example for b -partners.

Example 45 We already showed in Ex. 42 that the open nets D and U are b -responsive; their mutual interaction, as detailed in Prop. 43, takes place

over the interface places q and d . Thus, U is a b -partner of D , and vice versa. \diamond

Example 46 The open net U' in Fig. 32a represents another user of the database server D . It has the same functionality as the open net U in Fig. 31b, but may additionally decide to quit and shut down the database (output place s). The open nets D and U' are composable; their composition $D \oplus U'$ is a closed net, which is depicted in Fig. 32b.

U' is not a b -partner of D : After sending a message s , open net D could fire *shutdown* and *forward* which leads to the nonfinal and noncommunicating marking $[f]$ of $D \oplus U'$. We generalize this observation to all open nets that are composable with D : No b -partner of D sends s , as otherwise D can successively fire the transitions *shutdown* and *forward*. After firing *forward*, open net D neither receives any input on s or q nor provides any output on d or f besides the single token on f produced by the firing of *forward*. Thus, after receiving s , open net D cannot participate in mutual interaction as stated in Prop. 43, and, therefore, no b -partner of D can send s . \diamond

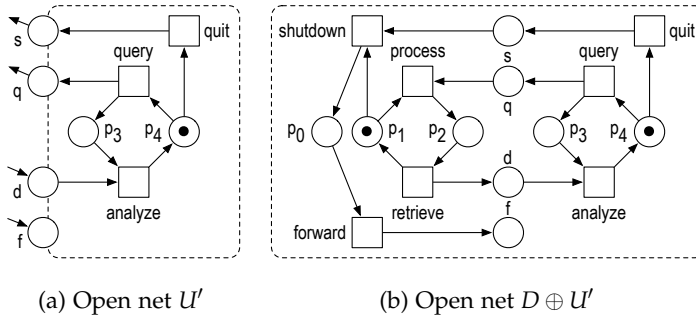


Figure 32: The open net U' modeling another user of the open net D in Fig. 31a, and their composition $D \oplus U'$. In addition to the figures, we have $\Omega_{U'} = \{[\]\}$, and $\Omega_{D \oplus U'} = \{[p_0]\}$.

While every open net has at least one partner, as explained in Sect. 3.1, there exist open nets that do not have any b -partner. An example is an open net N with a transition t such that $\bullet t = \{p\}$ and $t \bullet = \{p, o\}$ for an initially marked internal place p and an output place o . Repeatedly firing t violates any bound, which no open net that is composable with N could prevent.

We redefine the notion of conformance from Def. 36 for b -responsiveness to b -conformance.

Definition 47 [b -conformance]

For two interface-equivalent open nets $Impl$ and $Spec$, $Impl$ b -conforms to $Spec$, denoted by $Impl \sqsubseteq_{b, conf} Spec$, if for all open nets C the following holds: If C is a b -partner of $Spec$, then C is also a b -partner of $Impl$.

Later we will see that also b -conformance is not a precongruence, so we already introduce its coarsest precongruence. We refer to the coarsest precongruence that is contained in b -conformance as *compositional b -conformance*.

Definition 48 [compositional b -conformance]

We denote by $\sqsubseteq_{b, conf}^c$ the largest subset of $\sqsubseteq_{b, conf}$ such that $\sqsubseteq_{b, conf}^c$ is a precongruence with respect to \oplus . For two interface-equivalent open nets $Impl$ and $Spec$, $Impl$ compositionally b -conforms to $Spec$, if $Impl \sqsubseteq_{b, conf}^c Spec$.

We give an example for b -conformance.

Example 49 Figure 33a depicts a patched database server D' . It has the same functionality as D in Fig. 31a but never forwards a shutdown message to the output place f and, hence, terminates after receiving a shutdown message (final marking []).

The open net U in Fig. 31b is a b -partner of D' just as U is a b -partner of D , as they mutually communicate over the places q and d (recall that U does not send s). In contrast, the open net U' in Fig. 32a is not a b -partner of D (see Ex. 45), but it is a b -partner of D' : The only reachable marking without further communication—that is, marking [] of $D' \oplus U'$ —is a final marking of $D' \oplus U'$. We depict $D' \oplus U'$ in Fig. 33b.

The open net D' b -conforms to the open net D : No b -partner of D can send s , as already explained in Ex. 45. In contrast, D does not b -conform to D' : Receiving s is catastrophic for D but not necessarily for D' , because D' may reach its final marking []. For example, the open net U' is a b -partner of D' but no b -partner of D . Again, these statements hold for any bound b . \diamond

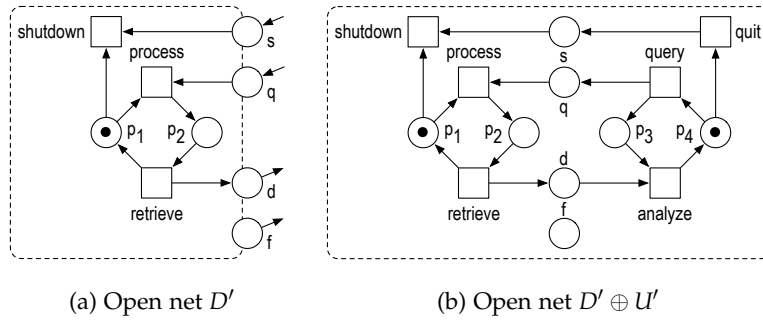


Figure 33: Two open nets modeling a patched database server, and its composition with the open net U' in Fig. 32a. In addition to the figures, we have $\Omega_{D'} = \Omega_{D' \oplus U'} = \{[]\}$.

Like the conformance relation in Sect. 3.1, we can show that b -conformance is not compositional. In other words, the preorder $\sqsubseteq_{b, conf}$ is a strict superset of the precongruence $\sqsubseteq_{b, conf}^c$.

Example 50 We extend the example in Ex. 49 with the open nets X and Y in Fig. 34. The open net X is a b -partner of $D \oplus Y$ but no b -partner of $D' \oplus Y$: Whereas the transition $activate$ of Y can be fired in $(D \oplus Y) \oplus X$ (enabling b -responsiveness), it cannot be fired in $(D' \oplus Y) \oplus X$. Thus, although D' b -conforms to D by Ex. 49, $D' \oplus Y$ does not b -conform to $D \oplus Y$ because of the open net X . Therefore, D' does not compositionally b -conform to D . \diamond

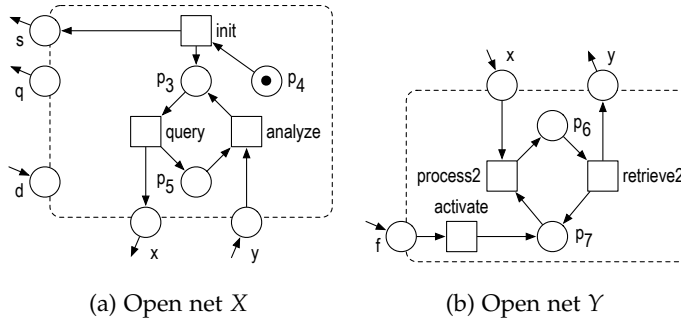


Figure 34: Two open nets proving that b -conformance is not a precongruence with regard to open net composition \oplus . In addition to the figures, we have $\Omega_X = \{[p_3]\}$ and $\Omega_Y = \{[p_7]\}$.

3.3 CLASSIFYING BOTH FORMALIZATIONS

In the previous sections, we formalized responsiveness and b -responsiveness for open nets as well as the arising conformance and the b -conformance relation. In this section, we compare responsiveness and b -responsiveness to existing behavioral correctness criteria for open systems. In addition, we compare conformance and b -conformance. We postpone a classification of conformance and b -conformance into a spectrum of preorders between systems, as this depends on characterizations of both relations which we elaborate in Part II.

3.3.1 Classifying responsiveness and b -responsiveness

Figure 35 depicts the relations between four behavioral correctness criteria for open nets that are closely related to responsiveness: b -bounded weak termination [181, 162, 44, 259] (b -WT), weak termination [167, 259] (WT), b -bounded deadlock freedom [166, 258] (b -DF), and deadlock freedom [166, 258] (DF). An arrow (and a sequence of arrows) between two criteria denotes the implication relation; for example, weak termination implies (is “stricter” than) deadlock freedom. Deadlock freedom requires the composition of two open nets to be deadlock-free. Recall from Sect. 2.3 that our notion of deadlock freedom is non-standard [216] as we distinguish between final and nonfinal markings [166]. Weak termination requires the composition of two open nets to be weakly terminating, i.e., a final marking is always reachable. We can combine both weak termination and deadlock freedom with b -boundedness of the composition, yielding b -bounded weak termination and b -bounded deadlock freedom, respectively.

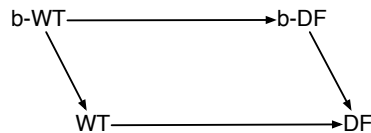


Figure 35: Some behavioral correctness criteria for open nets.

Two open nets are responsive if they have the perpetual possibility to either terminate or communicate; they are b -responsive if their composition is additionally b -bounded. Consequently, b -responsiveness is a “stricter”

correctness criterion than responsiveness: Two b -responsive open nets are responsive as well, yet the converse does not hold in general.

On the one hand, responsiveness is a stricter notion than deadlock freedom; that is, two responsive open nets are always deadlock-free. The converse does not hold in general.

Example 51 Consider the open nets N_1 and N_2 in Fig. 36. The net $N_1 \oplus N_2$ is deadlock-free because of transition t_2 . However, the only reachable marking $[p_0, p_2]$ of $N_1 \oplus N_2$ is neither final nor enables a transition that is connected to an interface place of N_1 or N_2 . Thus, N_1 and N_2 are deadlock-free but not responsive. \diamond

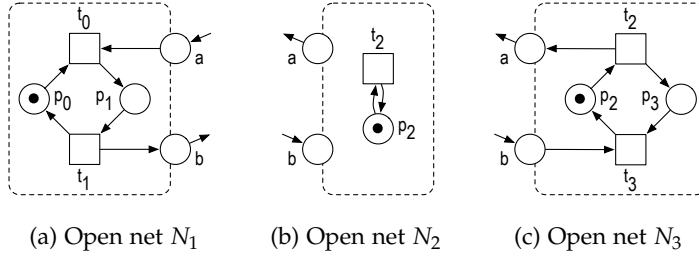


Figure 36: Three open nets that classify responsiveness between weak termination and deadlock freedom. In addition to the figures, we have $\Omega_{N_1} = \{[p_0]\}$, $\Omega_{N_2} = \{[]\}$, and $\Omega_{N_3} = \{[p_3]\}$.

On the other hand, responsiveness is a weaker notion than weak termination, meaning two weakly terminating open nets are always responsive. Again, the converse does not hold in general:

Example 52 The open nets N_1 and N_3 in Fig. 36 are responsive, perpetually communicating over the interface places a and b . However, the only final marking $[p_0, p_3]$ of $N_1 \oplus N_3$ is not reachable in $N_1 \oplus N_3$. Thus, N_1 and N_3 are responsive but not weakly terminating. \diamond

As the open nets $N_1 \oplus N_2$ and $N_1 \oplus N_3$ are 1-bounded, we conclude that b -responsiveness is also stricter than b -bounded deadlock freedom and b -bounded weak termination is stricter than b -responsiveness. We depict in Fig. 37 the classification of responsiveness and b -responsiveness into the spectrum of behavioral correctness criteria for open nets from Fig. 35.

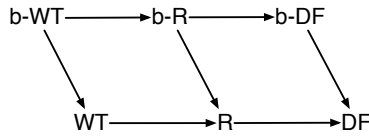


Figure 37: Responsiveness (R) and b -responsiveness (b-R) in a spectrum of behavioral correctness criteria for open nets.

3.3.2 Comparing conformance and b -conformance

Although the notions of responsiveness and b -responsiveness differ only by b -boundedness of the composition of two open nets, the resulting notions of conformance and b -conformance are incomparable. We illustrate this with

two technical examples. With the first example, we show that conformance does not imply b -conformance.

Example 53 Consider the two open nets N_4 and N_5 in Fig. 38. The open net N_4 conforms to the open net N_5 , because every partner of N_5 mutually communicates over the places a and b , which is also possible in the composition with N_4 . However, N_4 does not b -conform to N_5 : The place p_0 is unbounded in every composition of N_4 with a b -partner C of N_5 and, thus, C is not a b -partner of N_4 . More precisely, N_4 has, in contrast to N_5 , no b -partner at all. \diamond

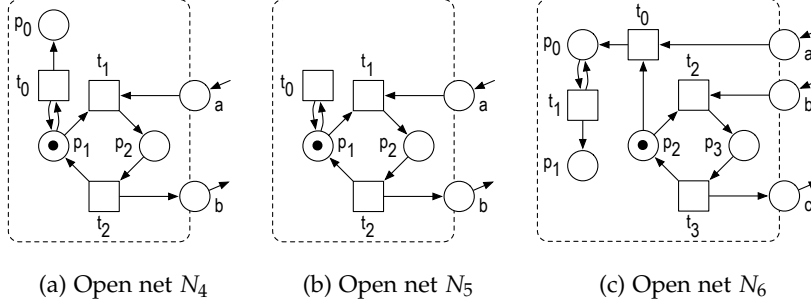


Figure 38: Three open nets proving that conformance and b -conformance are incomparable. In addition to the figures, we have $\Omega_{N_4} = \Omega_{N_5} = \Omega_{N_6} = \{\{\}\}$.

With the second example, we show that b -conformance does not imply conformance.

Example 54 We define the open net N_7 as the open net N_6 in Fig. 38c but with $\Omega_{N_7} = \{m \in \text{Bags}(P_{N_7}) \mid \forall p \in P_{N_7} \setminus \{p_0, p_1\} : m(p) = 0\}$ as its set of final markings. N_6 b -conforms to N_7 because no b -partner C of N_7 can send a message a ; otherwise, transition t_0 may fire and the place p_1 becomes unbounded in $N_7 \oplus C$. Thus, every b -partner C of N_7 perpetually communicates with N_7 over the places b and c , and therefore C is also a b -partner of N_6 .

However, N_6 does not conform to N_7 , because a partner C of N_7 may send a message a , which leads to a final marking in $N_7 \oplus C$ but not in $N_6 \oplus C$. Thus, C may not be a partner of N_6 , which contradicts the definition of conformance. \diamond

3.4 CONCLUSIONS AND RELATED WORK

In this chapter, we formalized responsiveness, b -responsiveness and the corresponding (compositional) conformance relations, (compositional) conformance and (compositional) b -conformance. As we require a user to define the bound b , we obtain a family of preorders and precongruences for b -responsiveness, each parameterized by b . We compared responsiveness and b -responsiveness to two other behavioral correctness criteria for open nets, and showed that responsiveness is stricter than deadlock freedom and weaker than weak termination. In addition, we showed that conformance and b -conformance are incomparable. In the following, we review related work on responsiveness.

The idea of responsiveness for finite state open systems with final states has been coined by Wolf [258]: An open net N is responsive if $\text{inner}(N)$ is

b -bounded and from every reachable marking we can reach either a final marking or a marking that enables a transition with an output place in its postset [258]. In other words, Wolf defines responsiveness for single open nets and considers only such responsive nets; this guarantees stricter forms of our responsiveness and b -responsiveness notions. More generally, we also deal with open nets that are responsive in some open net compositions but not in others. Our definition of b -responsiveness is also more general in terms of boundedness: In [258], only internally b -bounded open nets are considered, whereas we consider arbitrary open nets (i.e., even internally unbounded), as long as their composition with a partner is b -bounded.

Müller [187] presents an asymmetrical definition of responsiveness from the point of view of one individual open system in a composition. In contrast, our notions of responsiveness and b -responsiveness are symmetrical.

Our definition of responsiveness corresponds to the notion of final-responsiveness in [249] and generalizes the notion of responsiveness in [247]: While responsiveness in [247] requires at least one net of the composition to repeatedly talk to the other net, our responsiveness in Def. 31 also allows the composition to terminate instead (i.e., to reach a common final marking).

Recently, responsiveness has gained interest because it is crucial in the setting of interacting open systems. An example is Microsoft's asynchronous event driven programming language P, which is used to implement device drivers [76]. Desai et al. [76] define responsiveness for bounded message channels, which is similar to our notion of b -responsiveness. However, their notion of responsiveness additionally requires that no message in any channel is ignored forever. Therefore, b -responsiveness in Def. 41 is a more general notion than responsiveness in [76].

In other work, the term responsiveness refers to different properties: Reed et al. [214] aim to exclude certain deadlocks, whereas responsiveness in our setting refers to the ability to communicate. The works [137, 14, 100] consider with the π -calculus a more expressive model than open nets but in the setting of synchronous communication, whereas we consider asynchronous communication. Moreover, responsiveness in [14, 100] and lock-freedom in [137] guarantee that communication over a certain channel is eventually possible. In contrast, our notion of responsiveness requires that communication over some channel is always possible.

Kobayashi [137] defines responsiveness over the infinite runs of the system, thereby using a strong fairness for the channel synchronization. Moreover, the language considered in [137] does not support choices. Acciai and Boreale [14] use a type system and reduction rules different from Kobayashi, and they give an example of a responsive process that cannot be expressed in [137].

Gamboni and Ravara [100] use a variant of the π -calculus that is more expressive than the one in [137]: Choices are part of the language and it is also possible to express how many times a communication over a certain channel should take place. In addition to responsiveness in [14] (called activeness in [100]), Gamboni and Ravara require that whenever communication takes place over a channel, then the respective processes conform to a specified protocol. The latter property is called responsiveness in [100].

Recently, Padovani [200] has taken up lock-freedom from Kobayashi [137]. He defines a behavioral type system using asynchronously communicating session types and considers the progress property. However, progress is a stricter notion than b -responsiveness as it (like responsiveness in [76]) additionally requires that no message in any channel is ignored forever.

Part II

THE MODEL-MODEL SCENARIO

CONFORMANCE AND COMPOSITIONAL CONFORMANCE

This chapter is based on results published in [193, 247, 249].

IN the model-model scenario, we assume that both the specification and the implementation of an open system are given as formal models. Then, we want to verify responsiveness by using conformance checking between the two formal models. We already presented with open nets a suitable formalism for open systems in Chap. 2. In Chap. 3, we formalized two variants of responsiveness for open nets and the arising (compositional) conformance relations: Responsiveness and (compositional) conformance, and b -responsiveness and (compositional) b -conformance. Thereby, we formalized the corresponding conformance relations as partner-inclusion: The implementation $Impl$ conforms to the specification $Spec$ if the set of partners of $Impl$ includes the set of partners of $Spec$; $Impl$ compositionally conforms to $Spec$ if they conform and their conformance relation is preserved under the open nets composition operator. Figure 39 illustrates again our formalization of conformance.

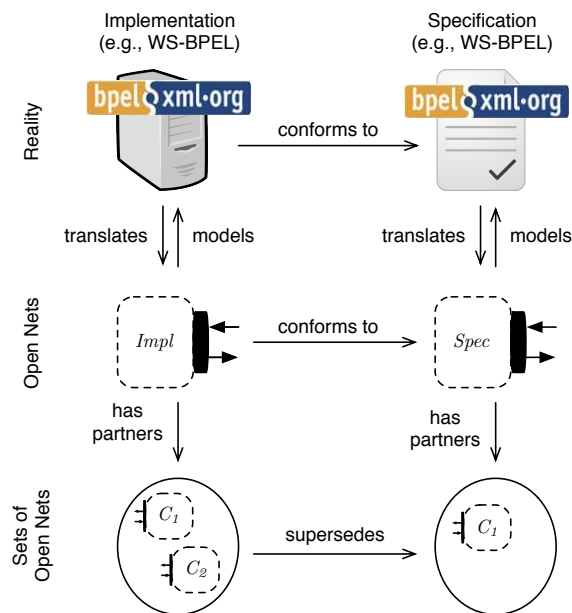


Figure 39: Formalizing conformance as partner inclusion.

In the chapters of Part II, we analyze conformance and b -conformance for compositionality and decidability. It turns out that both conformance and b -conformance are not compositional. Therefore, we additionally characterize compositional conformance and compositional b -conformance. Orthogonally, we investigate whether conformance and b -conformance as well as compositional conformance and compositional b -conformance are decidable. It turns out that b -conformance and compositional b -conformance are decidable whereas conformance and compositional conformance are not. Consequently, we elaborate decision procedures only for b -conformance and com-

positional b -conformance. Because conformance and compositional conformance turn out to be undecidable and both undecidability proofs are similar in their structure, we merged our results about conformance and compositional conformance into one chapter—that is, this chapter. In Chap. 5, we investigate b -conformance and in Chap. 6, we investigate compositional b -conformance. Table 1 illustrates how we structure the above mentioned results into the chapters of Part II. We left out Chap. 7, in which we summarize the results of Part II and review related work.

relation	characterization	compositionality	decidability
conformance	Chap. 4	Chap. 4	Chap. 4
b -conformance	Chap. 5	Chap. 6	Chap. 5 & Chap. 6

Table 1: The structure of Part II without Chap. 7. We highlight the current chapter with a gray background.

In this chapter, we give a fine-grained analysis of the conformance and compositional conformance notions that we introduced in Sect. 3.1. Figure 40 illustrates how we achieve this. To this end, we provide for each open net a certain denotational semantics: A trace-based semantics for conformance, to which we refer as *stopdead*-semantics, and a failure-based semantics for compositional conformance, to which we refer as \mathcal{F}_{fin}^+ -semantics. Then, we show that a refinement relation based on these semantics coincides with the respective conformance relation. In other words, we provide a trace-based characterization of conformance and a failure-based characterization of compositional conformance. Based upon these characterizations, we show that conformance and compositional conformance are undecidable by reducing them to the halting problem of counter machines.

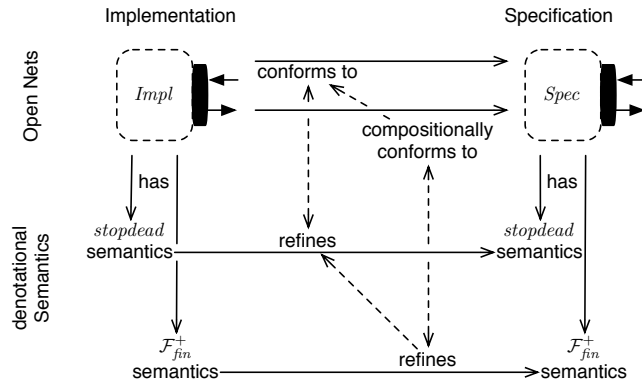


Figure 40: Characterizing conformance and compositional conformance using denotational semantics for open nets. A solid arc illustrates the relation described by the corresponding arc label. Dashed arcs illustrate logical implication or logical equivalence, depending on their number of heads.

We characterize conformance in Sect. 4.1 and compositional conformance in Sect. 4.2. We prove conformance and compositional conformance to be undecidable in Sect. 4.3 and conclude this chapter with a discussion in Sect. 4.4.

4.1 CHARACTERIZING CONFORMANCE

In this section, we aim to characterize the conformance relation between two interface-equivalent open nets $Impl$ and $Spec$. To this end, we provide a trace-based semantics for each open net in Sect. 4.1.1. The semantics consists of two sets of traces. Inclusion of the two sets of traces of $Impl$ in the two sets of traces of $Spec$ defines a refinement relation that coincides with conformance, which we show in Sect. 4.1.2. This way, we provide a trace-based characterization of conformance. Figure 41 illustrates the content of this section.

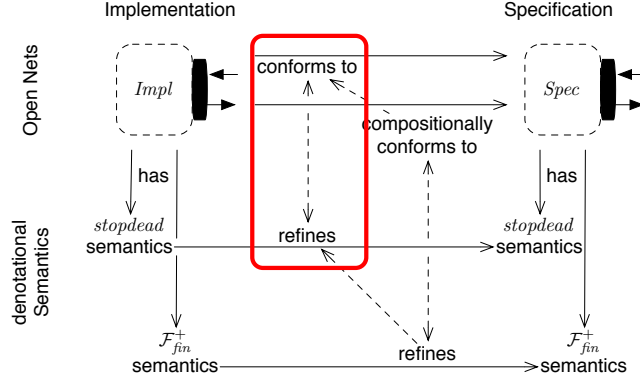


Figure 41: The content of Sect. 4.1.

4.1.1 The stopdead-semantics for open nets

The idea for a trace-based semantics characterizing conformance is to collect complete traces of (the environment of) an open net, thereby distinguishing between successful and unsuccessful completed traces. Our trace-based semantics of an open net N is based on the so-called *stop-traces* and *dead-traces* of N 's environment $env(N)$. A *stop-trace* records a run of $env(N)$ that ends in a marking weakly enabling actions of I only, such that N stops unless some input from another open net is provided. A *dead-trace* is a *stop-trace* leading to nonfinal markings only. Our notion of a *stop-trace* and a *dead-trace* is a weak version of two notions with the same name in [228], where only transitions of I and no τ -transitions are allowed to be enabled. We refer to the semantics consisting of the two sets of *stop-traces* and *dead-traces* as *stopdead-semantics*.

Definition 55 [stopdead-semantics]

Let N be a labeled net. A marking m of N is a *stop except for inputs* if there is no $o \in \Sigma^{out}$ such that $m \xrightarrow{o}$. If additionally, there is no final marking m' of N with $m \xrightarrow{\epsilon} m'$, then m is *dead except for inputs*. The *stopdead-semantics* of N is defined by the two sets of traces

- $stop(N) = \{w \mid m_N \xrightarrow{w} m \text{ and } m \text{ is a stop except for inputs}\}$, and
- $dead(N) = \{w \mid m_N \xrightarrow{w} m \text{ and } m \text{ is dead except for inputs}\}$.

Example 56 As a running example for this chapter, consider again the unreliable time server S and its client C from Sect. 3.1. For convenience, we depict them again, together with the second client C' from Ex. 34, in Fig. 42. The language of S is

$$L(S) = \{w \in \{r, t\}^* \mid \forall v \sqsubseteq w : |v|_t \leq |v|_r + 1\} \\ \cup \{wez \mid w, z \in \{r, t\}^* \wedge \forall v \sqsubseteq w : |v|_t \leq |v|_r + 1 \wedge |wz|_t \leq |w|_r\}.$$

Observe that after firing e , transition r is continuously enabled in $env(S)$ while transition t may also fire because of pending tokens on the place t^0 . Every *stop*-trace of S either contains an e or the number of r 's is smaller than the number of t 's; more precisely,

$$stop(S) = \{w \in \{r, t\}^* \mid |w|_t = |w|_r + 1 \wedge \forall v \sqsubseteq w : |v|_t \leq |v|_r + 1\} \\ \cup \{wez \mid w, z \in \{r, t\}^* \wedge wez \in L(S)\}.$$

As $[]$ is the only final marking of S , we have

$$dead(S) = \{w \in \{r, t\}^* \mid |w|_t = |w|_r + 1 \wedge \forall v \sqsubseteq w : |v|_t \leq |v|_r + 1\} \\ \cup \{wez \mid w, z \in \{r, t\}^* \wedge wez \in L(S) \wedge |wz|_t < |wz|_r\}.$$

The trace e illustrates the difference between *stop*- and *dead*-traces of S : It always leads to the marking $[]$ of $env(S)$, thus $e \in stop(S)$ but $e \notin dead(S)$.

For C , the marking $[p_3]$ is the only reachable stop except for inputs of $env(C)$. Thus, every *stop*-trace of C has an equal number of t 's and r 's and does not contain an e , or at least as many t 's as r 's because of transition *recover* and at least as many r 's as t 's because of transition *response*; more precisely,

$$stop(C) = \{w \in \{r, t\}^* \mid |w|_t = |w|_r \wedge \forall v \sqsubseteq w : |v|_t \geq |v|_r\} \\ \cup \{wezr \mid w \in \{r, t\}^* \wedge z \in \{e, r, t\}^* \wedge \forall v \sqsubseteq w : |v|_t \geq |v|_r \\ \wedge |ez|_e \leq |wez|_t - |w|_r \wedge |wez|_t \leq |wez|_r\}.$$

As $[p_3]$ is the only final marking of C and all *stop*-traces of C lead to $[p_3]$, we have $dead(C) = \emptyset$.

The open net C' is a modification of C —that is, transitions *reset* and *recover* removed and $[p_4]$ instead of $[p_3]$ as the only final marking. Therefore, we have

$$stop(C') = \{w \in \{r, t\}^* \mid |w|_t = |w|_r \wedge \forall v \sqsubseteq w : |v|_t \geq |v|_r\} \\ \cup \{wez \mid w \in \{r, t\}^* \wedge z \in \{r, t, e\}^* \wedge \forall v \sqsubseteq wz : |v|_t \geq |v|_r\} \\ dead(C') = \{w \in \{r, t\}^* \mid |w|_t = |w|_r \wedge \forall v \sqsubseteq w : |v|_t \geq |v|_r\} \\ \cup \{wez \mid w \in \{r, t\}^* \wedge z \in \{r, t, e\}^* \wedge \forall v \sqsubseteq wz : \\ |v|_t \geq |v|_r \wedge (|wz|_t > |wz|_r \vee |z|_e \geq 1)\}.$$

The fact that every *stop*-trace of C that does not contain an e is also a *dead*-trace of C' derives from the changed final marking of C' . The trace e illustrates the difference between the open nets C and C' : We have $e \notin stop(C)$ because e always leads to a marking m such that place p_4 is marked, and, thus, m is not a stop except for inputs of C because of transition *reset*. In contrast to C , we have $e \in stop(C')$ because transition *reset* was removed from C' . \diamond

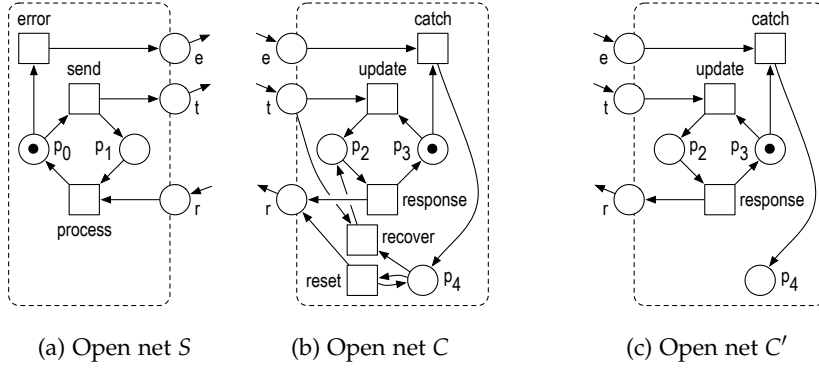


Figure 42: The open nets S , C , and C' from Sect. 3.1. In addition to the figures, we have $\Omega_S = \{[]\}$, $\Omega_C = \{[p_3]\}$, and $\Omega_{C'} = \{[p_4]\}$.

The presence of *stop*- and *dead*-traces in open nets N_1 and N_2 is closely related to the question whether N_1 and N_2 are responsive. We continue by relating responsiveness to markings that are dead except for inputs.

Lemma 57 [responsiveness vs. dead except for inputs]

Let N_1 and N_2 be composable open nets such that $N_1 \oplus N_2$ is a closed net. Let $E = env(N_1) \parallel env(N_2)$, and let m be a marking of $N_1 \oplus N_2$ and \bar{m} be a marking of E such that m and \bar{m} agree. Then the following hold:

1. If m is responsive, then \bar{m} is not dead except for inputs.
2. If m and \bar{m} strongly agree, the converse of (1) also holds.

Proof. Let $C = N_1 \oplus N_2$ and $O = (I_1 \cap O_2) \uplus (I_2 \cap O_1)$. We have $C = env(N_1 \oplus N_2)$ and $O = O_1 \uplus O_2$, because $N_1 \oplus N_2$ is a closed net. Note that only transitions in O are not τ -labeled in E .

(1) If m and \bar{m} do not agree strongly, \bar{m} is not even a stop except for inputs as there exists an $o \in O$ with $\bar{m} \xrightarrow{o}$ by Def. 25, hence $\bar{m} \xrightarrow{o}$. So assume that m and \bar{m} agree strongly. We distinguish whether we can reach a final marking from m or not: If there is a final marking m' of C reachable from m , then, according to Prop. 28(1), there is a marking \bar{m}' reachable from \bar{m} in E such that m' and \bar{m}' agree (even strongly). Marking \bar{m}' is a final marking of E by Prop. 28(3). Thus, \bar{m} is not dead except for inputs by Def. 55. If there is no final marking reachable from m , then, as m is responsive, we can fire some vt in C such that t is the first transition that produces a token on some $x \in O$, i.e., $m \xrightarrow{vt} m'$ in C . Then it is possible to insert transitions from O of E into v such that for the resulting $v't$: $\bar{m} \xrightarrow{v't} \bar{m}'$ in E and also m' and \bar{m}' strongly agree by Prop. 28(1). Hence either $\bar{m} \xrightarrow{y}$ for one of the inserted transitions y or $\bar{m} \xrightarrow{x}$, and \bar{m} is not a stop except for inputs (and thus not dead except for inputs).

(2) We distinguish whether \bar{m} is a stop except for inputs or not: If \bar{m} is no stop except for inputs, and does not enable any transition $x \in O$ (by strong agreement), we have $\bar{m} \xrightarrow{v} \bar{m}' \xrightarrow{t} \bar{m}''$ in E where neither t nor any transition in v is in O , and \bar{m}'' enables a transition $x \in O$ disabled at \bar{m}' . Hence, $x^o \in t^\bullet$ in E and, consequently, $x \in t^\bullet$ in C by Def. 17. Applying Prop. 28(2), we get $m \xrightarrow{v} m'$ in C such that m' and \bar{m}' agree. Thus, transition t is enabled at m' in C , and m is responsive. If \bar{m} is a stop except for inputs, then there is

a final marking \bar{m}' of E reachable from \bar{m} . Applying Prop. 28(2), there is a marking m' reachable from m in C such that m' and \bar{m}' agree. Marking m' is a final marking of C by Prop. 28(3), proving responsiveness of m . \square

Next, we relate a marking that is dead except for inputs in the parallel composition of two environments to a marking that is dead except for inputs in one of the involved environments.

Lemma 58 [dead except for inputs vs. *stopdead*-semantics]

Let N_1 and N_2 be composable open nets, and let $E = env(N_1) \parallel env(N_2)$. Let m_1 and m_2 be a marking of $env(N_1)$ and $env(N_2)$, respectively. Then, $m = m_1 + m_2$ is dead except for inputs in E iff m_1 is a stop except for inputs and m_2 is dead except for inputs, or vice versa.

Proof. \Rightarrow : W.l.o.g., assume that m_1 is not a stop except for inputs due to $m_1 \xrightarrow{o}$ with $o \in O_1$. As m_2 enables $o \in I_2$, we get $m \xrightarrow{o}$ with $o \in O_1 \uplus O_2$ by Prop. 23, hence m is no stop except for inputs, a contradiction. Thus, both m_1 and m_2 are stops except for inputs. Assume neither m_1 nor m_2 are dead except for inputs due to m'_1 and m'_2 , respectively. Then $m = m_1 + m_2 \xrightarrow{\varepsilon} m'_1 + m'_2$ by Prop. 23 and $m'_1 + m'_2$ is a final marking by Prop. 28(3). This contradicts the assumption.

\Leftarrow : Because m_1 and m_2 are stops except for inputs, there is no $o \in O_1 \uplus O_2$ such that $m_1 \xrightarrow{o}$ in $env(N_1)$ and $m_2 \xrightarrow{o}$ in $env(N_2)$. Applying Prop. 23, $m_1 + m_2 \xrightarrow{o}$ is not in E ; thus, m is a stop except for inputs. W.l.o.g., assume m_2 is dead except for inputs. Whenever $m \xrightarrow{\varepsilon} m'$, Prop. 23 gives us $m_1 \xrightarrow{\varepsilon} m'_1$ and $m_2 \xrightarrow{\varepsilon} m'_2$ where neither m'_2 nor—by Prop. 28(3)— $m' = m'_1 + m'_2$ are final. Thus, m is dead except for inputs in E by Def. 55. \square

We combine Lem. 57 and Lem. 58 and show how the *stopdead*-semantics can be used to characterize responsiveness.

Proposition 59 [responsiveness vs. *stopdead*-semantics]

For two composable open nets N_1 and N_2 such that $N_1 \oplus N_2$ is a closed net, we have

$$N_1 \text{ and } N_2 \text{ are responsive} \quad \text{iff} \quad \begin{aligned} stop(N_1) \cap dead(N_2) &= \emptyset \text{ and} \\ dead(N_1) \cap stop(N_2) &= \emptyset. \end{aligned}$$

Proof. Let $C = N_1 \oplus N_2$ and $E = env(N_1) \parallel env(N_2)$.

\Rightarrow : Proof by contraposition. W.l.o.g., we assume a trace $w \in stop(N_1) \cap dead(N_2)$. Hence, $m_{env(N_1)} \xrightarrow{w} m_1$ in $env(N_1)$ and $m_{env(N_2)} \xrightarrow{w} m_2$ in $env(N_2)$ such that m_1 is a stop except for inputs and m_2 is dead except for inputs. By Lem. 58, $m_1 + m_2$ is dead except for inputs in E . By Prop. 28(2), a marking m is reachable in C such that m and $m_1 + m_2$ agree, and m is not responsive by Lem. 57.

\Leftarrow : Proof by contraposition. Assume $m_C \xrightarrow{\varepsilon} m$ in C such that m is not responsive. Applying Prop. 28(1), we can reach some $m_1 + m_2$ in E (with m_i a marking of N_i , $i = 1, 2$) such that m and $m_1 + m_2$ strongly agree, and, by Lem. 57, $m_1 + m_2$ is dead except for inputs. By Prop. 23, we have $m_{env(N_1)} \xrightarrow{w} m_1$ in $env(N_1)$ and $m_{env(N_2)} \xrightarrow{w} m_2$ in $env(N_2)$, and by Lem. 58, m_1 is a stop except for inputs and m_2 is dead except for inputs, or vice versa. Thus, $w \in stop(N_1) \cap dead(N_2)$ or $w \in dead(N_1) \cap stop(N_2)$. \square

Example 60 For the open nets S and C in Fig. 42, their *stop*- and *dead*-traces are given in Ex. 56. We can see that $\text{stop}(S) \cap \text{dead}(C) = \emptyset$ because $\text{dead}(C) = \emptyset$. In addition, we have $\text{dead}(S) \cap \text{stop}(C) = \emptyset$: Every *dead*-trace of S without an e has an unequal number of t 's and r 's, whereas every *stop*-trace of C without an e has an equal number of t 's and r 's. Every *dead*-trace of S with exactly one e has at most as many t 's as r 's preceding e , whereas every *stop*-trace of C with exactly one e must have more t 's than r 's preceding e . Thus, S and C are indeed responsive by Prop. 59, as already claimed in Ex. 32.

For S and the open net C' in Fig. 42c, we can see in Ex. 56 that $\text{stop}(S) \cap \text{dead}(C') = \emptyset$: Every *dead*-trace of C' either is a *stop*-trace of C (and thus not a *stop*-trace of S by the previous argumentation) or contains an e and the number of t 's is greater than the number of r 's. In contrast, every *stop*-trace of S that contains an e has at most as many t 's as r 's. In addition, we have $\text{dead}(S) \cap \text{stop}(C') = \emptyset$: Every *stop*-trace of C' either is a *stop*-trace of C (and thus not a *dead*-trace of S by the previous argumentation) or contains an e and has at least as many t 's as r 's. In contrast, every *dead*-trace of S that contains an e has fewer t 's than r 's. Thus, S and C' are responsive by Prop. 59, as already claimed in Ex. 35. \diamond

4.1.2 Refinement on the stopdead-semantics

In the previous section, we defined a trace-based semantics for open nets that consists of *stop*-traces (i.e., successfully completed traces) and *dead*-traces (i.e., unsuccessfully completed traces). Inclusion of the *stop*- and *dead*-traces of two open nets defines a refinement relation. With the next theorem, we prove that an open net $Impl$ conforms to an open net $Spec$ if and only if every *stop*-trace of $Impl$ is contained in the *stop*-traces of $Spec$ and every *dead*-trace of $Impl$ is contained in the *dead*-traces of $Spec$. In other words, we provide a trace-based characterization of conformance.

Theorem 61 [conformance and stopdead-inclusion coincide]

For two interface-equivalent open nets $Impl$ and $Spec$, we have

$$Impl \sqsubseteq_{\text{conf}} Spec \quad \text{iff} \quad \text{stop}(Impl) \subseteq \text{stop}(Spec) \text{ and} \\ \text{dead}(Impl) \subseteq \text{dead}(Spec).$$

Proof. \Leftarrow : Proof by contraposition. Consider an open net C such that $Impl \oplus C$ and, equivalently, $Spec \oplus C$ are closed nets. Assume that C is not a partner of $Impl$. Then $Impl$ and C are not responsive by Def. 33, and we find a trace $w \in \text{stop}(Impl) \cap \text{dead}(C)$ or $w \in \text{dead}(Impl) \cap \text{stop}(C)$ by Prop. 59. Due to $\text{stop}(Impl) \subseteq \text{stop}(Spec)$ and $\text{dead}(Impl) \subseteq \text{dead}(Spec)$, we have $w \in \text{stop}(Spec) \cap \text{dead}(C)$ or $w \in \text{dead}(Spec) \cap \text{stop}(C)$, respectively. Again with Prop. 59, $Spec$ and C are not responsive; that is, C is not a partner of $Spec$.

\Rightarrow : The idea is to construct for a *dead*-trace (*stop*-trace) w of $Impl$ an open net and to show using conformance of $Impl$ and $Spec$ that w is also a *dead*-trace (*stop*-trace) of $Spec$.

Let I be the input and O be the output places of $Impl$ and, equivalently, of $Spec$. If $I = O = \emptyset$, we have $\text{stop}(Impl) = \{\varepsilon\} = \text{stop}(Spec)$. Furthermore, either $\text{dead}(Impl) = \emptyset$ (and we are done) or $\text{dead}(Impl) = \{\varepsilon\}$ and we consider an open net C just consisting of a marked place, giving a final marking; C is not a controller of $Impl$, hence not of $Spec$, implying $\text{dead}(Spec) = \{\varepsilon\}$.

For the case $I \neq \emptyset \neq O$, we consider a trace $w \in \text{dead}(\text{Impl})$. Let $w = w_1 \dots w_n$ with $w_j \in I \uplus O$, for $j = 1, \dots, n$, and let $o \in I$ be arbitrary but fixed. Define the open net $N_w = (P, T, F, m_{N_w}, \emptyset, O, I)$ by

- $P = \{p_0, \dots, p_n\}$,
- $T = \{t_1, \dots, t_n\}$,
- $F = \begin{aligned} & \{(p_i, t_{i+1}) \mid 0 \leq i \leq n-1\} \\ & \uplus \{(t_i, p_i) \mid 1 \leq i \leq n\} \\ & \uplus \{(w_i, t_i) \mid 1 \leq i \leq n, w_i \in O\} \\ & \uplus \{(t_i, w_i) \mid 1 \leq i \leq n, w_i \in I\}, \text{ and} \end{aligned}$
- $m_{N_w} = [p_0]$.

We extend N_w to an open net $N_{w,o} = (P', T', F', m_{N_{w,o}}, \Omega, O, I)$ —see Fig. 43—with

- $P' = P \uplus \{p, p'\} \uplus \{p'_0, \dots, p'_{n-1}\}$,
- $T' = T \uplus \{t, t'_0, \dots, t'_{n-1}, t''_0, \dots, t''_{n-1}\} \uplus \{t_{w_i} \mid w_i \in O\}$,
- $F' = \begin{aligned} & F \\ & \uplus \{(p', t), (t, p'), (t, o)\} \\ & \uplus \{(p, t_{w_i}) \mid w_i \in O\} \\ & \uplus \{(t_{w_i}, p') \mid w_i \in O\} \\ & \uplus \{(w_i, t_{w_i}) \mid w_i \in O\} \\ & \uplus \{(p_i, t'_i) \mid 0 \leq i \leq n-1\} \\ & \uplus \{(t'_i, p'_i) \mid 0 \leq i \leq n-1\} \\ & \uplus \{(p'_i, t''_i) \mid 0 \leq i \leq n-1\} \\ & \uplus \{(t''_i, p'_i) \mid 0 \leq i \leq n-1\} \\ & \uplus \{(t''_i, o) \mid 0 \leq i \leq n-1\}, \end{aligned}$
- $m_{N_{w,o}} = [p_0, p]$, and
- $\Omega = \{[p_n, p]\}$.

At a stop except for inputs of $\text{env}(N_{w,o})$, no transition t with $o \in t^\bullet$ (in $N_{w,o}$) is enabled or can be enabled by firing τ -labeled transitions of $\text{env}(N_{w,o})$ by Def. 55. Hence, a marking of $\text{env}(N_{w,o})$ is a reachable stop except for inputs if and only if it is the marking $[p_n, p]$ (1), i.e., $\text{dead}(N_{w,o}) = \emptyset$ —keep in mind that every $a \in I$ is an output place of $N_{w,o}$.

Obviously, $\text{Impl} \oplus N_{w,o}$ as well as $\text{Spec} \oplus N_{w,o}$ are closed nets by construction of $N_{w,o}$. Because $w \in \text{stop}(N_{w,o})$ according to observation (1), Impl and $N_{w,o}$ are not responsive by Prop. 59 and choice of w . Hence, $N_{w,o}$ is not a partner of Impl and neither a partner of Spec , because Impl conforms to Spec . Thus, Spec and $N_{w,o}$ are not responsive because of Def. 33. Again with Prop. 59 and Def. 55, there exists $v \in (I \uplus O)^*$ with $m_{\text{env}(\text{Spec})} \xrightarrow{v} m_1$ and $m_{\text{env}(N_{w,o})} \xrightarrow{v} m_2$ such that both m_1 and m_2 are stops except for inputs, and additionally m_1 or m_2 is dead except for inputs. As $\text{dead}(N_{w,o}) = \emptyset$, m_1 is dead except for inputs of $\text{env}(\text{Spec})$; furthermore, $m_2 = [p_n, p]$.

According to observation (1), transitions t_1, \dots, t_n of $N_{w,o}$ occur in this order in a run σ of $\text{env}(N_{w,o})$ underlying v and, thus, there is no occurrence of a transition t'_j in σ by construction. Furthermore, no transition t_{w_i} has fired and removed the token from p . These facts imply that the Parikh vectors of σ and the run underlying w agree: Each t_i consumes a token from

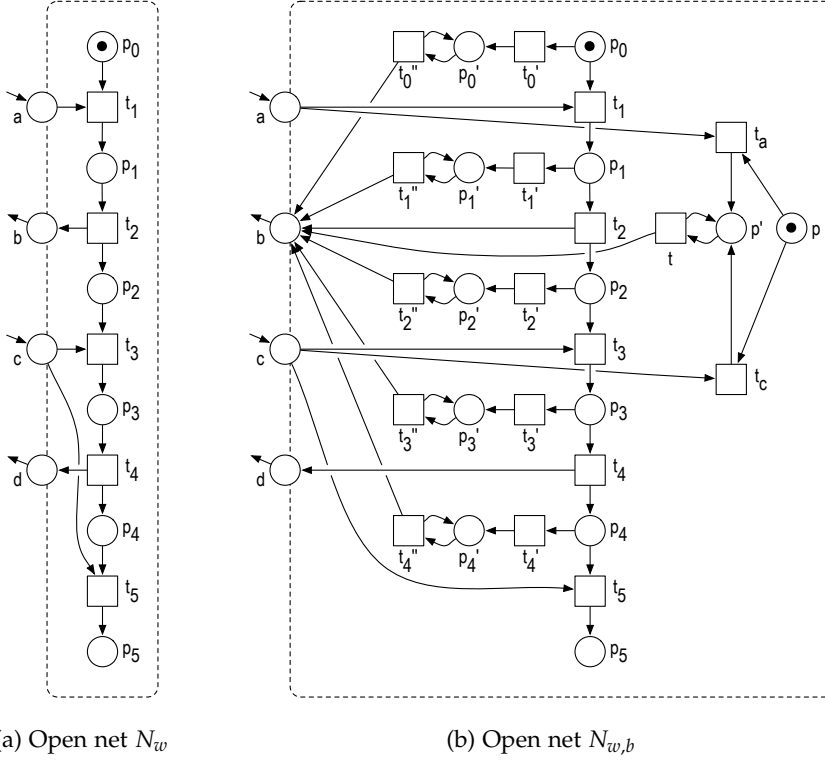


Figure 43: Construction of N_w and $N_{w,b}$ for $w = abcdc$ with $b, d \in I$ and $a, c \in O$; we have $\Omega_{N_{w,b}} = \{[p, p_5]\}$.

or produces a token on w_i , but all interface places are empty at the end of the traces.

In σ , each occurrence of t_j with $w_j \in t_j^\bullet$ (as output place of $N_{w,o}$, i.e., $w_j \in I$) is paired with a succeeding occurrence of w_j (as transition of $env(N_{w,o})$); otherwise, transition w_j would be enabled at m_2 in $env(N_{w,o})$ and m_2 would not be a stop except for inputs. As transition w_j is not in conflict with any other transition of $env(N_{w,o})$, we assume that w_j fires immediately after t_j . In the corresponding rearranged trace v' of v , all $w_j \in I$ occur in the same order as in w , and v' still leads to m_2 .

Similarly, each occurrence of t_j with $w_j \in \bullet t_j$ (as input place of $N_{w,o}$, i.e., $w_j \in O$) is paired with a preceding occurrence of w_j (as transition of $env(N_{w,o})$), which can be delayed such that it occurs immediately before t_j . In the corresponding rearranged trace v'' of v' , all $w_j \in O$ occur in the same order as in w , because the runs underlying v' and v'' have the same Parikh vector as the run underlying w ; thus, v'' is w .

We have transformed v into w by moving $w_j \in I$ backwards and $w_j \in O$ forwards. This can also be done in the run underlying v in $env(Spec)$, because the respective transitions have an empty preset and postset, respectively. Thus, $m_{env(Spec)} \xrightarrow{w} m_1$ and $m_{env(N_{w,o})} \xrightarrow{w} m_2$ and therefore $w \in dead(Spec)$.

For a trace $w \in stop(Impl)$, we fix some arbitrary $o \in I$ and define an open net $N'_{w,o} = (P', T', F', m_{N'_{w,o}}, \emptyset, O, I)$, which is identical to $N_{w,o}$ except for its empty set of final markings. Thus, $w \in dead(N'_{w,o})$, and we succeed with an argumentation similar to the previous one. \square

Example 62 Consider again the patched time server S' from Sect. 3.1, which we depict again in Fig. 44. For S' , we have

$$\text{stop}(S') = \{w \in \{r, t\}^* \mid |w|_t = |w|_r + 1 \wedge \forall v \sqsubseteq w : |v|_t \leq |v|_r + 1\}$$

and thus $\text{stop}(S') \subseteq \text{stop}(S)$ (see Ex. 56). In addition, each stop -trace of S' reaches the nonfinal marking $[p_1]$ in S' and in S , hence $\text{stop}(S') = \text{dead}(S')$ and $\text{dead}(S') \subseteq \text{dead}(S)$ (see Ex. 56). Therefore, we conclude with Thm. 61 that S' conforms to S , as already claimed in Ex. 38. \diamond

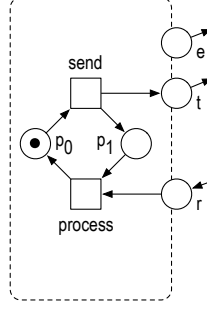


Figure 44: The patched time server S' from Sect. 3.1. We have $\Omega_{S'} = \{\{\}\}$.

Conformance, as defined in Def. 36, does not guarantee compositionality; that is, it is not a precongruence with respect to the open net composition operator \oplus . We showed this in Ex. 40 using the two open nets S and S' . To see the difference between S and S' , consider for example the trace rt . The trace rt reaches the marking $[p_1, r^i]$, $[p_0]$, $[p_1, t^o]$, or $[e^o]$ in $\text{env}(S)$, and the marking $[p_1, r^i]$, $[p_0]$, or $[p_1, t^o]$ in $\text{env}(S')$. In any of the four markings of $\text{env}(S)$, the trace re is always possible in $\text{env}(S)$, but re is not possible in any of the markings in $\text{env}(S')$. We can generalize this observation to any trace of $\text{env}(S)$ that does not contain an e . In the marking reached, there is a token on p_0 , p_1 , or e^o , and the trace re is always possible. In contrast, $\text{env}(S')$ can always refuse re , because $\text{env}(S')$ can never perform e at all. Therefore, it is not possible to differentiate between S and S' with something even weaker than standard failure semantics, as introduced by Brookes et al. [50]. Our trace-based semantics in Def. 55 is weaker than failures, but we must differentiate between S and S' to characterize compositional conformance. Therefore, we introduce a failure-based semantics for open nets that guarantees compositionality in the following section. Building upon this failure-based semantics, we characterize compositional conformance.

4.2 CHARACTERIZING COMPOSITIONAL CONFORMANCE

In this section, we introduce a failure-based semantics for open nets. We show that our semantics is compositional (i.e., the semantics of a composition of two open nets can be derived from the semantics of the composed open nets) in Sect. 4.2.1, and that it coincides with compositional conformance in Sect. 4.2.2. Thus, we provide a failure-based characterization of compositional conformance. Figure 45 illustrates the content of this section.

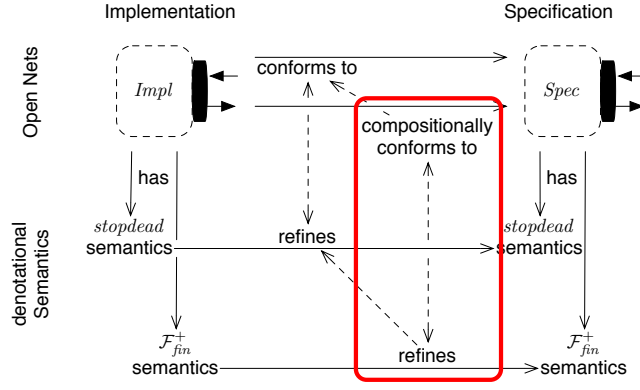


Figure 45: The content of Sect. 4.2.

4.2.1 The \mathcal{F}_{fin}^+ -semantics for open nets

Taking into account the counterexample showing that conformance is not compositional in Ex. 40 and the observation that refusal information is necessary to distinguish open nets in terms of conformance (see the last paragraph of Sect. 4.1.2), we shall characterize compositional conformance in terms of a variant of failure semantics. For this, we will not use CSP failures, as introduced by Brookes et al. [50], but Vogler’s \mathcal{F}^+ -semantics [246], which was also introduced by Voorhoeve and Mauw [250] as impossible futures semantics. Whereas a *failure* in [50] is a pair (w, X) where w is a trace of a labeled net and X is a subset of the alphabet—a *refusal set*—the \mathcal{F}^+ -semantics is a stronger notion, considering pairs (w, X) where X is a set of traces x such that wx is not a trace of the net; such a pair is a *tree failure*.

The \mathcal{F}^+ -semantics does not distinguish between final and nonfinal markings, whereas the notion of responsiveness does. In fact, this information is needed to determine whether a marking is dead except for inputs. Therefore, we enhance the \mathcal{F}^+ -semantics: The idea is basically to add an additional ingredient to a tree failure (w, X) yielding a *fintree failure* (w, X, Y) . This new ingredient is a set Y , collecting traces that cannot lead the net to a final marking—including traces that cannot be performed at all. As the traces in X , we bind the traces in Y to a certain marking m that is reached by executing w . Different markings m can be reached by w because of nondeterminism, so different sets Y may be assigned to a tree failure (w, X) . This construction ensures that we can identify all traces in $dead(N)$.

Definition 63 [\mathcal{F}_{fin}^+ -semantics]

The \mathcal{F}_{fin}^+ -semantics of a labeled net N is the set of *fintree failures* defined as

$$\begin{aligned} \mathcal{F}_{fin}^+(N) = \{ & (w, X, Y) \in \Sigma^* \times \mathcal{P}(\Sigma^+) \times \mathcal{P}(\Sigma^*) \mid \exists m \in M_N : m_N \xrightarrow{w} m \\ & \wedge \forall x \in X : m \not\xrightarrow{x} \\ & \wedge \forall y \in Y : \forall m' : m \xrightarrow{y} m' \text{ implies } m' \notin \Omega_N \}. \end{aligned}$$

We say that after executing w , N *refuses* X and *fin-refuses* Y ; the set X is the *refusal set* and the set Y is the *fin-refusal set* of a fintree failure (w, X, Y) .

Figure 46 illustrates a fintree failure (w, X, Y) of a labeled net N . A marking m is reachable with the trace w from the initial marking m_N . The large white circle illustrates the set of markings of N . The smaller gray circle il-

illustrates the markings of N that are reachable from m . We depict a final marking of N as a black dot. We have $x \in X$ and $y \in Y$; that is, N refuses the word x and fin-refuses the word y after the trace w . Observe that x may be in Y as well, because it does not lead to a final marking of N either.

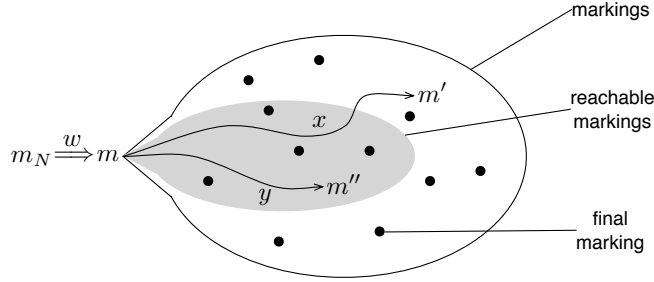


Figure 46: An illustration of a fintree failure (w, X, Y) of a labeled net N . We have $x \in X$ and $y \in Y$.

Example 64 Consider again the open nets S and S' in Fig. 42a and Fig. 44. After executing the trace ε , $env(S)$ reaches the marking $[p_0]$, $[p_1, t^0]$, or $[e^0]$. In all cases, $env(S)$ cannot refuse the set of traces r^*e : Transition r is always enabled in $env(S)$, and there is always a marking reachable that enables transition e . In contrast, after executing the trace ε , $env(S')$ reaches either the marking $[p_0]$ or the marking $[p_1, t^0]$. In both markings, it can refuse the set of traces r^*e because no reachable marking of $env(S')$ enables transition e . The empty set \emptyset is a fin-refusal set of every marking of $env(S)$ and $env(S')$. Thus, we can distinguish S and S' by their \mathcal{F}_{fin}^+ -semantics: We have $(\varepsilon, r^*e, \emptyset) \notin \mathcal{F}_{fin}^+(S)$ but $(\varepsilon, r^*e, \emptyset) \in \mathcal{F}_{fin}^+(S')$, for instance.

We can also distinguish S and S' by focusing solely on their fin-refusal sets: After executing the trace ε reaching $[p_0]$, $[p_1, t^0]$, or $[e^0]$ in $env(S)$, we can always perform some $w \in (tr)^*e$ and reach the final marking $[\]$ of $env(S)$. Thus, $env(S)$ cannot fin-refuse the set $(tr)^*e$ after ε . In contrast, $env(S')$ fin-refuses the set $(tr)^*e$ after ε because $env(S')$ can never perform e . Thus, we have $(\varepsilon, \emptyset, (tr)^*e) \notin \mathcal{F}_{fin}^+(S)$ but $(\varepsilon, \emptyset, (tr)^*e) \in \mathcal{F}_{fin}^+(S')$. Combining both fintree failures with the fintree failures from the paragraph above, we also get $(\varepsilon, r^*e, (tr)^*e) \notin \mathcal{F}_{fin}^+(S)$ but $(\varepsilon, r^*e, (tr)^*e) \in \mathcal{F}_{fin}^+(S')$. \diamond

In the remainder of this section, we show that the \mathcal{F}_{fin}^+ -semantics of a composition $N_1 \oplus N_2$ can be derived from the \mathcal{F}_{fin}^+ -semantics of the composed open nets N_1 and N_2 . Therefore, we relate the operator \oplus on open nets to the operator \uparrow on labeled nets and use that operator \uparrow is operator \parallel followed by hiding of common actions according to Def. 19.

As a first step, we show that if we consider the composition of two open nets N_1 and N_2 , then its \mathcal{F}_{fin}^+ -semantics coincides with that of the parallel composition of the two environments $env(N_1)$ and $env(N_2)$.

Lemma 65 [\mathcal{F}_{fin}^+ -semantics for open net composition]

For two composable open nets N_1 and N_2 , we have

$$\mathcal{F}_{fin}^+(env(N_1 \oplus N_2)) = \mathcal{F}_{fin}^+(env(N_1) \uparrow env(N_2)).$$

Proof. This lemma follows directly from Lem. 30: If one net has a fintree failure (w, X, Y) due to a marking m , then the other net can reach an agreeing marking m' with the trace w . If a trace $x \in X$ could be performed from m' in the second net, this would also be possible from m in the first net due to weak bisimilarity, yielding a contradiction.

If a final marking m'_1 could be reached from m' by performing $y \in Y$, then an agreeing m_1 can be reached from m . In the second net, all merged interface places p or their derived p^i and p^o are empty at m'_1 , as they are at m_1 . Hence, m_1 and m'_1 coincide on the common places and m_1 is final. This is a contradiction, and (w, X, Y) is also a fintree failure of the second net. \square

Next, we show how to determine the \mathcal{F}_{fin}^+ -semantics for the parallel composition of two labeled nets without hiding.

Lemma 66 [\mathcal{F}_{fin}^+ -semantics for labeled net composition]

For two composable labeled nets N_1 and N_2 , we have

$$\begin{aligned} \mathcal{F}_{fin}^+(N_1 \parallel N_2) = \{ & (w, X, Y) \mid \exists (w_i, X_i, Y_i) \in \mathcal{F}_{fin}^+(N_i) \text{ for } i = 1, 2 : \\ & w \in w_1 \parallel w_2 \wedge \forall x \in X, y \in Y : \\ & (x \in x_1 \parallel x_2 \text{ implies } x_1 \in X_1 \vee x_2 \in X_2) \\ & \wedge (y \in y_1 \parallel y_2 \text{ implies } y_1 \in Y_1 \vee y_2 \in Y_2) \}. \end{aligned}$$

Proof. \subseteq : Let $E = N_1 \parallel N_2$ and let (w, X, Y) be a fintree failure of E . Then there exists a marking m with $m_E \xrightarrow{w} m$ according to Def. 63. Applying Prop. 23 (only if), we find w_1 and w_2 such that $w \in w_1 \parallel w_2$, $m_{N_1} = m_E|_{P_1} \xrightarrow{w_1} m|_{P_1}$, and $m_{N_2} = m_E|_{P_2} \xrightarrow{w_2} m|_{P_2}$. For $i = 1, 2$, put $X_i = \{x \in \Sigma_i^+ \mid m|_{P_i} \not\xrightarrow{x}\}$ and $Y_i = \{y \in \Sigma_i^* \mid \forall m' : m|_{P_i} \xrightarrow{y} m' \text{ implies } m' \notin \Omega_i\}$; then, $(w_i, X_i, Y_i) \in \mathcal{F}_{fin}^+(N_i)$. Consider the implication for $x \in X$ we have to show: If $x_1 \notin X_1$, $x_2 \notin X_2$ and $x \in x_1 \parallel x_2$, we would get $m \xrightarrow{x}$ by Prop. 23 (if), a contradiction. Similarly for $y \in Y$: If $y_1 \notin Y_1$, $y_2 \notin Y_2$ and $y \in y_1 \parallel y_2$, we would get $m|_{P_i} \xrightarrow{y_i} m_i \in \Omega_i$, $i = 1, 2$; this implies $m \xrightarrow{y} m_1 + m_2 \in \Omega_E$, a contradiction.

\supseteq : Given (w, X, Y) arising from $(w_1, X_1, Y_1) \in \mathcal{F}_{fin}^+(N_1)$ and $(w_2, X_2, Y_2) \in \mathcal{F}_{fin}^+(N_2)$ due to m_1 and m_2 , one finds that w is a trace of E reaching $m_1 + m_2$. To show that each x satisfying the respective implication can be refused by $m_1 + m_2$, assume $m_1 + m_2 \not\xrightarrow{x}$ by contraposition. By Prop. 23 (only if), there are x_1 and x_2 with $x \in x_1 \parallel x_2$, $m_1 \xrightarrow{x_1}$ and $m_2 \xrightarrow{x_2}$, i.e. $x_1 \notin X_1 \wedge x_2 \notin X_2$. This justifies X , and the case of Y follows a similar argumentation. \square

We now consider hiding for the \mathcal{F}_{fin}^+ -semantics.

Lemma 67 [\mathcal{F}_{fin}^+ -semantics under hiding]

For a labeled net N and a label set $A \subseteq \Sigma^*$, we have

$$\mathcal{F}_{fin}^+(N/A) = \{(\phi(w), X, Y) \mid (w, \phi^{-1}(X), \phi^{-1}(Y)) \in \mathcal{F}_{fin}^+(N)\}.$$

Proof. We adapt this from the \mathcal{F}^+ -semantics in [246, Theorem 3.4.2], which is preserved under hiding for labeled nets—that is, $\mathcal{F}^+(N/A) = \{(\phi(w), X) \mid (w, \phi^{-1}(X)) \in \mathcal{F}^+(N)\}$. \square

We finally combine Lem. 65, Lem. 66, and Lem. 67 to show how the \mathcal{F}_{fin}^+ -semantics for the composition $N_1 \oplus N_2$ of two open nets N_1 and N_2 can be determined by the \mathcal{F}_{fin}^+ -semantics of N_1 and N_2 .

Proposition 68 [\mathcal{F}_{fin}^+ -semantics for open net composition]

For two composable open nets N_1 and N_2 , we have

$$\begin{aligned} \mathcal{F}_{fin}^+(N_1 \oplus N_2) = & \{(w, X, Y) \mid \exists (w_i, X_i, Y_i) \in \mathcal{F}_{fin}^+(N_i) \text{ for } i = 1, 2 : \\ & w \in w_1 \uparrow w_2 \wedge \forall x \in X, y \in Y : \\ & (x \in x_1 \uparrow x_2 \text{ implies } x_1 \in X_1 \vee x_2 \in X_2) \\ & \wedge (y \in y_1 \uparrow y_2 \text{ implies } y_1 \in Y_1 \vee y_2 \in Y_2)\}. \end{aligned}$$

Proof. Lemma 66 shows that the right part of this equation—with \parallel replacing \uparrow —is equal to $\mathcal{F}_{fin}^+(env(N_1) \parallel env(N_2))$; then, one can hide the common actions of $env(N_1)$ and $env(N_2)$, and by Lem. 67 the right hand side is equal to $\mathcal{F}_{fin}^+(env(N_1) \uparrow env(N_2))$; the latter is equal to $\mathcal{F}_{fin}^+(env(N_1 \oplus N_2)) = \mathcal{F}_{fin}^+(N_1 \oplus N_2)$ by Lem. 65. \square

4.2.2 Refinement on the \mathcal{F}_{fin}^+ -semantics

Having introduced the \mathcal{F}_{fin}^+ -semantics for open nets, we define a refinement relation between two open nets based on their \mathcal{F}_{fin}^+ -semantics, and show that this refinement relation coincides with compositional conformance.

Like for the *stopdead*-semantics in Sect. 4.1.2, inclusion of the \mathcal{F}_{fin}^+ -semantics of two open nets implies a refinement relation. However, in contrast to the *stopdead*-semantics, inclusion of the \mathcal{F}_{fin}^+ -semantics is a sufficient but not a necessary criterion for compositional conformance.

For the compositional conformance relation, the fintree failures used in the \mathcal{F}_{fin}^+ -semantics give too detailed information about the moment of choice in an open net: For example, the fintree failure $(\varepsilon, re, \emptyset) \in \mathcal{F}_{fin}^+(S')$, as used in Ex. 64, tells us already that the trace re can be refused from the initial marking of $env(S')$. We remove this information by closing up under an ordering over fintree failures: We say a fintree failure (w, X, Y) is *dominated* by a fintree failure $(wx, x^{-1}X, x^{-1}Y)$ for $x \in \{\varepsilon\} \cup \downarrow X \cup \downarrow Y$. We then define a relation between two interface-equivalent open nets *Impl* and *Spec* not by inclusion of their respective \mathcal{F}_{fin}^+ -semantics but in such a way that every fintree failure of *Impl* is dominated by a fintree failure of *Spec*. The resulting refinement relation $\sqsubseteq_{\mathcal{F}_{fin}^+}$ is an adaption of the refinement relation $\sqsubseteq_{\mathcal{F}^+}$ from Rensink and Vogler [217], incorporating the fin-refusal sets of two fintree failures into the definition of $\sqsubseteq_{\mathcal{F}^+}$.

Definition 69 [\mathcal{F}_{fin}^+ -refinement]

For two action-equivalent labeled nets *Impl* and *Spec*, *Impl* \mathcal{F}_{fin}^+ -refines *Spec*, denoted by $Impl \sqsubseteq_{\mathcal{F}_{fin}^+} Spec$, if

$$\begin{aligned} \forall (w, X, Y) \in \mathcal{F}_{fin}^+(Impl) : \\ \exists x \in \{\varepsilon\} \cup \downarrow X \cup \downarrow Y : (wx, x^{-1}X, x^{-1}Y) \in \mathcal{F}_{fin}^+(Spec). \end{aligned}$$

For two interface-equivalent open nets $Impl$ and $Spec$, we define $Impl \sqsubseteq_{\mathcal{F}_{fin}^+} Spec$, if $env(Impl) \sqsubseteq_{\mathcal{F}_{fin}^+} env(Spec)$.

Example 70 We have $(\varepsilon, \emptyset, (tr)^*e) \in \mathcal{F}_{fin}^+(S')$ by Ex. 64. Assume S' \mathcal{F}_{fin}^+ -refines S . Then there exists an $x \in \{\varepsilon\} \cup \downarrow \emptyset \cup \downarrow (tr)^*e$ such that $(x, x^{-1}\emptyset, x^{-1}(tr)^*e) \in \mathcal{F}_{fin}^+(S)$ according to Def. 69. By the suffix closure, we have $x = e$, $x \in (tr)^*$, or $x \in (tr)^*e$. We distinguish these three cases:

- If $x = e$, then $(e, \emptyset, \{\varepsilon\}) \in \mathcal{F}_{fin}^+(S)$. However, we have $(e, \emptyset, \{\varepsilon\}) \notin \mathcal{F}_{fin}^+(S)$, because we always reach the final marking $[\]$ after trace e in $env(S)$.
- If $x \in (tr)^*$, then $(x, \emptyset, (tr)^*e) \in \mathcal{F}_{fin}^+(S)$. However, for all $x \in (tr)^*$, we have $(x, \emptyset, (tr)^*e) \notin \mathcal{F}_{fin}^+(S)$: After x , we reach any of the markings $[p_0]$, $[p_1, t]$, or $[e]$ in $env(S)$, from which we can always reach the final marking $[\]$ with trace e or tre .
- If $x \in (tr)^*e$, then $(x, \emptyset, r(tr)^*e) \in \mathcal{F}_{fin}^+(S)$. However, for all $x \in (tr)^*e$, we have $(x, \emptyset, r(tr)^*e) \notin \mathcal{F}_{fin}^+(S)$: After x , we are in the marking $[p_1]$ from which we reach the final marking $[\]$ with trace re .

Thus, S' does not \mathcal{F}_{fin}^+ -refine S . \diamond

Having characterized the \mathcal{F}_{fin}^+ -semantics for an open net composition in Prop. 68, we shall show that \mathcal{F}_{fin}^+ -refinement is a precongruence for the open net composition operator \oplus . First, we show the precongruence result for labeled nets and operator \parallel . Then, we show that this result is also preserved under hiding. Finally, we combine these results to show the precongruence for open nets and the operator \oplus .

Our definition of \mathcal{F}_{fin}^+ -refinement in Def. 69 is an adaption of the refinement relation $\sqsubseteq_{\mathcal{F}^+}$ in [217]. The refinement relation $\sqsubseteq_{\mathcal{F}^+}$ coincides with should (or fair) testing [196, 48, 217] as proved in [217, Theorem 36], and should testing is a precongruence for labeled net composition [217]. Therefore, for the first and second step, we can build upon the proof ideas introduced for should testing in [217, Lemma 46], where saturation conditions like Lem. 71(1–3) below are employed. The key idea in [217] is to shift traces from the refusal set of $Impl$. We apply the same proof strategy for the X -part of the fintree failures, which is closed under suffix by Lem. 71(3). Because this does not hold for the Y -part, we cannot directly apply this idea here. We overcome this problem by adding the refusal set X to the fin-refusal set Y , thereby using the fourth of the following saturation conditions on fintree failures.

Lemma 71(1) states that, given a fintree failure (w, X, Y) , the sets X and Y can be arbitrarily decreased and the resulting triple is again a fintree failure. Furthermore, the refusal part of \mathcal{F}_{fin}^+ is saturated in the sense that the sets X and Y can be extended by any set of traces z such that $(wz, z^{-1}X, z^{-1}Y) \notin \mathcal{F}_{fin}^+(N)$ by Lem. 71(2). Lemma 71(3) states that the X -part is closed under suffix, and Lem. 71(4) shows that the refusal part of \mathcal{F}_{fin}^+ is saturated in the sense that the refusal set X can be added to fin-refusal set Y .

Lemma 71 [saturation conditions]

For a labeled net N , we have

1. $(w, X, Y) \in \mathcal{F}_{fin}^+(N), X' \subseteq X, Y' \subseteq Y$ implies $(w, X', Y') \in \mathcal{F}_{fin}^+(N)$
2. $(w, X, Y) \in \mathcal{F}_{fin}^+(N) \wedge \forall z \in Z : (wz, z^{-1}X, z^{-1}Y) \notin \mathcal{F}_{fin}^+(N)$ implies $(w, X \cup Z, Y \cup Z) \in \mathcal{F}_{fin}^+(N)$
3. $(w, X, Y) \in \mathcal{F}_{fin}^+(N)$ implies $(w, \uparrow X, Y) \in \mathcal{F}_{fin}^+(N)$
4. $(w, X, Y) \in \mathcal{F}_{fin}^+(N)$ implies $(w, X, X \cup Y) \in \mathcal{F}_{fin}^+(N)$

Proof. Items (1), (3), and (4) are obvious by Def. 63. To see item (2), assume that some z could be performed from the marking m justifying the fintree failure (w, X, Y) . \square

We have also explored the idea to encode each $w \in Y$ by $w\checkmark$ for a new symbol \checkmark . Then, one can add the resulting traces to X and work with something that looks like an ordinary tree failure. The hope was that this would allow us to use the result [217, Lemma 46] instead of its proof idea, but we have not managed to show the necessary saturation conditions for the domain used in [217].

With the saturation conditions in Lem. 71, we prove that \mathcal{F}_{fin}^+ -refinement is a precongruence for labeled nets with respect to the operator \parallel .

Lemma 72

\mathcal{F}_{fin}^+ -refinement is a precongruence for labeled nets with respect to \parallel .

Proof. Let $Impl$ and $Spec$ be two action-equivalent labeled nets, and let labeled net C be composable with $Spec$ (and therefore with $Impl$). Let further $Impl \sqsubseteq_{\mathcal{F}_{fin}^+} Spec$. We show that $Impl \parallel C \sqsubseteq_{\mathcal{F}_{fin}^+} Spec \parallel C$, following to a large extent the proof of [217, Lemma 46]. For understandability, we also show the full proof for the case $\Sigma_{Spec} = \Sigma_C$ here, because in this case the projection functions in the construction of the synchronized fintree failures become the identity over the complete alphabet and hence disappear.

Consider a fintree failure $(w, X_{Impl} \cup X_C, Y_{Impl} \cup Y_C) \in \mathcal{F}_{fin}^+(Impl \parallel C)$ such that $(w, X_{Impl}, Y_{Impl}) \in \mathcal{F}_{fin}^+(Impl)$ and $(w, X_C, Y_C) \in \mathcal{F}_{fin}^+(C)$. Define the set

$$W = \{v \mid (wv, v^{-1}X_C, v^{-1}Y_C) \notin \mathcal{F}_{fin}^+(C)\}$$

which contains those traces that can be added to X_C and Y_C according to Lem. 71(2). We shift the traces in W from $Impl$ to C . To this end, we define four sets

$$\begin{aligned} X'_{Impl} &= X_{Impl} \setminus \uparrow W, \\ Y'_{Impl} &= Y_{Impl} \setminus \uparrow W, \\ X'_C &= X_C \cup \uparrow W, \\ Y'_C &= Y_C \cup \uparrow W. \end{aligned}$$

We immediately see: $X_{Impl} \cup X_C \subseteq X'_{Impl} \cup X'_C$, $Y_{Impl} \cup Y_C \subseteq Y'_{Impl} \cup Y'_C$, and $X'_{Impl} \cup Y'_{Impl} \subseteq X_{Impl} \cup Y_{Impl} \cup X_C \cup Y_C$. We have $(w, X'_{Impl}, Y'_{Impl}) \in \mathcal{F}_{fin}^+(Impl)$

by Lem. 71(1). Due to $Impl \sqsubseteq_{\mathcal{F}_{fin}^+} Spec$, there exists $x \in \{\varepsilon\} \cup \downarrow X'_{Impl} \cup \downarrow Y'_{Impl}$ such that

$$(wx, x^{-1}X'_{Impl}, x^{-1}Y'_{Impl}) \in \mathcal{F}_{fin}^+(Spec). \quad (1)$$

We have $x \notin \uparrow W$. Assume the contrary: $x = \varepsilon$ implies $\varepsilon \in W$ which is a contradiction to the construction of W ; $x \in \downarrow X'_{Impl}$ implies $\exists x' \in X'_{Impl} : x \sqsubseteq x' \wedge \exists v \in W : v \sqsubseteq x \sqsubseteq x'$ which is a contradiction to the definition of X'_{Impl} . The same argument also applies to $x \in Y'_{Impl}$.

From $x \notin W$, it follows that $(wx, x^{-1}X_C, x^{-1}Y_C) \in \mathcal{F}_{fin}^+(C)$. Further, for all $u \in x^{-1}W$ (i.e., $xu \in W$), $(wxu, u^{-1}x^{-1}X_C, u^{-1}x^{-1}Y_C) \notin \mathcal{F}_{fin}^+(C)$.

By Lem. 71(2), $(wx, x^{-1}(X_C \cup W), x^{-1}(Y_C \cup W)) \in \mathcal{F}_{fin}^+(C)$. Consider now the second ingredient, $x^{-1}(X_C \cup W)$. By Lem. 71(3), this implies the fin-tree failure $(wx, \uparrow x^{-1}(X_C \cup W), x^{-1}(Y_C \cup W)) \in \mathcal{F}_{fin}^+(C)$. With Lem. 2(3), we have $\uparrow x^{-1}(X_C \cup W) \supseteq x^{-1}(X_C \cup \uparrow W) = x^{-1}X'_C$. Now, according to Lem. 71(1), $x^{-1}X'_C$ can replace $x^{-1}(X_C \cup W)$.

Consider now the third ingredient, $x^{-1}(Y_C \cup W)$. By Lem. 71(4), we extend this set to $x^{-1}(Y_C \cup W) \cup x^{-1}X'_C \supseteq x^{-1}\uparrow W \cup x^{-1}Y_C = x^{-1}(\uparrow W \cup Y_C) = x^{-1}Y'_C$. Now, Lem. 71(1) allows that $x^{-1}Y'_C$ can replace $x^{-1}(Y_C \cup W)$.

Combining these results, we get $(wx, x^{-1}X'_C, x^{-1}Y'_C) \in \mathcal{F}_{fin}^+(C)$. By Lem. 66 and (1), we obtain $(wx, x^{-1}(X'_{Impl} \cup X'_C), x^{-1}(Y'_{Impl} \cup Y'_C)) \in \mathcal{F}_{fin}^+(Spec \parallel C)$. By Lem. 71(1), we have $(wx, x^{-1}(X_{Impl} \cup X_C), x^{-1}(Y_{Impl} \cup Y_C)) \in \mathcal{F}_{fin}^+(Spec \parallel C)$ where $x \in (\{\varepsilon\} \cup \downarrow X'_{Impl} \cup \downarrow Y'_{Impl}) \subseteq (\{\varepsilon\} \cup \downarrow X_{Impl} \cup \downarrow Y_{Impl} \cup \downarrow X_C \cup \downarrow Y_C)$.

Now consider the general case. Let π and π_C denote projections, projecting words onto the alphabets $\Sigma_{Spec} = \Sigma_{Impl}$ and Σ_C , respectively. We have

$$\pi(V \cup W) = \pi(V) \cup \pi(W) \quad (2)$$

$$\pi(\uparrow V) \subseteq \uparrow \pi(V) \quad (3)$$

$$\pi(w^{-1}V) \subseteq \pi(w)^{-1}\pi(V) \quad (4)$$

Consider a fintree failure $(w, X_{Impl} \cup X_C, Y_{Impl} \cup Y_C) \in \mathcal{F}_{fin}^+(Impl \parallel C)$ such that $(\pi(w), \pi(X_{Impl}), \pi(Y_{Impl})) \in \mathcal{F}_{fin}^+(Impl)$ and $(\pi_C(w), \pi_C(X_C), \pi_C(Y_C)) \in \mathcal{F}_{fin}^+(C)$. Define the set

$$W = \{v \mid (\pi_C(wv), \pi_C(v^{-1}X_C), \pi_C(v^{-1}Y_C)) \notin \mathcal{F}_{fin}^+(C)\}.$$

We shift the traces in W from $Impl$ to C . To this end, we define four sets

$$\begin{aligned} X'_{Impl} &= X_{Impl} \setminus \uparrow W, \\ Y'_{Impl} &= Y_{Impl} \setminus \uparrow W, \\ X'_C &= X_C \cup \uparrow W, \\ Y'_C &= Y_C \cup \uparrow W. \end{aligned}$$

We immediately see: $X_{Impl} \cup X_C \subseteq X'_{Impl} \cup X'_C$, $Y_{Impl} \cup Y_C \subseteq Y'_{Impl} \cup Y'_C$, and $X'_{Impl} \cup Y'_{Impl} \subseteq X_{Impl} \cup Y_{Impl} \cup X_C \cup Y_C$. We have $(\pi(w), \pi(X'_{Impl}), \pi(Y'_{Impl})) \in \mathcal{F}_{fin}^+(Impl)$ by Lem. 71(1). Because of $Impl \sqsubseteq_{\mathcal{F}_{fin}^+} Spec$, there exists an $x \in \{\varepsilon\} \cup \downarrow X'_{Impl} \cup \downarrow Y'_{Impl}$ such that $(\pi(wx), \pi(x)^{-1}\pi(X'_{Impl}), \pi(x)^{-1}\pi(Y'_{Impl})) \in \mathcal{F}_{fin}^+(Spec)$. Hence, we have

$$(\pi(wx), \pi(x^{-1}X'_{Impl}), \pi(x^{-1}Y'_{Impl})) \in \mathcal{F}_{fin}^+(Spec) \quad (5)$$

due to (4) and Lem. 71(1).

Again, trace $x \notin \uparrow W$ (by the same argumentation as in the proof of the case $\Sigma_{Spec} = \Sigma_C$), and we conclude that $(\pi_C(wx), \pi_C(x^{-1}X_C), \pi_C(x^{-1}Y_C)) \in \mathcal{F}_{fin}^+(C)$. Further, for all $u \in x^{-1}W$ (i.e., $xu \in W$), we have the fintree failure $(\pi_C(wxu), \pi_C(u^{-1}x^{-1}X_C), \pi_C(u^{-1}x^{-1}Y_C)) \notin \mathcal{F}_{fin}^+(C)$ due to the definition of W . Now, $(\pi_C(wxu), \pi_C(u)^{-1}\pi_C(x^{-1}X_C), \pi_C(u)^{-1}\pi_C(x^{-1}Y_C)) \notin \mathcal{F}_{fin}^+(C)$ with (4) and Lem. 71(1), and we obtain, by (2) and Lem. 71(2),

$$(\pi_C(wx), \pi_C(x^{-1}(X_C \cup W)), \pi_C(x^{-1}(Y_C \cup W))) \in \mathcal{F}_{fin}^+(C).$$

Consider the second ingredient, $\pi_C(x^{-1}(X_C \cup W))$ of this fintree failure. Applying Lem. 71(3), we obtain $\uparrow \pi_C(x^{-1}(X_C \cup W))$ and with Lem. 71(1) and (3), $\pi_C(\uparrow x^{-1}(X_C \cup W))$. Because $x \notin \uparrow W$, we can apply Lem. 2(3) and Lem. 71(1), and we arrive at $\pi_C(x^{-1}(X_C \cup \uparrow W)) = \pi_C(x^{-1}X'_C)$.

For the third ingredient $\pi_C(x^{-1}(Y_C \cup W))$ of this fintree failure, we obtain by Lem. 71(4) $\pi_C(x^{-1}(Y_C \cup W)) \cup \pi_C(x^{-1}X'_C)$. By (2), we can transform this into $\pi_C(x^{-1}(Y_C \cup W \cup X'_C))$ and by Lem. 71(1) into $\pi_C(x^{-1}(Y_C \cup \uparrow W)) = \pi_C(x^{-1}Y'_C)$.

Combining these results yields $(\pi_C(wx), \pi_C(x^{-1}X'_C), \pi_C(x^{-1}Y'_C)) \in \mathcal{F}_{fin}^+(C)$. Then, with Lem. 66 and (5), we obtain $(wx, x^{-1}(X'_{Impl} \cup X'_C), x^{-1}(Y'_{Impl} \cup Y'_C)) \in \mathcal{F}_{fin}^+(Spec \parallel C)$.

Applying Lem. 71(1) yields that $(wx, x^{-1}(X_{Impl} \cup X_C), x^{-1}(Y_{Impl} \cup Y_C)) \in \mathcal{F}_{fin}^+(Spec \parallel C)$ where $x \in (\{\varepsilon\} \cup \downarrow X'_{Impl} \cup \downarrow Y'_{Impl}) \subseteq (\{\varepsilon\} \cup \downarrow X_{Impl} \cup \downarrow Y_{Impl} \cup \downarrow X_C \cup \downarrow Y_C)$. \square

We already remarked in [249], that the proof of Lem. 72 is not restricted to sets of fintree failures of labeled nets, but holds for general sets of fintree failures in labeled transition systems, for which the four saturation conditions in Lem. 71 hold.

Lemma 73

\mathcal{F}_{fin}^+ -refinement for labeled nets is preserved under hiding.

Proof. Let $Impl$ and $Spec$ be two action-equivalent labeled nets such that $Impl \sqsubseteq_{\mathcal{F}_{fin}^+} Spec$, let $A \subseteq \Sigma^*$ and $(w, X, Y) \in \mathcal{F}_{fin}^+(Impl/A)$. Consider the fintree failure $(v, \phi^{-1}(X), \phi^{-1}(Y)) \in \mathcal{F}_{fin}^+(Impl)$ with $w = \phi(v)$. Because $Impl \mathcal{F}_{fin}^+$ -refines $Spec$, there exists an $x \in \{\varepsilon\} \cup \downarrow \phi^{-1}(X) \cup \downarrow \phi^{-1}(Y)$ with $(vx, x^{-1}\phi^{-1}(X), x^{-1}\phi^{-1}(Y)) \in \mathcal{F}_{fin}^+(Spec)$. It can be shown that $\phi^{-1}(\phi(x)^{-1}X) = x^{-1}\phi^{-1}(X)$. Using this observation together with $(v, \phi^{-1}(X), \phi^{-1}(Y)) \in \mathcal{F}_{fin}^+(Impl)$, we conclude that $(\phi(vx), \phi(x)^{-1}X, \phi(x)^{-1}Y) \in \mathcal{F}_{fin}^+(Spec/A)$ and also $(\phi(v)\phi(x), \phi(x)^{-1}X, \phi(x)^{-1}Y) \in \mathcal{F}_{fin}^+(Spec/A)$. Because $\phi(v) = w$ and $\phi(x) \in \{\varepsilon\} \cup \downarrow X \cup \downarrow Y$, the lemma holds. \square

Lemma 72 and Lem. 73 enable us to show the first main result of this section: \mathcal{F}_{fin}^+ -refinement is a precongruence for the open net composition operator \oplus . The proof idea is to translate the operator \oplus on open nets into the operator \uparrow on labeled nets followed by hiding.

Theorem 74 [\mathcal{F}_{fin}^+ -refinement is a precongruence]

\mathcal{F}_{fin}^+ -refinement is a precongruence for open nets with respect to \oplus .

Proof. Let $Impl$ and $Spec$ be two interface-equivalent open nets with $Impl \sqsubseteq_{\mathcal{F}_{fin}^+} Spec$, and let C be an open net composable with both. We have to show that $Impl \oplus C \sqsubseteq_{\mathcal{F}_{fin}^+} Spec \oplus C$.

By Lem. 65, $\mathcal{F}_{fin}^+(env(Spec \oplus C)) = \mathcal{F}_{fin}^+(env(Spec) \uparrow env(C))$. Let A denote the common actions of $env(Spec)$ and $env(C)$. Then we can replace operator \uparrow with operator \parallel and make the hiding explicit, which results in $\mathcal{F}_{fin}^+(env(Spec) \uparrow env(C)) = \mathcal{F}_{fin}^+((env(Spec) \parallel env(C))/A)$ by Def. 19. Likewise, we derive $\mathcal{F}_{fin}^+(env(Impl \oplus C)) = \mathcal{F}_{fin}^+((env(Impl) \parallel env(C))/A)$. We have $env(Impl) \sqsubseteq_{\mathcal{F}_{fin}^+} env(Spec)$ by assumption and, due to the precongruence results in Lem. 72 and Lem. 73, we obtain that $(env(Impl) \parallel env(C))/A \sqsubseteq_{\mathcal{F}_{fin}^+} (env(Spec) \parallel env(C))/A$. As this only depends on the \mathcal{F}_{fin}^+ -semantics of the two nets, we directly have $Impl \oplus C \sqsubseteq_{\mathcal{F}_{fin}^+} Spec \oplus C$ with Lem. 65. \square

With the next theorem, we show the second main result of this section: \mathcal{F}_{fin}^+ -refinement coincides with the coarsest precongruence that is contained in the conformance relation—that is, compositional conformance.

Theorem 75 [\mathcal{F}_{fin}^+ -refinement is the coarsest precongruence]

For two interface-equivalent open nets $Impl$ and $Spec$, we have

$$Impl \sqsubseteq_{conf}^c Spec \quad \text{iff} \quad Impl \sqsubseteq_{\mathcal{F}_{fin}^+} Spec.$$

Proof. \Leftarrow : Consider a trace $w \in stop(Impl)$ ($w \in dead(Impl)$); we prove $w \in stop(Spec)$ ($w \in dead(Spec)$). Then, applying Thm. 61, we get $Impl \sqsubseteq_{conf} Spec$, and this in turn also shows the claim with Thm. 74 and the definition of \sqsubseteq_{conf}^c . So let O be the set of output places of $Impl$ and of $Spec$.

We have $w \in stop(Impl)$ if and only if $(w, O, \emptyset) \in \mathcal{F}_{fin}^+(Impl)$ by Def. 55 and Def. 63. Then, by $Impl \sqsubseteq_{\mathcal{F}_{fin}^+} Spec$, there must be a suitable $x \in \{\varepsilon\} \cup O = \{\varepsilon\} \cup \downarrow O$ that satisfies the defining condition of Def. 63. We cannot have $x \in O$ because $(wx, \{\varepsilon\}, \emptyset) \notin \mathcal{F}_{fin}^+(Spec)$ by Def. 63. Thus, $x = \varepsilon$ and $(w, O, \emptyset) \in \mathcal{F}_{fin}^+(Spec)$, implying $w \in stop(Spec)$. Analogously, we have $w \in dead(Impl)$ if and only if $(w, O, \{\varepsilon\}) \in \mathcal{F}_{fin}^+(Impl)$ by Def. 55. Again, $x = \varepsilon$ and thus $(w, O, \{\varepsilon\}) \in \mathcal{F}_{fin}^+(Spec)$, implying $w \in dead(Spec)$.

\Rightarrow : Suppose $Impl \sqsubseteq_{conf}^c Spec$, and let $(w, X, Y) \in \mathcal{F}_{fin}^+(Impl)$. In addition, consider an open net C with the new output x and the new input y . Open net C has the empty initial marking and contains only a single transition that can indefinitely repeat to produce a token in x while consuming a token from place y . In addition, its final marking is the empty marking. The idea is to construct an open net N from (w, X, Y) such that C is not a partner of $Impl \oplus N$ because of (w, X, Y) . By $Impl \sqsubseteq_{conf}^c Spec$ and because \sqsubseteq_{conf}^c is a precongruence, we have $Impl \oplus N \sqsubseteq_{conf}^c Spec \oplus N$ and thus $Impl \oplus N \sqsubseteq_{conf} Spec \oplus N$ by Def. 36. Hence, C is also not a partner of $Spec \oplus N$, and from this we shall conclude that (w, X, Y) is dominated by a fintree failure in $\mathcal{F}_{fin}^+(Spec)$ according to Def. 69. Then we will have proved $Impl \sqsubseteq_{\mathcal{F}_{fin}^+} Spec$.

The open net N has input places $I = O_{Impl} \uplus \{x\}$, output places $O = I_{Impl} \uplus \{y\}$, and enables a transition sequence $v = t_1 \dots t_k$. Each transition in v is connected to an interface place of N such that the corresponding trace of interface actions is w ; that is, the net N contains net N_w as shown

in Fig. 43a. Thus, we can essentially fire the trace w of $env(N)$ in $Impl \oplus N$ and, therefore, in $Impl \oplus N \oplus C$ by firing v instead of the labeled transitions. This way, we reach in $Impl$ a marking m that refuses X in $env(Impl)$; in N , there is only one token in a place p_ϵ and the token in a place p has been consumed. This token is necessary to enable transition t' that is—together with transition t —essential for responsiveness, because these transitions can repeatedly communicate with C . The place p can only be marked again by firing some transition t'_x with $x \in X$, and this in turn requires the firing of a transition sequence that—similarly to v —looks to $Impl$ like the trace x . But this trace cannot be fired at m . In addition, every trace $y \in Y$ that cannot lead to a final marking in $Impl$ leads to a final marking in the tree part of N . This construction guarantees that there is a marking reachable in the composition $Impl \oplus N \oplus C$ which is neither communicating (because place p is not marked and hence there is no communication between C and N) nor reaches a final marking (because if $N \oplus C$ is in a final marking, then $Impl$ is not). As a consequence, $Impl \oplus N \oplus C$ is not responsive and, thus, C is not a partner of $Impl \oplus N$.

To achieve the effect just described, the second part of the open net N encodes the tree part for X and Y of fintree failure (w, X, Y) ; this second part is a tree representing $X \cup Y$. Common prefixes thereby correspond to the same path in this part. If a path corresponds to some $y \in Y$, a token on the place at the end of this path is a final marking of N ; if for example $b \in Y$, then the marking with just one token on the place p_b is final. For a path corresponding to some $x \in X$, a token in the respective place allows to mark p again. Figure 47 illustrates this construction; it is an adaptation of a construction that is used in [246, Fig. 3.19].

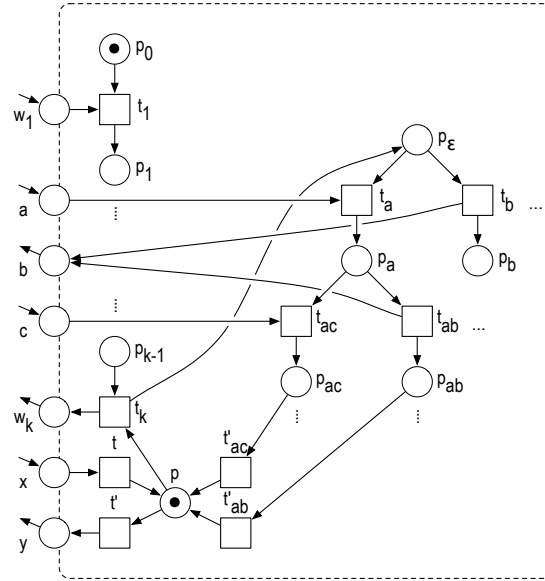


Figure 47: Illustration of the construction of open net N ; the marking $[p_a]$ can be a final marking of N .

Let $w = w_1 \dots w_k$ such that for $j = 1, \dots, k$, $w_j \in I_{Impl} \uplus O_{Impl}$. Define the open net $N = (P, T, F, m_N, \Omega, O, I)$ by

- $P = \{p\} \uplus \{p_i \mid 0 \leq i \leq k-1\} \uplus \{p_u \mid u \in \downarrow X \cup \downarrow Y \cup \{\epsilon\}\}$

- $T = \{t, t'\} \uplus \{t_i \mid 1 \leq i \leq k\} \uplus \{t_u \mid u \in \downarrow X \cup \downarrow Y \wedge u \neq \varepsilon\} \uplus \{t'_z \mid z \in X\}$
- $F = \{(p_i, t_{i+1}) \mid 0 \leq i \leq k-1\} \uplus \{(t_i, p_i) \mid 1 \leq i \leq k-1\} \uplus \{(x, t), (t, p), (p, t'), (t', y), (p, t_k), (t_k, p_\varepsilon)\} \uplus \{(p_u, t_{ua}) \mid a \in I_{Impl} \uplus O_{Impl} \wedge ua \in \downarrow X \cup \downarrow Y\} \uplus \{(t_u, p_u) \mid u \in \downarrow X \cup \downarrow Y \wedge u \neq \varepsilon\} \uplus \{(p_z, t'_z), (t'_z, p) \mid z \in X\} \uplus \{(w_i, t_i) \mid 1 \leq i \leq k \wedge w_i \in O_{Impl}\} \uplus \{(t_i, w_i) \mid 1 \leq i \leq k \wedge w_i \in I_{Impl}\} \uplus \{(a, t_{ua}) \mid a \in O_{Impl} \wedge ua \in \downarrow X \cup \downarrow Y\} \uplus \{(t_{ua}, a) \mid a \in I_{Impl} \wedge ua \in \downarrow X \cup \downarrow Y\},$
- $m_N = [p_0, p]$, and
- $\Omega = \{[p_z] \mid z \in Y\}$.

As argued previously, we now have that C is not a partner of $Spec \oplus N$; that is, some marking m_1 can be reached in $Spec \oplus N \oplus C$ where responsiveness is violated. Clearly, places p , x , and y must be empty in m_1 ; thus, v has been fired in N plus possibly some transitions in the fintree part of the net. There is just one token in the places of $inner(N)$, and it is in some p_u with $uu' \in X$ (resp. $uu' \in Y$). Let m_2 be the projection of m_1 onto the places of $Spec$. From the point of view of $Spec$, we have fired a trace wu of $env(Spec)$ reaching m_2 . Because in $Spec \oplus N \oplus C$ no $t'_{uu'}$ can become enabled and the composition cannot reach a final marking—otherwise, C would be a partner—no u' can be fired in $env(Spec)$ at m_2 and a final marking is not reachable. Thus, we conclude $(wu, \{u' \mid uu' \in X\}, \{u' \mid uu' \in Y\}) \in \mathcal{F}_{fin}^+(Spec)$ and, therefore, $Impl \sqsubseteq_{\mathcal{F}_{fin}^+} Spec$. \square

Example 76 We already showed in Ex. 40 that for the open nets S and S' , $S' \sqsubseteq_{conf}^c S$ does not hold. We can now confirm this with Thm. 75, because S' does not \mathcal{F}_{fin}^+ -refine S by Ex. 70. \diamond

With Thm. 61, we have characterized the conformance relation for responsiveness introduced in Def. 31, and with Thm. 75 the coarsest precongruence contained in that relation (i.e., compositional conformance). However, it turns out that both relations are not suitable for (compositional) verification: We show their undecidability in the following section.

4.3 UNDECIDABILITY OF CONFORMANCE AND COMPOSITIONAL CONFORMANCE

In this section, we show conformance and compositional conformance to be undecidable by reducing both to the halting problem of Minsky's counter machines [178]. For the reduction, we use the trace-based characterization of conformance in Thm. 61 and the failure-based characterization of compositional conformance in Thm. 75. We start by introducing counter machines and their halting problem in Sect. 4.3.1. Next, we show that conformance is undecidable in Sect. 4.3.2 and that compositional conformance is undecidable in Sect. 4.3.3.

4.3.1 Counter machines and their halting problem

We define a counter machine as in [178].

Definition 77 [counter machine]

Let $m, n \in \mathbb{N}^+$. A counter machine C with m counters c_1, \dots, c_m (m -counter machine for short) is a program consisting of n commands

$$\begin{aligned} 1 &: \text{CMD}_1; \\ 2 &: \text{CMD}_2; \\ &\dots \\ n &: \text{CMD}_n \end{aligned}$$

where CMD_n is a *HALT*-command and $\text{CMD}_1, \dots, \text{CMD}_{n-1}$ are commands of the following two types (where $1 \leq k, k_1, k_2 \leq n, 1 \leq j \leq m$):

TYPE 1: $c_j := c_j + 1$; goto k

TYPE 2: if $c_j = 0$ then goto k_1 else ($c_j := c_j - 1$; goto k_2)

Define the set $BS(C)$ of *branching states* of C as $BS(C) = \{i \in \mathbb{N}^+ \mid \text{CMD}_i \text{ is of type 2}\}$.

As a running example, consider the 2-counter machine *ADD* in Alg. 1. The 2-counter machine *ADD* consists of three commands: one of each type and the *HALT*-command. It expects two given integers x_1 and x_2 as inputs, and returns their sum $x_1 + x_2$ stored in the counter c_2 . The set of branching states of *ADD* is the singleton $BS(ADD) = \{1\}$, and obviously *ADD* halts on any inputs.

Input : An integer x_1 stored in c_1 , an integer x_2 stored in c_2

Output : The integer $x_1 + x_2$ stored in c_2

- 1 if $c_1 = 0$ then goto 3 else ($c_1 := c_1 - 1$; goto 2) ;
- 2 $c_2 := c_2 + 1$; goto 1 ;
- 3 *HALT*

Algorithmus 1 : The 2-counter machine *ADD* for adding two integers x_1 and x_2 .

In the following, we describe a basic net consisting of three labeled net patterns—one pattern for each *CMD*-type and an auxiliary notion of a “definitely cheating” pattern—which we use to simulate a counter machine. These patterns are an extension of the “Jančar-Patterns” [126], which we show in Fig. 48a—Fig. 48c. In the original patterns, every transition is labeled with itself. For each transition t of the original patterns, we add two transitions and two places controlling t ’s firing. In addition, we shift the label from t to the newly introduced transitions, and label t with τ . Figure 48d—Fig. 48f illustrate the extended patterns.

Definition 78 [basic net]

Let C be an m -counter machine with n commands. The *basic net* $net(C)$ of C is a labeled net constructed as follows (assuming $1 \leq k, k_1, k_2 \leq n, 1 \leq j \leq m$):

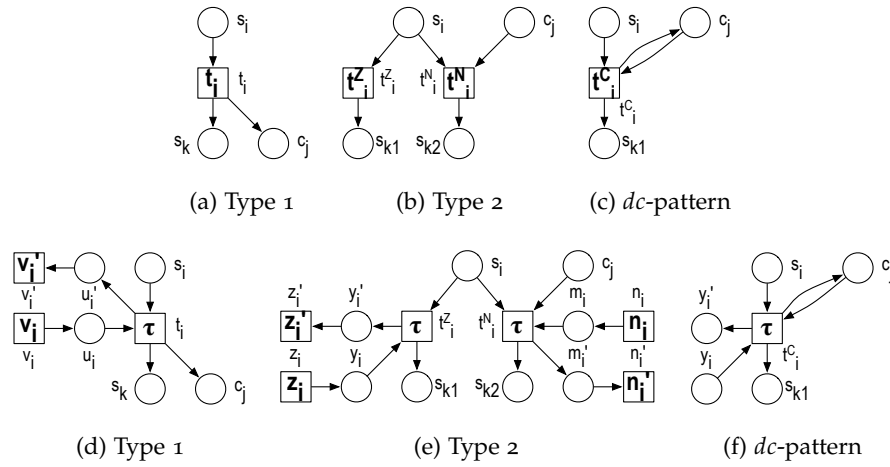


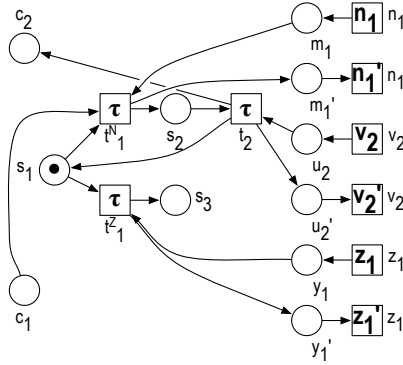
Figure 48: The “Jančar-Patterns” as introduced by Jančar [126] (first row), and the extended patterns for the constructions of $net(C)$ in Def. 78 (second row).

1. Let c_1, \dots, c_m (the counter part) and s_1, \dots, s_n (the state part) be places of $net(C)$.
2. For $i = 1, \dots, n - 1$ add new transitions and arcs depending on the type of the command CMD_i :
 - TYPE 1: $c_j := c_j + 1$; goto k
 Add places u_i, u_i' , transitions t_i, v_i, v_i' , and arcs (v_i, u_i) , (u_i, t_i) , (t_i, u_i') , (u_i', v_i') , (s_i, t_i) , (t_i, s_{k1}) , and (t_i, c_j) . For the labeling, we set $l(v_i) = v_i$, $l(v_i') = v_i'$, and $l(t_i) = \tau$.
 - TYPE 2: if $c_j = 0$ then goto k_1 else $(c_j := c_j - 1$; goto $k_2)$
 Add places y_i, y_i', m_i, m_i' , transitions t_i^Z, z_i, z_i' (to simulate the case in which counter c_j is zero) and t_i^N, n_i, n_i' (to simulate the case in which counter c_j is not empty), and arcs (z_i, y_i) , (y_i, t_i^Z) , (t_i^Z, y_i') , (y_i', z_i') , (n_i, m_i) , (m_i, t_i^N) , (t_i^N, m_i') , (m_i', n_i') , (s_i, t_i^Z) , (t_i^Z, s_{k1}) , (s_i, t_i^N) , (c_j, t_i^N) , and (t_i^N, s_{k2}) . For the labeling, we set $l(z_i) = z_i$, $l(z_i') = z_i'$, $l(n_i) = n_i$, $l(n_i') = n_i'$, and $l(t_i^Z) = l(t_i^N) = \tau$.
3. Let the initial marking put just one token on s_1 , and let \emptyset be the set of final markings of $net(C)$.
4. Let every unprimed transition label of $net(C)$ (other than τ) be an input action, and let every primed transition label of $net(C)$ be an output action.

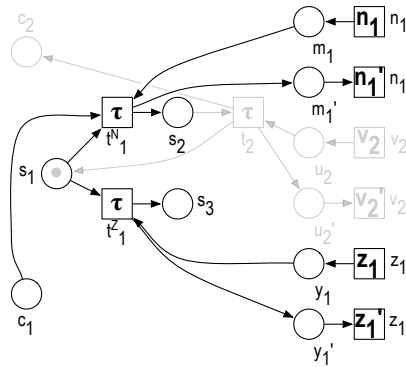
Adding a *dc*-pattern (*dc* for “definitely cheating”) to $net(C)$ for $i \in BS(C)$ means adding a τ -labeled transition t_i^C (a *dc*-transition) and arcs (y_i, t_i^C) , (t_i^C, y_i') , (s_i, t_i^C) , (t_i^C, s_{k1}) , (c_j, t_i^C) , (t_i^C, c_j) . (Note that t_i^C is a copy of t_i^Z with additional arcs to/from c_j .)

For the 2-counter machine *ADD* from Alg. 1, Fig. 49a depicts the basic net $net(ADD)$. The first command CMD_1 of *ADD* is of type 2; its pattern consists of the transitions $t_1^N, t_1^Z, n_1, n_1', z_1, z_1'$ and we highlighted it in Fig. 49b. The second command CMD_2 of *ADD* is of type 1; its pattern consists of the transitions t_2, v_2, v_2' and we highlighted it in Fig. 49c. The counters c_1 and c_2 are modeled by the places c_1 and c_2 , and the current state

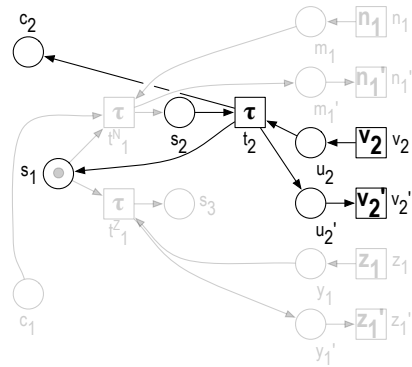
of ADD is modeled by marking one of the places s_1, s_2, s_3 . The input actions of $net(ADD)$ are $\{n_1, z_1, v_2\}$, and the output actions are $\{n'_1, z'_1, v'_2\}$.



(a) Labeled net $net(ADD)$



(b) Highlighted pattern for CMD_1



(c) Highlighted pattern for CMD_2

Figure 49: The basic net $net(ADD)$ of the 2-counter machine ADD from Alg. 1 with the highlighted patterns for CMD_1 (which is of type 2) and CMD_2 (which is of type 1).

For any counter machine C with counters c_1, \dots, c_m and for any input values x_1, \dots, x_m , we can “simulate” C with $net(C)$ by adding x_j tokens to the initial marking of place c_j ($1 \leq j \leq m$). However, it is possible to “cheat” in the pattern of type 2 (see Fig. 48e): By cheating we mean that transition t_i^Z fires although the respective place c_j is not empty. Also note that firing a dc -transition has the same effect as firing the respective transition t_i^Z transition in terms of the state of C ; that is, shifting a token from place s_j to place s_{k_1} .

The construction of $net(C)$ applies to any m -counter machine C in the following, because already for two counters the halting problem is undecidable [178].

Theorem 79 [halting problem [178]]

It is undecidable whether a given 2-counter machine halts on given inputs.

We proceed by reducing conformance to the halting problem of 2-counter machines.

4.3.2 Conformance is undecidable

The following lemma relates the halting problem of 2-counter machines to the inclusion of the sets of *stop*-traces of two constructed labeled nets. We follow the proof strategy from [126]: For a 2-counter machine C and given input values x_1 and x_2 , we construct two labeled nets N_1 and N_2 which are modifications of $\text{net}(C)$ simulating C . The construction of N_1 and N_2 ensures that the only way to exhibit the noninclusion is to simulate C without cheating and to terminate—which is possible if and only if C halts for x_1 and x_2 .

Lemma 80 [halting problem vs. *stop*-inclusion]

Let C be a 2-counter machine and $x_1, x_2 \in \mathbb{N}$. We can construct two action-equivalent labeled nets N_1 and N_2 (as modifications of $\text{net}(C)$) such that the following conditions are equivalent:

1. C does not halt for the given inputs x_1 and x_2 .
2. N_1 and N_2 are bisimilar.
3. $\text{stop}(N_1) \subseteq \text{stop}(N_2)$.

Proof. We construct N_1 and N_2 from $\text{net}(C)$ and the input values x_1 and x_2 in four steps:

1. Take $\text{net}(C)$ and extend its initial marking by x_1 tokens in c_1 and x_2 tokens in c_2 .
2. Add places p, p', o, e , transitions $t_p, t_{p'}, q, t_e, f$, and arcs $(p, t_p), (t_p, p), (p', t_{p'}), (t_{p'}, p'), (t_p, o), (t_{p'}, o), (o, q), (p, t_e), (s_n, t_e), (t_e, e),$ and (e, f) . Label the transitions $t_p, t_{p'}$, and t_e with τ , transition q with the output action q , and transition f with the output action f . Figure 50a sketches step one and step two for ADD with inputs $x_1 := 1$ and $x_2 := 1$, and Fig. 50b highlights the difference to the labeled net $\text{net}(\text{ADD})$ from Fig. 49a.
3. For each branching state $i \in \text{BS}(C)$ that checks counter c_j , add two *dc*-patterns: the τ -labeled transitions $t_i^C, t_i^{C'}$, and the arcs $(s_i, t_i^C), (s_i, t_i^{C'}), (t_i^C, s_{k_1}), (t_i^{C'}, s_{k_1}), (y_i, t_i^C), (y_i, t_i^{C'}), (t_i^C, y_i), (t_i^{C'}, y_i)$ (i.e., detecting cheating on the zero-branch), $(c_j, t_i^C), (t_i^C, c_j), (c_j, t_i^{C'}), (t_i^{C'}, c_j)$ (i.e., cheating means c_j is not empty), and $(p, t_i^C), (t_i^C, p'), (p', t_i^{C'}), (t_i^{C'}, p)$ (i.e., detecting cheating means switching the token between p and p'). Figure 51a sketches this step for ADD and $\text{BS}(\text{ADD}) = \{1\}$. We highlighted the first *dc*-pattern in Fig. 51b. The second *dc*-pattern is identical to the first *dc*-pattern except that it consumes a token from p' and produces a token on p .
4. Take two copies of the arising net. In one copy, put one token in p yielding the labeled net N_1 . In the other, put one token in p' yielding the labeled net N_2 . Figure 52a and Fig. 52b indicate this for ADD , if we ignore the dashed frame.

In every reachable marking, the places p and p' together hold at most one token. As long as any of the places p, p' , and o is marked, the corresponding marking is not a stop except for inputs: The transition q is labeled with an

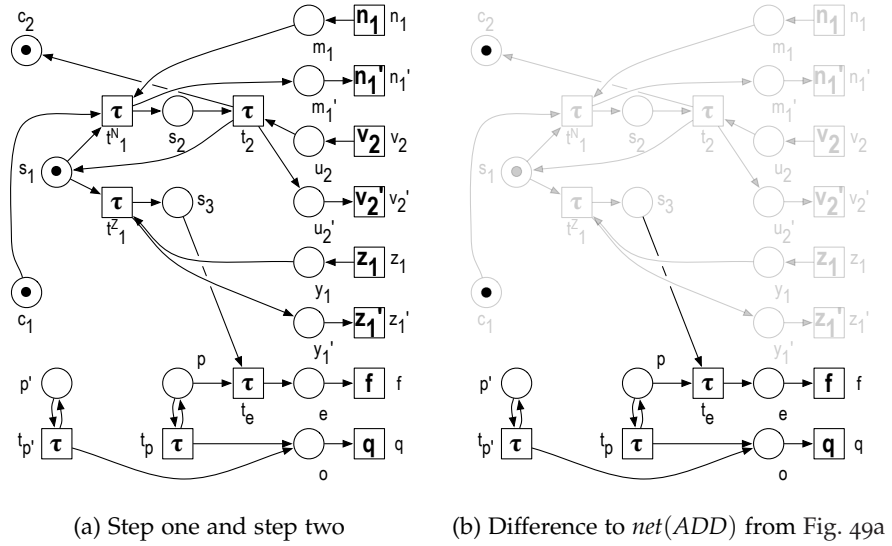


Figure 50: Step one and step two of the auxiliary constructions for Lem. 80 and the 2-counter machine ADD .

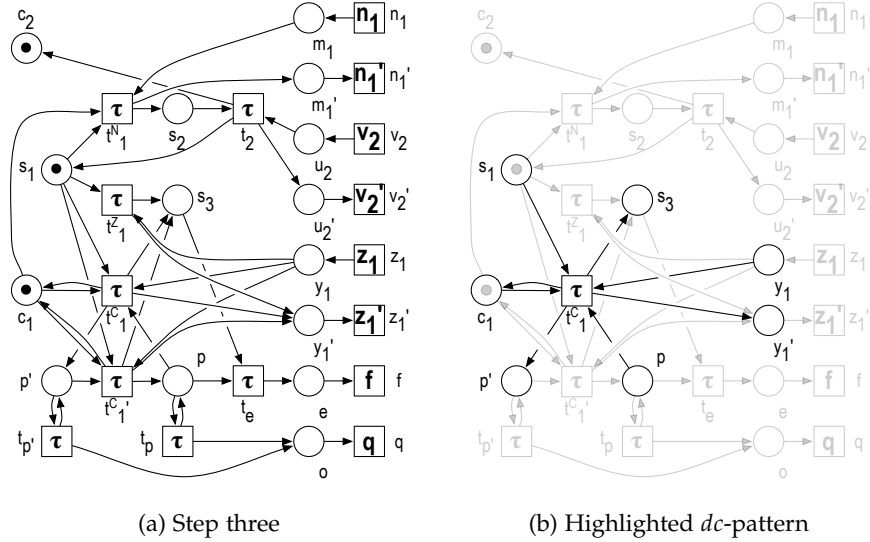


Figure 51: Step three of the auxiliary constructions for Lem. 80 and the 2-counter machine ADD .

output action and may fire. Thus, the only way to reach a stop except for inputs is to empty the place o by firing q , and to have one token on p and fire t_e and f .

(1) implies (2): Assume C does not halt for inputs x_1 and x_2 . Let D be the set of all pairs (m, m') of equal markings m of N_1 and N_2 . Let M be the set of all pairs (m_1, m_2) such that m_1 and m_2 are reachable by the same correct run in N_1 and N_2 , respectively. A run is correct if it simulates C without cheating—that is, no dc -transition fires, and transition t_i^Z (for $i \in BS(C)$) fires only if the respective place c_j is empty. We show that $D \uplus M$ is a bisimulation; thus, N_1 and N_2 are bisimilar as $(m_{N_1}, m_{N_2}) \in M$ by the construction of N_1 and N_2 .

So consider a pair $(m_1, m_2) \in M$. As m_1 and m_2 is reached by the same correct run σ in N_1 and N_2 , respectively, m_1 and m_2 differ only in the places p

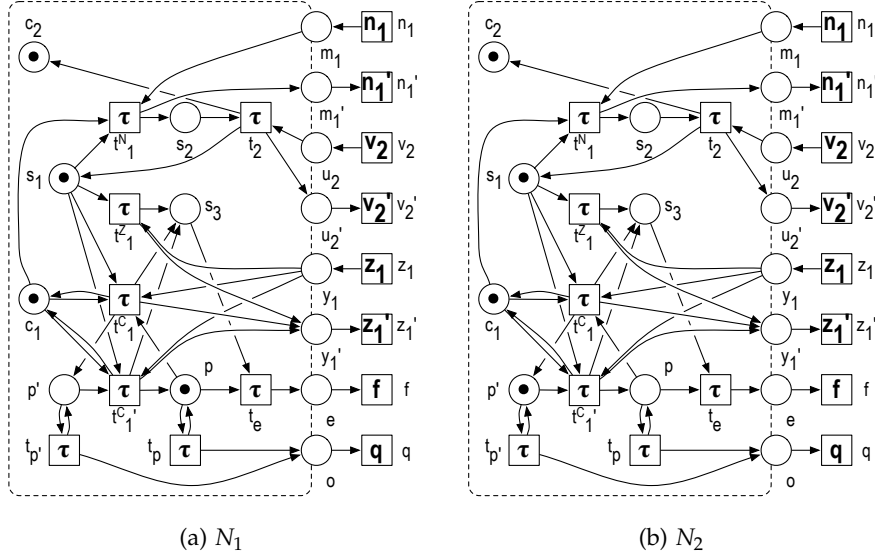


Figure 52: The labeled nets N_1 and N_2 (ignoring the dashed frame) for Lem. 80 and the open nets $open(N_1)$ and $open(N_2)$ (ignoring all transitions outside the dashed frame) for Thm. 81 and the 2-counter machine ADD from Alg. 1.

and p' , w.l.o.g., we have $m_1(p) = 1$, $m_1(p') = 0$, and $m_2(p) = 0$, $m_2(p') = 1$. Thus, every transition, except t_e and the dc -transitions, is enabled at m_1 in N_1 if and only if is enabled at m_2 in N_2 . Transition t_e is never enabled, because σ is a correct run, and C does not halt by assumption (i.e., place s_n is never marked). We distinguish two cases:

1. The firing of any transition besides t_i^Z , t_i^C , and $t_i^{C'}$ (for $i \in BS(C)$) at m_1 in N_1 can be simulated by the firing of the same transition at m_2 in N_2 , and vice versa. The respective firings lead again to a marking pair in M .
2. If cheating is possible in N_1 at m_1 and N_1 fires t_i^Z , t_i^C , or $t_i^{C'}$ with $i \in BS(C)$ when the respective place c_j is not empty, then one transition out of the set $\{t_i^Z, t_i^C, t_i^{C'}\}$ can fire in N_2 such that both nets have the same marking m (and thus $(m, m) \in D$) afterward. In detail: If $m_1 \xrightarrow{t_i^Z} m$ in N_1 , then $m_2 \xrightarrow{t_i^{C'}} m$ in N_2 ; if $m_1 \xrightarrow{t_i^C} m$ in N_1 , then $m_2 \xrightarrow{t_i^Z} m$ in N_2 . The same argument applies if cheating is possible in N_2 : If $m_2 \xrightarrow{t_i^Z} m$ in N_2 , then $m_1 \xrightarrow{t_i^C} m$ in N_1 ; if $m_2 \xrightarrow{t_i^{C'}} m$ in N_2 , then $m_1 \xrightarrow{t_i^Z} m$ in N_1 .

If N_1 and N_2 have the same marking (i.e., we have a pair in D), then each can simulate the other by firing the same transition, remaining in D . Thus, $D \uplus M$ is a bisimulation.

(2) implies (3): trivial

(3) implies (1): By contraposition, assume C halts for inputs x_1 and x_2 . Then, we construct a run $m_{N_1} \xrightarrow{\sigma} m$ in N_1 such that σ simulates C correctly (i.e., without cheating) and $m(s_n) = 1$ (i.e., C reaches the $HALT$ command): For each command CMD_i that C performs, we add three transitions to σ . If $i \notin BS(C)$, we add $v_i t_i v_i'$ to σ . If $i \in BS(C)$, we add $z_i t_i^Z z_i'$ (if the respective counter is zero) or $n_i t_i^N n_i'$ (otherwise) to σ . Now the trace w corresponding to the run $\sigma t_e f$ is a $stop$ -trace of N_1 , i.e., $w \in stop(N_1)$.

To perform the same trace in N_2 , there is no choice but to perform the same run σ (except for possibly firing t_p or t'_p in-between): For example, to perform action v_i one has to fire transition v_i , and to perform action v'_i then one has to fire transitions $t_i v'_i$. Observe that one cannot fire $t_i^C z'_i$ or $t_i'^C z'_i$ to perform action z'_i because the firing of t_i^Z is correct at this stage and, thus, the respective counter (and the corresponding place) is empty. However, after σ the transition t_e is not enabled in N_2 , because p is not marked. Thus, $w \notin L(N_2)$, which implies $w \notin \text{stop}(N_2)$. \square

The respective set of final markings of the two labeled nets that we constructed in the proof of Lem. 80 is empty. In the following theorem—the main result of this section—we reduce conformance to the halting problem of a 2-counter machine with Lem. 80, thereby exploiting that the sets of *stop*- and *dead*-traces coincide for labeled nets with an empty set of final markings.

Theorem 81 [undecidability of conformance]

For two interface-equivalent open nets Impl and Spec , $\text{Impl} \sqsubseteq_{\text{conf}} \text{Spec}$ is undecidable.

Proof. Let C be a 2-counter machine with input values x_1 and x_2 . We construct two interface-equivalent open nets $\text{open}(N_1)$ and $\text{open}(N_2)$ from the labeled nets N_1 and N_2 from Lem. 80 by removing all transitions t that are not τ -labeled, and interpreting t 's preset (postset) as output (input) place. Figure 52a and Fig. 52b illustrate $\text{open}(N_1)$ and $\text{open}(N_2)$ for ADD , if we ignore all transitions outside the dashed frame and the adjacent arcs. Clearly, $\text{stop}(\text{open}(N_1)) = \text{stop}(N_1)$ and $\text{stop}(\text{open}(N_2)) = \text{stop}(N_2)$. As $\text{open}(N_1)$ and $\text{open}(N_2)$ have the empty set of final markings, we have $\text{stop}(\text{open}(N_1)) = \text{dead}(\text{open}(N_1))$ and $\text{stop}(\text{open}(N_2)) = \text{dead}(\text{open}(N_2))$. Now assume that conformance is decidable. Then $\text{open}(N_1)$ conforms to $\text{open}(N_2)$ if and only if $\text{stop}(\text{open}(N_1)) \subseteq \text{stop}(\text{open}(N_2))$ by Thm. 61 if and only if C does not halt for the given inputs x_1 and x_2 by Lem. 80. Thus, we can decide the halting problem for 2-counter machines, which is a contradiction to Thm. 79. Therefore, conformance is undecidable. \square

4.3.3 Compositional conformance is undecidable

In this section, we show that also the coarsest precongruence that is contained in the conformance relation (i.e., compositional conformance) is undecidable. Here, it is essential that compositional conformance can be characterized using a modification $\mathcal{F}_{\text{fin}}^+$ of the \mathcal{F}^+ -semantics [246, 217], as shown in Thm. 75. With this, it is not difficult to prove the following lemma based on the construction of two labeled nets in the proof of Lem. 80.

Lemma 82 [halting problem vs. compositional conformance]

Let C be a 2-counter machine and $x_1, x_2 \in \mathbb{N}$. We can construct two action-equivalent labeled nets N_1 and N_2 (as modifications of $\text{net}(C)$) such that

$$C \text{ does not halt for the given inputs } x_1 \text{ and } x_2 \quad \text{iff} \quad N_1 \sqsubseteq_{\text{conf}}^c N_2.$$

Proof. We construct the labeled nets N_1 and N_2 as in the proof of Lem. 80. \Rightarrow : N_1 and N_2 have no final markings by Lem. 80. Thus, for $i \in \{1, 2\}$ and any set $Y \subseteq \Sigma_i^*$, $(w, X, Y) \in \mathcal{F}_{\text{fin}}^+(N_i)$ if and only if $(w, X, \emptyset) \in \mathcal{F}_{\text{fin}}^+(N_i)$.

In other words, the Y set of any fintree failure of N_i is arbitrary. As N_1 and N_2 are bisimilar, we have $(w, X, \emptyset) \in \mathcal{F}_{fin}^+(N_1)$ if and only if $(w, X, \emptyset) \in \mathcal{F}_{fin}^+(N_2)$. As the Y sets are arbitrary, we conclude that $N_1 \sqsubseteq_{\mathcal{F}_{fin}^+} N_2$ by Def. 69 and $N_1 \sqsubseteq_{conf}^c N_2$ by Thm. 75.

\Leftarrow : Let $w \in L(N_1)$. Then $(w, \emptyset, \emptyset) \in \mathcal{F}_{fin}^+(N_1)$ by Def. 63, thus $(w, \emptyset, \emptyset) \in \mathcal{F}_{fin}^+(N_2)$ by assumption, Thm. 75, and Def. 69. Therefore, $w \in L(N_2)$ by Def. 63. If C halts for the inputs x_1 and x_2 , then $L(N_1) \not\subseteq L(N_2)$ as shown in the proof of Lem. 80. By contraposition, we conclude that C does not halt for the inputs x_1 and x_2 . \square

With Lem. 82, we immediately conclude the undecidability of compositional conformance from Thm. 79 with an argument as in the proof of Thm. 81.

Theorem 83 [undecidability of compositional conformance]

For two interface-equivalent open nets $Impl$ and $Spec$, $Impl \sqsubseteq_{conf}^c Spec$ is undecidable.

4.4 CONCLUSIONS

In Sect. 3.1, we formalized responsiveness as a fundamental behavioral correctness criterion for open nets. Responsiveness induces a preorder based on partner inclusion—that is, the conformance relation.

We provided open nets with the *stopdead*-semantics, which is a weak version of the semantics with the same name in [228], and showed that set-wise inclusion of the *stopdead*-semantics characterizes conformance. In addition, we detailed that compositional conformance cannot be characterized with the *stopdead*-semantics or, in general, a denotational semantics weaker than standard failures semantics. Therefore, we provided open nets with the \mathcal{F}_{fin}^+ -semantics, which is an extension of Vogler’s \mathcal{F}^+ -semantics [246]. Refinement on the \mathcal{F}_{fin}^+ -semantics characterizes compositional conformance. Based on the characterizations of conformance and compositional conformance, we showed that both relations are undecidable. Our proofs worked by reduction to the halting problem of 2-counter machines using a variation of the “Jančar-Patterns” [126].

This chapter is based on results published in [192, 248, 249].

IN the previous chapter, we characterized conformance and analyzed it for compositionality and decidability. It turned out that conformance is not compositional. Thus, we also characterized the coarsest precongruence that is contained in the conformance relation—that is, compositional conformance. We showed that both conformance and compositional conformance are undecidable and, thus, not applicable for the verification of open systems. As conformance and compositional conformance turn out to be undecidable, we further explore the notion of *b*-responsiveness that we already introduced in Sect. 3.2: We require the composition of two open nets to be responsive and, additionally, to be *b*-bounded, where *b* denotes a bound (see Conv. 3). The relation *b*-conformance is the conformance relation that corresponds to *b*-responsiveness. In this chapter, we give a fine-grained analysis of *b*-conformance. Table 2 illustrates how this chapter fits into the structure of Part II, if we leave out Chap. 7.

relation	characterization	compositionality	decidability
conformance	Chap. 4	Chap. 4	Chap. 4
<i>b</i> -conformance	Chap. 5	Chap. 6	Chap. 5 & Chap. 6

Table 2: The structure of Part II without Chap. 7. We highlight the current chapter with a gray background.

The highlighted part of Fig. 53 illustrates how we analyze *b*-conformance. To this end, we provide a denotational semantics for open nets, thereby extending the *stopdead*-semantics from Chap. 4 for *b*-conformance to the *b*-coverable *stopdead*-semantics. Then, we show that a refinement relation based on the *b*-coverable *stopdead*-semantics coincides with *b*-conformance. Based on that characterization of *b*-conformance, we show that *b*-conformance is decidable: We represent the trace sets of the *b*-coverable *stopdead*-semantics by an LTS. A bisimulation relation between two LTS then decides *b*-conformance. In addition, we develop a finite characterization of all *b*-partners and of all *b*-conforming open nets for a given open net. These finite characterizations also serve as alternative decision procedures for *b*-responsiveness and for *b*-conformance.

This chapter is structured as follows: We characterize *b*-conformance in Sect. 5.1 and provide a decision procedure in Sect. 5.2. We elaborate an alternative decision procedure in Sect. 5.3 and show how to characterize all *b*-conforming open nets. We implemented both decision procedures and present the implementation in Sect. 5.4. Section 5.5 concludes this chapter.

5.1 CHARACTERIZING *b*-CONFORMANCE

In this section, we characterize the *b*-conformance relation between two interface-equivalent open nets *Impl* and *Spec*. To this end, we provide each

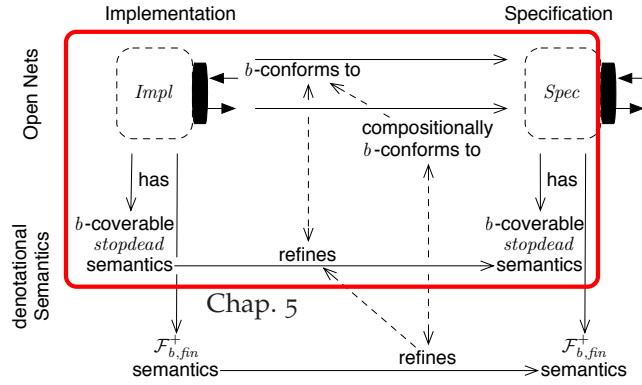


Figure 53: Characterizing b -conformance using a denotational semantics for open nets. A solid arc illustrates the relation described by the corresponding arc label. Dashed arcs illustrate logical implication or logical equivalence, depending on their number of heads.

open net with a trace-based semantics—this time, four sets of traces. Inclusion of the four sets of traces of $Impl$ in the four sets of traces of $Spec$ defines a refinement relation that coincides with b -conformance. In other words, we provide a trace-based characterization of b -conformance.

5.1.1 The b -bounded stopdead-semantics for open nets

Our trace-based semantics for b -responsiveness of an open net N extends the *stopdead*-semantics of Def. 55 by information about possible bound violations of N . A bound violation is a marking that is not b -bounded, and we investigate the traces leading to such a bound violation, called *strict bound_b-violators*. A bound violation is regarded as catastrophic because it cannot be corrected. Thus, the behavior after a bound violation does not matter, and we will hide all possible differences by treating all *strict bound_b-violators* and their continuations in the same way. Technically, we achieve the hiding by including all continuations of *strict bound_b-violators* in a set *bound_b*, the set of *bound_b-violators*. For the same reason, *bound_b* is contained in the other three components of our b -bounded *stopdead*-semantics: The language of N , and the sets of *stop*-traces and *dead*-traces from the *stopdead*-semantics in Def. 55. This technique is called *flooding* in [103].

Definition 84 [b -bounded stopdead-semantics]

Let N be a labeled net. A trace w is a *strict bound_b-violation* of N if there exists a marking m with $m_N \xrightarrow{w} m$ that is not b -bounded. Every continuation of a *strict bound_b-violation* is a *bound_b-violation* of N . The b -bounded *stopdead*-semantics of N is defined by the following four sets of traces

- $bound_b(N) = \{w \in (I \uplus O)^* \mid w \text{ is a bound}_b\text{-violation of } N\}$,
- $L_b(N) = L(N) \cup bound_b(N)$,
- $stop_b(N) = stop(N) \cup bound_b(N)$, and
- $dead_b(N) = dead(N) \cup bound_b(N)$.

Example 85 As a running example for this chapter, consider again the database D and its user U from Sect. 3.2. For convenience, we depict them again in Fig. 54. Observe that after firing *shutdown* in $env(D)$, transitions *process* and *retrieve* are never enabled while there may be still pending tokens on the places q^i and d^o . In addition, the transitions *shutdown* and *forward* may fire at most once in $env(D)$. Therefore, the language of D is

$$\begin{aligned} L(D) = & \{w \in \{s, q, d\}^* \mid \forall v \sqsubseteq w : |v|_d \leq |v|_q\} \\ & \cup \{w f z \mid w, z \in \{s, q, d\}^* \wedge \forall v \sqsubseteq w : |v|_d \leq |v|_q \\ & \quad \wedge |w|_s > 0 \wedge |z|_d \leq |w|_q - |w|_d\}. \end{aligned}$$

To respect bound 1, after producing a first token on q^i (s^i), $env(D)$ may produce a “second” token on q^i (s^i) only after the firing of transition *process* (*shutdown*); that is, the first token on q^i (s^i) is consumed and q^i (s^i) is empty again. Otherwise, the place q^i (s^i) may hold two tokens, which violates bound 1. The $bound_1$ -violators of D are

$$\begin{aligned} bound_1(D) = & \uparrow \{w \in L(D) \mid \exists v \sqsubseteq w : |v|_d + 1 < |v|_q\} \\ & \cup \uparrow \{w \in L(D) \mid \exists v \sqsubseteq w : |v|_f + 1 < |v|_s\}. \end{aligned}$$

Every *stop*-trace of D either contains an f or does not contain an s and the number of d 's equals the number of q 's; more precisely,

$$\begin{aligned} stop(D) = & \{w \in \{q, d\}^* \mid \forall v \sqsubseteq w : |v|_d \leq |v|_q \wedge |w|_d = |w|_q\} \\ & \cup \{w f z \mid w, z \in \{s, q, d\}^* \wedge \forall v \sqsubseteq w : |v|_d \leq |v|_q \\ & \quad \wedge |w|_s > 0 \wedge |z|_d \leq |w|_q - |w|_d\}. \end{aligned}$$

As $[p_0]$ is the only final marking of D , we have $dead(D) = stop(D)$.

The language of U is

$$L(U) = \{w \in \{q, d, f\}^* \mid \forall v \sqsubseteq w : |v|_d + 1 \geq |v|_q\}.$$

Observe that we can only consume a token from the place d^i after producing a token on the place q^o in $env(U)$. In addition, the places q^o , d^i and f^i are unbounded in $env(U)$, and a token on f^i cannot be removed by any transition. To violate bound 1, it suffices to produce more tokens on d^i than has been consumed from q^o ; for example, even the trace d of $env(U)$ leads to the marking $[p_3, q^o, q^o]$ that violates bound 1. Therefore, the $bound_1$ -violators of U are

$$\begin{aligned} bound_1(U) = & \uparrow \{w \in L(U) \mid \exists v \sqsubseteq w : |v|_d > |v|_q\} \\ & \cup \uparrow \{w \in L(U) \mid |w|_f > 1\}. \end{aligned}$$

The markings $[p_3]$ and $[p_3, f^i]$ are the only stops except for inputs of $env(U)$ that are reachable without violating bound 1. Therefore, in every *stop*-trace of U , the number of q 's equals the number of d 's plus 1 (because transition *query* is enabled at the initial marking of $env(U)$); more precisely,

$$stop(U) = \{w \in \{q, d, f\}^* \mid \forall v \sqsubseteq w : |v|_d + 1 \geq |v|_q \wedge |w|_d + 1 = |w|_q\}.$$

Because the only final marking $[\]$ is not reachable in $env(U)$, we have $dead(U) = stop(U)$. \diamond

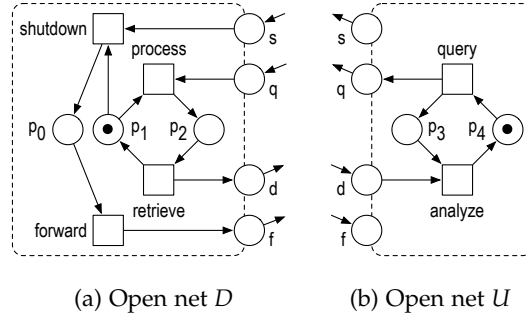


Figure 54: The open nets D and U from Sect. 3.2. In addition to the figures, we have $\Omega_D = \{[p_0]\}$ and $\Omega_U = \{[]\}$.

The set of $bound_b$ -violators and the flooded language L_b is already part of the b -bounded *stopdead*-semantics for deadlock freedom [227, 228]. Therefore, we recall how these sets are calculated for a composition of two labeled nets and a composition of two open nets.

Proposition 86 [$bound_b$ and L_b of composition]

For two composable labeled nets N_1 and N_2 , we have

1. $bound_b(N_1 \parallel N_2) = \uparrow (bound_b(N_1) \parallel L_b(N_2)) \cup \uparrow (L_b(N_1) \parallel bound_b(N_2))$,
2. $L_b(N_1 \parallel N_2) = (L_b(N_1) \parallel L_b(N_2)) \cup bound_b(N_1 \parallel N_2)$, and

for two composable open nets N_1 and N_2 , we have

3. $bound_b(N_1 \oplus N_2) = \uparrow (bound_b(N_1) \uparrow L_b(N_2)) \cup \uparrow (L_b(N_1) \uparrow bound_b(N_2))$,
4. $L_b(N_1 \oplus N_2) = (L_b(N_1) \uparrow L_b(N_2)) \cup bound_b(N_1 \oplus N_2)$.

Proof. The first equation has already been proved for $b = 1$ in [246, Theorem 3.3.3]; we can use the same considerations to show that this result can be generalized to an arbitrary bound $b \in \mathbb{N}^+$. The second equation follows directly from the first equation and [246, Theorem 3.1.7(4)]. The third and the fourth equation have already been proved in [228, Theorem 30]. \square

The trace ε is the only trace of a closed net by Def. 17. Thus, the question whether ε is a $bound_b$ -violator is equal to the question whether the closed net is b -bounded. In other words, a closed net N is b -bounded if and only if $bound_b(N) = \emptyset$; otherwise—that is, N is not b -bounded—we have $bound_b(N) = \{\varepsilon\}$. Therefore, we directly conclude the following corollary from Prop. 86.

Corollary 87 [b -boundedness vs. $bound_b$ and L_b intersection]

For two composable open nets N_1 and N_2 such that $N_1 \oplus N_2$ is a closed net, we have

$$N_1 \oplus N_2 \text{ is } b\text{-bounded} \quad \text{iff} \quad \begin{aligned} bound_b(N_1) \cap L_b(N_2) &= \emptyset \text{ and} \\ L_b(N_1) \cap bound_b(N_2) &= \emptyset. \end{aligned}$$

An open net C is a b -partner of an open net N if and only if C is a partner of N and $N \oplus C$ is b -bounded. As we already remarked before Cor. 87, the composition $N \oplus C$ of two composable open nets N and C is b -bounded if and only if $\text{bound}_b(N \oplus C) = \emptyset$. Thus, we combine Cor. 87 with Prop. 59 for the following characterization of b -responsiveness.

Proposition 88 [b -responsiveness vs. b -bounded *stopdead*-semantics]

For two composable open nets N_1 and N_2 such that $N_1 \oplus N_2$ is a closed net, we have

$$\begin{aligned} N_1 \text{ and } N_2 \text{ are } b\text{-responsive} \quad \text{iff} \quad & \text{bound}_b(N_1) \cap L_b(N_2) = \emptyset \text{ and} \\ & L_b(N_1) \cap \text{bound}_b(N_2) = \emptyset \text{ and} \\ & \text{stop}_b(N_1) \cap \text{dead}_b(N_2) = \emptyset \text{ and} \\ & \text{dead}_b(N_1) \cap \text{stop}_b(N_2) = \emptyset. \end{aligned}$$

Proof. \Rightarrow : $N_1 \oplus N_2$ is b -bounded by Def. 41, thus $\text{bound}_b(N_1 \oplus N_2) = \emptyset$ by Def. 84. Therefore, $\text{bound}_b(N_1) \cap L_b(N_2) = \emptyset$ and $L_b(N_1) \cap \text{bound}_b(N_2) = \emptyset$ by Cor. 87. In addition, $N_1 \oplus N_2$ is responsive by Def. 41, thus $\text{stop}(N_1) \cap \text{dead}(N_2) = \emptyset$ and $\text{dead}(N_1) \cap \text{stop}(N_2) = \emptyset$ by Prop. 59. We already have $\text{bound}_b(N_1) \cap \text{bound}_b(N_2) = \emptyset$ by the first intersection of the right-hand side, thus $\text{stop}_b(N_1) \cap \text{dead}_b(N_2) = \emptyset$ and $\text{dead}_b(N_1) \cap \text{stop}_b(N_2) = \emptyset$ by Def. 84.

\Leftarrow : $N_1 \oplus N_2$ is b -bounded by the first two equations and Cor. 87 and responsive by the last two equations and Prop. 59. Thus, N_1 and N_2 are b -responsive by Def. 41. \square

Example 89 Consider again the open net D in Fig. 54a and the open net U in Fig. 54b. By Ex. 85, we have $\text{bound}_1(D) \cap L_1(U) = \emptyset$, $L_1(D) \cap \text{bound}_1(U) = \emptyset$, $\text{stop}_1(D) \cap \text{dead}_1(U) = \emptyset$, and $\text{dead}_1(D) \cap \text{stop}_1(U) = \emptyset$. Thus, U is a 1-partner (and, therefore, a b -partner for all $b \in \mathbb{N}^+$) of D by Prop. 88, which we already claimed in Ex. 45.

Now consider the second database user U' from Sect. 3.2, which we depict again in Fig. 55. The open net U' is a modification of U —that is, transition *quit* has been added. Observe that we can only consume a token from the place d^i after producing a token on the place q^o in $\text{env}(U')$. Therefore, the language of U' is

$$\begin{aligned} L(U') = & \{w \in \{q, d, f\}^* \mid \forall v \sqsubseteq w : |v|_d + 1 \geq |v|_q\} \\ & \cup \{wsz \mid w, z \in \{q, d, f\}^* \wedge \forall v \sqsubseteq w : |v|_d + 1 \geq |v|_q \\ & \wedge |w|_d \geq |w|_q \wedge |z|_q \leq |w|_d - |w|_q\}. \end{aligned}$$

The places q^o , d^i , and f^i are unbounded in $\text{env}(U')$ and a token on the place f^i cannot be removed by any transition. Like for the open net U , it suffices to produce more tokens on d^i than has been consumed from q^o or more than one token on f^i to violate the bound 1. Therefore, the bound_1 -violators of U' are

$$\begin{aligned} \text{bound}_1(U') = & \uparrow \{w \in L(U') \mid \exists v \sqsubseteq w : |v|_d > |v|_q\} \\ & \cup \uparrow \{w \in L(U') \mid |w|_f > 1\}. \end{aligned}$$

In every *stop*-trace of U' , there exists an s , or the number of d 's is smaller than the number of q 's because of transition *query*; more precisely,

$$\begin{aligned} \text{stop}(U') = & \{w \in \{q, d, f\}^* \mid \forall v \sqsubseteq w : |v|_d + 1 \geq |v|_q \wedge |w|_d + 1 = |w|_q\} \\ & \cup \{wsz \mid w, z \in \{q, d, f\}^* \wedge \forall v \sqsubseteq w : |v|_d + 1 \geq |v|_q \\ & \wedge |w|_d \geq |w|_q \wedge |z|_q \leq |w|_d - |w|_q\}. \end{aligned}$$

The only final marking $[\]$ of $\text{env}(U)$ is reachable. Therefore, the set $\text{dead}(U')$ is a strict subset of $\text{stop}(U')$, and we have

$$\begin{aligned} \text{dead}(U') = & \{w \in \{q, d, f\}^* \mid \forall v \sqsubseteq w : |v|_d + 1 \geq |v|_q \wedge |w|_d + 1 = |w|_q\} \\ & \cup \{wsz \mid w, z \in \{q, d, f\}^* \wedge \forall v \sqsubseteq w : |v|_d + 1 \geq |v|_q \\ & \wedge |w|_d \geq |w|_q \wedge |z|_q \leq |w|_d - |w|_q \\ & \wedge |wz|_d > |wz|_q\}. \end{aligned}$$

Comparing the b -bounded *stopdead*-semantics of U' and of D (see Ex. 85), we find a trace $sf \in \text{stop}(U') \cap \text{dead}(D)$, which implies $sf \in \text{stop}_b(U') \cap \text{dead}_b(D)$ for any bound b . Thus, U' is not a b -partner of D by Prop. 88, which justifies our claim in Ex. 45. \diamond

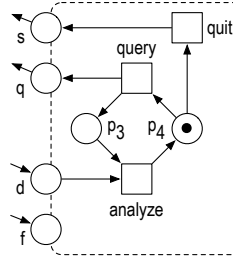


Figure 55: The second database user U' from Sect. 3.2. We have $\Omega_{U'} = \{[\]\}$.

Similar to the inclusion of the *stopdead*-semantics, inclusion of the b -bounded *stopdead*-semantics defines a refinement relation. However, inclusion of the *stopdead*-semantics coincides with conformance (see Thm. 61), but inclusion of the b -bounded *stopdead*-semantics gives only a necessary but not a sufficient condition for b -conformance.

Theorem 90 [b -bounded *stopdead*-inclusion implies b -conformance]

For two interface-equivalent open nets Impl and Spec , we have

$$\begin{aligned} \text{bound}_b(\text{Impl}) \subseteq \text{bound}_b(\text{Spec}) \quad \text{implies} \quad \text{Impl} \sqsubseteq_{b, \text{conf}} \text{Spec}. \\ \text{and } L_b(\text{Impl}) \subseteq L_b(\text{Spec}) \\ \text{and } \text{stop}_b(\text{Impl}) \subseteq \text{stop}_b(\text{Spec}) \\ \text{and } \text{dead}_b(\text{Impl}) \subseteq \text{dead}_b(\text{Spec}) \end{aligned}$$

Proof. Proof by contraposition. Consider an open net C such that $\text{Impl} \oplus C$ and, equivalently, $\text{Spec} \oplus C$ are closed nets. Otherwise, C is neither a b -partner of Impl nor of Spec . Assume that C is not a b -partner of Impl . Then, Impl and C are not b -responsive by Def. 44, and we have either $\text{bound}_b(\text{Impl}) \cap L_b(C) \neq \emptyset$, $L_b(\text{Impl}) \cap \text{bound}_b(C) \neq \emptyset$, $\text{stop}_b(\text{Impl}) \cap \text{dead}_b(C) \neq \emptyset$, or $\text{dead}_b(\text{Impl}) \cap \text{stop}_b(C) \neq \emptyset$ by Prop. 88. Because of the assumed inclusions, we have either $\text{bound}_b(\text{Spec}) \cap L_b(C) \neq \emptyset$, $L_b(\text{Spec}) \cap \text{bound}_b(C) \neq \emptyset$,

$stop_b(Spec) \cap dead_b(C) \neq \emptyset$, or $dead_b(Spec) \cap stop_b(C) \neq \emptyset$. Again with Prop. 88, we see that $Spec$ and C are not b -responsive; that is, C is not a b -partner of $Spec$ and thus $Impl \sqsubseteq_{b, conf} Spec$. \square

The converse of Thm. 90 does not hold in general, as we illustrate with the following two examples.

Example 91 Consider again the patched database D' from Sect. 3.2, which we depict again in Fig. 56. Observe that no reachable marking of $env(D')$ marks place f^o . The language of D' is

$$L(D') = \{w \in \{s, q, d\}^* \mid \forall v \sqsubseteq w : |v|_d \leq |v|_q\}.$$

Like D in Fig. 54a, the environment of D' respects bound 1 if, after producing a first token on q^i , $env(D')$ produces the next token on q^i only if there exists a token on d^o ; that is, the first token on q^i was already consumed and q^i is empty again. However, in contrast to D , transition s may fire at most once in $env(D')$ in order to respect the bound 1; otherwise, the place s^i may hold two tokens. Therefore, the $bound_1$ -violators of D' are

$$\begin{aligned} bound_1(D') = & \uparrow \{w \in L(D') \mid \exists v \sqsubseteq w : |v|_d + 1 < |v|_q\} \\ & \cup \uparrow \{w \in L(D') \mid \exists v \sqsubseteq w : |v|_s > 1\}. \end{aligned}$$

Every $stop$ -trace of D' either contains an s and reaches the marking $[\]$, or the number of d 's equals the number of q 's; more precisely,

$$\begin{aligned} stop(D') = & \{w \in \{q, d\}^* \mid \forall v \sqsubseteq w : |v|_d \leq |v|_q \wedge |w|_d = |w|_q\} \\ & \cup \{wsz \mid w, z \in \{s, q, d\}^* \wedge \forall v \sqsubseteq wz : |v|_d \leq |v|_q\}. \end{aligned}$$

As $[\]$ is the only final marking of D' , we have

$$\begin{aligned} dead(D') = & \{w \in \{q, d\}^* \mid \forall v \sqsubseteq w : |v|_d \leq |v|_q \wedge |w|_d = |w|_q\} \\ & \cup \{wsz \mid w, z \in \{s, q, d\}^* \wedge \forall v \sqsubseteq wz : |v|_d \leq |v|_q \\ & \quad \wedge (|wz|_s > 0 \vee |wz|_d < |wz|_q)\}. \end{aligned}$$

Now consider the open net D in Fig. 54a. We already claimed in Ex. 49 that D' b -conforms to D . However, comparing their respective b -bounded $stopdead$ -semantics from the previous paragraph and Ex. 85, we find a trace $s \in stop(D')$ but $s \notin stop(D) \cup bound_1(D)$ (and, thus, $s \notin stop_b(D)$). We cannot verify this claim with Thm. 90, but, intuitively, the trace s does not “destroy” b -conformance of D' and D because no b -partner of D would send a message s , leading to a part of D which cannot be reliably used by any b -partner. \diamond

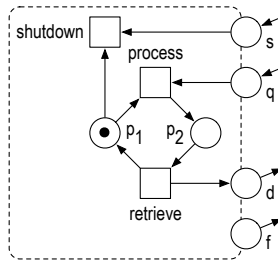


Figure 56: The patched database server D' from Sect. 3.2. We have $\Omega_{D'} = \{[\]\}$.

Example 92 For a second example that the converse of Thm. 90 does not hold, consider again the open nets D and D' in Fig. 54a and Fig. 56, this time, with the empty sets of final markings. No b -partner of D' will send an s because this would eventually lead to firing of *shutdown*, yielding a nonfinal and nonresponsive marking where neither p_1 nor p_2 contains a token. Thus, we conclude that D b -conforms to D' (remember that this only holds because we changed their sets of final markings). However, there exists a trace sf with $sf \in L(D) \setminus L(D')$ by Ex. 85 and Ex. 91, because $env(D')$ cannot fire the transition f at all. Hence, $L_b(D) \not\subseteq L_b(D')$. \diamond

5.1.2 The b -coverable stopdead-semantics for open nets

The cause of the counterexamples in Ex. 91 and Ex. 92 and, thus, the reason why the converse of Thm. 90 does not hold is that b -conformance ignores those parts of open nets *Impl* and *Spec* that cannot be used reliably—that is, those markings that cannot be covered in the composition with any b -partner. In contrast, standard trace-based semantics (like the *stopdead*-semantics in Def. 55 and the b -bounded *stopdead*-semantics in Def. 84) compare the two open nets as a whole.

That standard language inclusion can be too strict has been observed for a stronger criterion than b -responsiveness in [162, 43, 181]. Mooij et al. [181] propose two solutions to overcome this problem. The first idea is to restrict the class of open nets considered to those where every place and transition can be covered. The second idea is to encode the covering nature of b -conformance in the trace-based semantics. In the following, we work out the latter idea in the present setting:

We aim to encode the covering nature of b -conformance in the b -bounded *stopdead*-semantics. To achieve this, we introduce a set that captures all b -uncoverable traces—that is, traces w that cannot be executed by (the environment of) any b -partner of an open net N , regardless whether w can be executed in $env(N)$ or not.

Replacing in every trace set of the b -bounded *stopdead*-semantics of an open net N the set of $bound_b$ -violators by the set of b -uncoverable traces yields the b -coverable *stopdead*-semantics of N . This semantics differs from the trace-based semantics in Def. 55 and Def. 84, as the b -uncoverable traces are an *external* characterization—they quantify over all b -partners of N . The latter does not cause a problem, because we can still calculate this set, as we will show in Sect. 5.2.

Definition 93 [b -coverable *stopdead*-semantics]

Let N be an open net. A word $w \in (I \uplus O)^*$ is a b -uncoverable trace of N if there does not exist a b -partner C of N with $w \in L_b(C)$. The b -coverable *stopdead*-semantics of N is defined by the sets of traces

- $uncov_b(N) = \{w \in (I \uplus O)^* \mid w \text{ is a } b\text{-uncoverable trace of } N\}$,
- $uL_b(N) = L(N) \cup uncov_b(N)$,
- $ustop_b(N) = stop(N) \cup uncov_b(N)$, and
- $udead_b(N) = dead(N) \cup uncov_b(N)$.

Example 94 Consider again the open nets D and D' in Fig. 54a and Fig. 56. No b -partner of D has a trace of D that contains an s in its language; otherwise, D may reach the nonfinal and nonresponsive marking $[\]$. In addition, the number of q 's a b -partner C of D sends to D never exceeds the number of d 's already received from D plus b : For example, consider a bound of 1 and the initial state of $env(D)$. If C sends the first q , then $env(D)$ may still remain in the marking $[p_1, q^1]$ without firing *process*; if C sends a second q (i.e., violates the mentioned condition because the number of d 's is 0 and $2 > 0 + 1$), then q^i contains two tokens and bound 1 is violated. Therefore, we have

$$\begin{aligned} uncov_b(D) = & \uparrow \{w \in L(D) \mid |w|_s > 0\} \\ & \cup \uparrow \{w \in L(D) \mid \exists v \sqsubseteq w : |v|_q > |v|_d + b\}. \end{aligned}$$

We argued in Ex. 49 that D' b -conforms to D , and observed in Ex. 91 that $s \in stop_b(D')$ but $s \notin stop_b(D)$; that is, $stop_b$ -inclusion fails. By the above considerations, we have $s \in uncov_b(D)$. Thus, s is in the flooded $stop$ -set of the b -coverable $stopdead$ -semantics of D , i.e., $s \in ustop_b(D)$. \diamond

By Prop. 88, a $bound_b$ -violation of an open net is a b -uncoverable trace of N . So we directly conclude that the set of $bound_b$ -violators of N is contained in the set of b -uncoverable traces of N . As a result, the b -coverable $stopdead$ -semantics extends the b -bounded $stopdead$ -semantics by flooding more traces: $bound_b$ -violators and b -uncoverable traces.

Corollary 95

For an open net N , we have

- $bound_b(N) \subseteq uncov_b(N)$,
- $L_b(N) \subseteq uL_b(N)$,
- $stop_b(N) \subseteq ustop_b(N)$, and
- $dead_b(N) \subseteq udead_b(N)$.

Inclusion of the b -uncoverable traces, the flooded language, the flooded $stop$ -traces, and the flooded $dead$ -traces defines a refinement relation. We show that an open net $Impl$ b -conforms to an open net $Spec$ if and only if the respective traces of $Impl$'s b -coverable $stopdead$ -semantics are included in the respective traces of $Spec$'s b -coverable $stop$ -semantics. For the proof, we use the following lemma. The first item of Lem. 96 states that for every trace w , which is neither a trace nor a b -uncoverable trace of an open net N , there exists a b -partner of N containing w in its set of $bound_b$ -violators. The second item of Lem. 96 states that for every trace w of N , which is neither a $stop$ -trace nor b -uncoverable, there exists a b -partner of N containing w in its set of $dead$ -traces. The third item of Lem. 96 is similar to the second item, but with the $stop$ - and $dead$ -traces exchanged. It states that for every trace w of N , which is neither a $dead$ -trace nor b -uncoverable, there exists a b -partner of N containing w in its set of $stop$ -traces.

Lemma 96

Let N be an open net. Then

1. If $w \notin uL_b(N)$, then there exists a b -partner C of N with $w \in \text{bound}_b(C)$.
2. If $w \in L(N) \setminus \text{ustop}_b(N)$, then there exists a b -partner C of N with $w \in \text{dead}(C)$.
3. If $w \in L(N) \setminus \text{udead}_b(N)$, then there exists a b -partner C of N with $w \in \text{stop}(C)$.

Proof. (1) Let $w \notin uL_b(N)$ with $w = w_1 \dots w_k$ for $j = 1, \dots, k$, $w_j \in I_N \uplus O_N$. As $w \notin \text{uncov}_b(N)$, there exists a b -partner C of N with $w \in L_b(C)$ by Def. 93. If $w \notin \text{bound}_b(C)$, we construct from w and C a b -partner $N_w^{\text{bound}} \oplus C'$ of N with bound_b -violation w as follows: In a first step, we construct an open net N_w that basically shifts all tokens from N to (a copy of) C , and vice versa. Moreover, N_w tracks whether a firing sequence in C is (a “permutation” of) a prefix of w , and subsequently moving a token in N_w from an initially marked place p_0 to a place p_k . Intuitively, a token on a place p_j means we already encountered (a “permutation” of) the trace $w_1 \dots w_j$. For shifting, we introduce several transitions in N_w for each interface place in N . In a second step, we extend N_w : If and only if the place p_k is marked—that is, we encountered (a “permutation” of) the trace $w_1 \dots w_k = w$ —a “disaster” transition t_{disaster} is enabled, which may put an unlimited number of tokens onto an inner place p_{disaster} . The latter construction yields the open net N_w^{bound} . This construction guarantees that w is a bound_b -violation of $N_w^{\text{bound}} \oplus C'$.

Let $I' = \{i' \mid i \in I_C\}$ and $O' = \{o' \mid o \in O_C\}$ be “fresh” copies of I_C and O_C . We derive the open net $C' = (P_C, T_C, F'_C, m_C, I', O', \Omega_C)$ from C by renaming the interface of C and adjusting the flow relation accordingly. We define the open net $N_w = (P', T', F', m_{N_w}, \Omega_{N_w}, O_N \uplus O_{C'}, I_N \uplus I_{C'})$ with

- $P' = \{p_i \mid 0 \leq i \leq k\} \uplus \{p_{\text{err}}\}$,
- $T' = \{t_i^x \mid 1 \leq i \leq k \wedge x \in O_N \uplus I_N\} \uplus \{t_{\text{err}}^x \mid x \in O_N \uplus I_N\} \uplus \{t_{\text{err}}\}$,
- $F' = \{(x, t_i^x), (t_i^x, x'), (p_{i-1}, t_i^x) \mid 1 \leq i \leq k \wedge x \in O_N\} \uplus \{(x', t_i^x), (t_i^x, x), (p_{i-1}, t_i^x) \mid 1 \leq i \leq k \wedge x \in I_N\} \uplus \{(t_i^x, p_i) \mid 1 \leq i \leq k \wedge x \in O_N \uplus I_N \wedge x = w_i\} \uplus \{(t_i^x, p_{\text{err}}) \mid 1 \leq i \leq k \wedge x \in O_N \uplus I_N \wedge x \neq w_i\} \uplus \{(x, t_{\text{err}}^x), (t_{\text{err}}^x, x'), (p_{\text{err}}, t_{\text{err}}^x), (t_{\text{err}}^x, p_{\text{err}}) \mid x \in O_N\} \uplus \{(x', t_{\text{err}}^x), (t_{\text{err}}^x, x), (p_{\text{err}}, t_{\text{err}}^x), (t_{\text{err}}^x, p_{\text{err}}) \mid x \in I_N\} \uplus \{(p_k, t_{\text{err}}), (t_{\text{err}}, p_{\text{err}})\}$,
- $m_{N_w} = [p_0]$, and
- $\Omega_{N_w} = \{[p] \mid p \in P'\}$.

Figure 57 illustrates the construction of N_w . We have $L(C) = L(N_w \oplus C')$, $\text{bound}_b(C) = \text{bound}_b(N_w \oplus C')$, $\text{stop}(C) = \text{stop}(N_w \oplus C')$, and $\text{dead}(C) = \text{dead}(N_w \oplus C')$. Therefore, the open net $N_w \oplus C'$ is, as C , a b -partner of N by Prop. 88.

The places $p_0, \dots, p_k, p_{\text{err}}$ together always carry one token. Let m be an arbitrary marking of $\text{env}(N_w \oplus C')$ where p_k is marked and all interface places

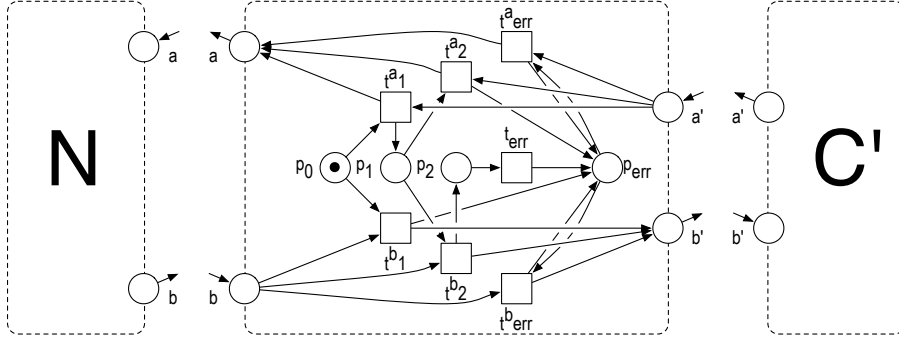


Figure 57: Illustration of the construction of the open net N_w for N with $I_N = \{a\}$, $O_N = \{b\}$, and $w = ab \notin uL_b(N)$.

of $N_w \oplus C'$ are empty. The key observation is that if a trace v of $env(N_w \oplus C')$ leads to m , then $v \notin L_b(N)$: By the choice of m and the construction of N_w , the Parikh vectors of v and w agree and we can transform w into v by moving $w_j \in O_{N_w \oplus C'} = I_N$ right and $w_j \in I_{N_w \oplus C'} = O_N$ left, like in the proof of Thm. 61. Because $w \notin L_b(N)$ by assumption, moving $w_j \in I_N$ right always results in a trace $v \notin L_b(N)$: If $env(N)$ cannot fire a run underlying w , then $env(N)$ cannot fire a run with an even later provision of a token on the input place w_j^i . Similarly, moving $w_j \in O_N$ left always results in a trace $v \notin L_b(N)$: If $env(N)$ cannot fire a run underlying w , then $env(N)$ cannot fire a run where a token on the output place w_j^o is needed earlier.

As a generalization, we now show that $v \notin L_b(N)$ for any trace v with $m_{env(N_w \oplus C')} \xrightarrow{v} m$ and $m(p_k) > 0$, performing an induction on the number of tokens that m puts on the interface places of $N_w \oplus C'$. The previous observation is the base case. If a place x^i is marked at m with x being an input place of $N_w \oplus C'$, then a token on x^i was not needed to perform v ; thus, we can move the last x in v to the end, obtaining another trace v' of $env(N_w \oplus C')$. Now $v' = v''x$ has the prefix $v'' \notin L_b(N)$ by induction. Thus, $v' \notin L_b(N)$ and moving $x \in O_N$ left results in $v \notin L_b(N)$ as argued previously. If an output place x^o is marked at m with x being an output place of $N_w \oplus C'$, then also vx is a trace of $env(N_w \oplus C')$ and $vx \notin L_b(N)$ by induction, thus, $v \notin L_b(N)$ because $x \in I_N$ can always fire in $env(N)$.

Next, we extend N_w to an open net N_w^{bound} by adding a *disaster* transition $t_{disaster}$. The transition $t_{disaster}$ may put an unlimited number of tokens onto an inner place $p_{disaster}$ of N_w^{bound} if and only if the place p_k is marked. Formally, we define the open net $N_w^{bound} = (P' \uplus \{p_{disaster}\}, T' \uplus \{t_{disaster}\}, F' \uplus F'', m_{N_w}, \Omega_{N_w}, O_N \uplus O_{C'}, I_N \uplus I_{C'})$ with $F'' = \{(p_k, t_{disaster}), (t_{disaster}, p_k), (t_{disaster}, p_{disaster})\}$. By construction of $N_w^{bound} \oplus C'$, a run underlying a newly introduced strict $bound_b$ -violator v of $N_w^{bound} \oplus C'$ (i.e., $v \in bound_b(N_w^{bound} \oplus C') \setminus bound_b(N_w \oplus C')$) marks place p_k . Thus, by the observation from the last paragraph, we have $v \notin L_b(N) \supseteq stop_b(N) \supseteq dead_b(N) \supseteq bound_b(N)$. Clearly, $N_w^{bound} \oplus C'$ is still a b -partner of N because of Prop. 88.

(2) Let $w \in L(N) \setminus ustop_b(N)$ with $w = w_1 \dots w_k$ for $j = 1, \dots, k$, $w_j \in I_N \uplus O_N$. As $w \notin uncov_b(N)$, there exists a b -partner C of N with $w \in L_b(C)$ by Def. 93. Then $w \in L(C) \setminus bound_b(C)$; otherwise, C is not a b -partner of N by Prop. 88. As w is no $stop$ -trace of N , there is always some output that $m_{env(N)}$ can perform after w . For each such output $o \in O_N$, we conclude that $wo \in L(N)$ and $wo \in L(C)$, thus, wo is not a (strict) $bound_b$ -violator of N by Prop. 88.

Like in the proof of (1), we construct a b -partner $N_w \oplus C'$ of N with *dead-trace* w : We track whether a firing sequence in C is (a “permutation” of) a prefix of w by composing C with another open net N_w , and subsequently moving a token in N_w from an initially marked place p_0 to a place p_k . Later, if and only if the place p_k is marked—that is, we encountered (a “permutation” of) the trace $w_1 \dots w_k = w$ —we prevent any output from N_w , but allow input to N_w . We put $\Omega_{N_w} = \{[p] \mid p \in P' \setminus \{p_k\}\}$. As $[p_k]$ is not a final marking of N_w , w will be a *dead-trace*. Once N_w receives one input—some $o \in O_N$ as discussed above—we make N_w transparent again.

Let $I' = \{i' \mid i \in I_C\}$ and $O' = \{o' \mid o \in O_C\}$ be “fresh” copies of I_C and O_C . We derive the open net $C' = (P_C, T_C, F'_C, m_C, I', O', \Omega_C)$ from C by renaming the interface of C and adjusting the flow relation accordingly. We define the open net $N_w = (P', T', F', m_{N_w}, \Omega_{N_w}, O_N \uplus O_{C'}, I_N \uplus I_{C'})$ with

- $P' = \{p_i \mid 0 \leq i \leq k\} \uplus \{p_{err}\}$,
- $T' = \{t_i^x \mid 1 \leq i \leq k+1 \wedge x \in O_N\} \uplus \{t_i^x \mid 1 \leq i \leq k \wedge x \in I_N\} \uplus \{t_{err}^x \mid x \in O_N \uplus I_N\}$,
- $F' = \{(x, t_i^x), (t_i^x, x'), (p_{i-1}, t_i^x) \mid 1 \leq i \leq k+1 \wedge x \in O_N\} \uplus \{(x', t_i^x), (t_i^x, x), (p_{i-1}, t_i^x) \mid 1 \leq i \leq k \wedge x \in I_N\} \uplus \{(t_i^x, p_i) \mid 1 \leq i \leq k \wedge x \in O_N \uplus I_N \wedge x = w_i\} \uplus \{(t_i^x, p_{err}) \mid 1 \leq i \leq k \wedge x \in O_N \uplus I_N \wedge x \neq w_i\} \uplus \{(t_{k+1}^x, p_{err}) \mid x \in O_N\} \uplus \{(x, t_{err}^x), (t_{err}^x, x'), (p_{err}, t_{err}^x), (t_{err}^x, p_{err}) \mid x \in O_N\} \uplus \{(x', t_{err}^x), (t_{err}^x, x), (p_{err}, t_{err}^x), (t_{err}^x, p_{err}) \mid x \in I_N\}$,
- $m_{N_w} = [p_0]$, and
- $\Omega_{N_w} = \{[p] \mid p \in P' \setminus \{p_k\}\}$.

Figure 58 illustrates the construction of N_w .

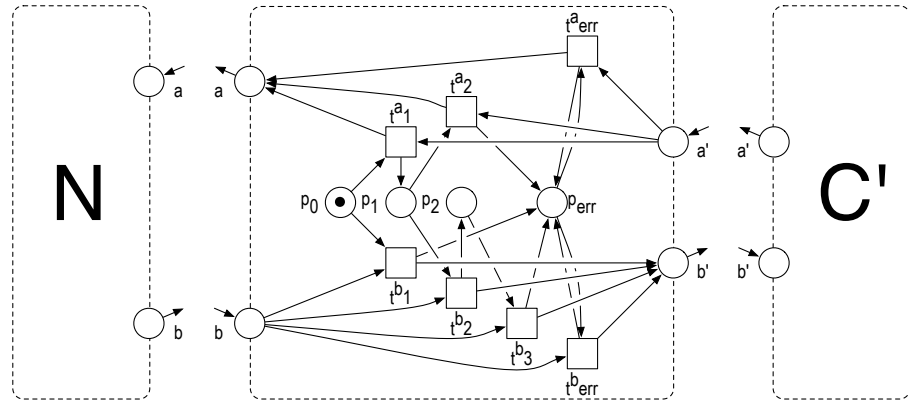


Figure 58: Illustration of the construction of the open net N_w for N with $I_N = \{a\}$, $O_N = \{b\}$, and $w = ab \in L(N) \setminus \text{ustop}_b(N)$.

The places p_0, \dots, p_k, p_{err} together always carry one token. Let m be an arbitrary marking of $\text{env}(N_w \oplus C')$ where p_k is marked and all interface places of $N_w \oplus C'$ are empty. The key observation is that if a trace v of $\text{env}(N_w \oplus C')$ leads to m , then $v \notin \text{stop}_b(N)$: By the choice of m and the construction of

N_w , the Parikh vectors of v and w agree and we can transform w into v by moving $w_j \in O_{N_w \oplus C'} = I_N$ right and $w_j \in I_{N_w \oplus C'} = O_N$ left, like in the proof of (1). Thus, a marking reached by v in $env(N)$ can also be reached by w , and we conclude that $v \notin stop_b(N)$.

By the construction of $N_w \oplus C'$, a run underlying a newly introduced *dead*- or *stop*-trace v of $N_w \oplus C'$ (i.e., $v \in dead_b(N_w \oplus C') \setminus dead_b(C)$ or $v \in stop_b(N_w \oplus C') \setminus stop_b(C)$) marks place p_k and leaves no token on an interface place of $N_w \oplus C'$. Thus, by the observation above, we have $v \notin stop_b(N)$ and, hence, $v \notin dead_b(N)$. In addition, we have $L(N_w \oplus C') \subseteq L(C)$ and $bound_b(N_w \oplus C') \subseteq bound_b(C)$ by the construction of $N_w \oplus C'$. Therefore, the open net $N_w \oplus C'$ is, as C , a b -partner of N by Prop. 88.

(3) This is analogous to the proof of (2), except that we have to add several places and transitions to N_w , yielding the open net N'_w : The introduced *dead*-trace w of $N_w \oplus C'$ may be a *stop*-trace of N , i.e., $w \in stop(N) \setminus dead(N)$. Thus, after N'_w recognizes (a “permutation” of) w , we buffer all messages from C' to N'_w inside N'_w : For each input place $x \in O_{C'}$ of N_w , we add a place \bar{x} and a transition $t_{\bar{x}}$ such that $\bullet t_{\bar{x}} = \{x\}$, $t_{\bar{x}}^\bullet = \{\bar{x}\}$, $x^\bullet = \{t_{\bar{x}}\}$, and the postset of \bar{x} in N'_w is the postset of x in N_w . In addition, we define all markings of N'_w as final markings. That way, we have $w \in stop_b(N'_w \oplus C')$ while guaranteeing $dead_b(N'_w \oplus C') \subseteq dead_b(C)$. \square

5.1.3 Refinement on the b -coverable *stopdead*-semantics

With Lem. 96, we can show that b -conformance coincides with the refinement relation defined by inclusion of the b -coverable *stopdead*-semantics.

Theorem 97 [b-conformance and b-coverable *stopdead*-inclusion coincide]

For two interface-equivalent open nets $Impl$ and $Spec$, we have

$$\begin{aligned} Impl \sqsubseteq_{b, conf} Spec \quad \text{iff} \quad & uncov_b(Impl) \subseteq uncov_b(Spec) \text{ and} \\ & uL_b(Impl) \subseteq uL_b(Spec) \text{ and} \\ & ustop_b(Impl) \subseteq ustop_b(Spec) \text{ and} \\ & udead_b(Impl) \subseteq udead_b(Spec) . \end{aligned}$$

Proof. \Rightarrow : Let $w \notin uncov_b(Spec)$; that is, there exists a b -partner C of $Spec$ with $w \in L_b(C)$ by Def. 93. Clearly, C is a b -partner of $Impl$ by $Impl \sqsubseteq_{b, conf} Spec$, thus $w \notin uncov_b(Impl)$. This proves $uncov_b(Impl) \subseteq uncov_b(Spec)$.

Let $w \in uL_b(Impl) \setminus uncov_b(Impl)$ and assume $w \notin uL_b(Spec)$. There exists a b -partner C of $Spec$ with $w \in bound_b(C)$ by Lem. 96(1). Clearly, C is not a b -partner of $Impl$ by Prop. 88, and we have a contradiction to our assumption that $Impl \sqsubseteq_{b, conf} Spec$. Thus, $w \in uL_b(Spec)$.

Let $w \in ustop_b(Impl) \setminus uncov_b(Impl)$ and assume $w \notin ustop_b(Spec)$. Then, $w \in stop(Impl) \subseteq L(Impl)$ and $w \in L(Spec)$, as $uL_b(Impl) \subseteq uL_b(Spec)$ has been shown already. We can construct a b -partner C of $Spec$ with $w \in dead(C)$ by Lem. 96(2). Clearly, C is not a b -partner of $Impl$ by Prop. 88, and we have a contradiction to our assumption that $Impl \sqsubseteq_{b, conf} Spec$. Thus, $w \in ustop_b(Spec)$.

Let $w \in udead_b(Impl) \setminus uncov_b(Impl)$ and assume $w \notin udead_b(Spec)$. Then, $w \in dead(Impl) \subseteq L(Impl)$ and $w \in L(Spec)$, as $uL_b(Impl) \subseteq uL_b(Spec)$ has been shown already. We can construct a b -partner C of $Spec$ with $w \in stop(C)$ by Lem. 96(3). Clearly, C is not a b -partner of $Impl$ by Prop. 88, and we

have a contradiction to our assumption that $Impl \sqsubseteq_{b, conf} Spec$. Thus, $w \in udead_b(Spec)$.

\Leftarrow : Proof by contraposition. Assume that the four inclusions hold and that C is not a b -partner of $Impl$. We show that C is not a b -partner of $Spec$ either.

$Impl$ and C are not b -responsive by Def. 44, and we have $bound_b(Impl) \cap L_b(C) \neq \emptyset$, $L_b(Impl) \cap bound_b(C) \neq \emptyset$, $stop_b(Impl) \cap dead_b(C) \neq \emptyset$, or $dead_b(Impl) \cap stop_b(C) \neq \emptyset$ by Prop. 88. We consider each case separately:

- $w \in bound_b(Impl) \cap L_b(C)$: Then $w \in bound_b(Impl) \subseteq uncov_b(Impl) \subseteq uncov_b(Spec)$ by Cor. 95 and assumption, so C is not a b -partner of $Spec$ by Def. 93.
- $w \in L_b(Impl) \cap bound_b(C)$: Then $w \in L_b(Impl) \subseteq uL_b(Impl) \subseteq uL_b(Spec)$ by Cor. 95 and assumption. If $w \in L(Spec)$ then C is not a b -partner of $Spec$ by $w \in bound_b(C)$ and Prop. 88; otherwise, if $w \in uncov_b(Spec)$, then C is not a b -partner of $Spec$ by $w \in bound_b(C) \subseteq L_b(C)$ and Def. 93.
- $w \in stop_b(Impl) \cap dead_b(C)$: Then, $w \in ustop_b(Impl) \subseteq ustop_b(Spec)$ by Cor. 95 and assumption. If $w \in stop(Spec)$ then C is not a b -partner of $Spec$ by $w \in dead_b(C)$ and Prop. 88; otherwise, if $w \in uncov_b(Spec)$, then C is not a b -partner of $Spec$ by $w \in dead_b(C) \subseteq L_b(C)$ and Def. 93.
- $w \in dead_b(Impl) \cap stop_b(C)$: Then, $w \in udead_b(Impl) \subseteq udead_b(Spec)$ by Cor. 95 and assumption. If $w \in dead(Spec)$ then C is not a b -partner of $Spec$ by Prop. 88; otherwise, if $w \in uncov_b(Spec)$, then C is not a b -partner of $Spec$ by $w \in stop_b(C) \subseteq L_b(C)$ and Def. 93. \square

Example 98 Consider again the open nets D and D' in Fig. 54a and Fig. 56. We claimed in Ex. 49 that D' b -conforms to D , but observed in Ex. 91 that $s \in stop_b(D')$ but $s \notin stop_b(D)$; that is, $stop_b$ -inclusion fails. However, Ex. 94 shows that $s \in uncov_b(D) \subseteq ustop_b(D)$. Thus, the difference between D and D' is hidden in the b -coverable $stopdead$ -semantics due to flooding: $s \in ustop_b(D) \subseteq ustop_b(D')$. Therefore, with Thm. 97, we can finally show that D' b -conforms to D . \diamond

Despite the external characterization of the trace set $uncov_b$, we can compute the b -coverable $stopdead$ -semantics of an open net N ; that is, we can decide whether two interface-equivalent open nets are b -conforming. We elaborate the corresponding decision procedure in the next section.

5.2 DECIDING b -CONFORMANCE

To decide b -conformance, we have to check four language inclusions according to Thm. 97. To this end, we represent each language by a finite LTS. To ease the four inclusion checks, we subsume the four LTSs into one deterministic, τ -free, and finite LTS CSD_b with different state labels: Each state label of CSD_b represents a partition of the language Σ^* , and each of the four languages in Thm. 97 is a distinct union of some of these partitions. Now, one inclusion check on CSD_b coincides with the four language inclusion checks on the LTSs that represent the languages. In other words, the constructed LTS CSD_b represents exactly the four languages of the b -coverable $stopdead$ -semantics. Technically, we check inclusion on CSD_b by computing a bisimulation and additionally check the related state labels. Figure 59 illustrates the decision procedure for b -conformance.

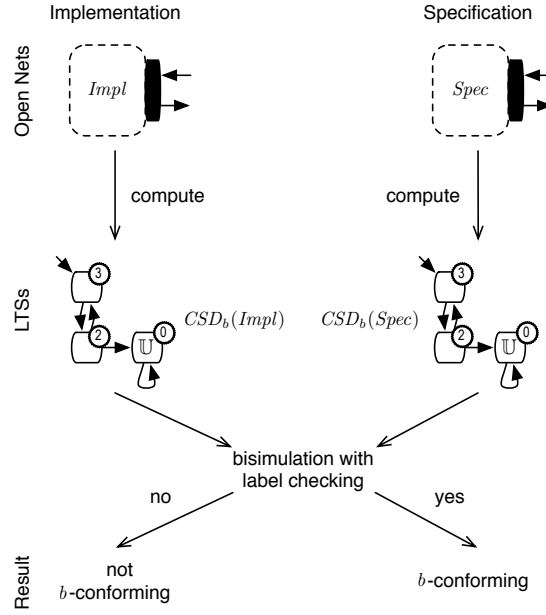


Figure 59: Using CSD_b to decide if an open net $Impl$ b -conforms to an interface-equivalent open net $Spec$.

For convenience, we demonstrate this technique in Sect. 5.2.1 on an easier case: We construct an LTS BSD_b that represents the four languages of the b -bounded *stopdead*-semantics in Def. 84. We prove the correctness of our construction and show that deciding the emptiness of language intersection on BSD_b coincides with the decision of b -responsiveness by Prop. 88. In Sect. 5.2.2, we then construct the previously mentioned LTS CSD_b from BSD_b . We prove that CSD_b represents the b -coverable *stopdead*-semantics of Def. 93 and show how to decide b -conformance on CSD_b using Thm. 117.

5.2.1 Deciding b -responsiveness using the LTS BSD_b

In the following, we construct an LTS $BSD_b(N)$ that characterizes the b -bounded *stopdead*-semantics of an open net N . By Def. 84, the b -bounded *stopdead*-semantics of N consists of the four languages $bound_b(N)$, $dead_b(N)$, $stop_b(N)$, and $L_b(N)$. When mentioned in this order, each language is a subset of its successor. Figure 60 illustrates these subset relations as an Euler diagram if we ignore the annotations to the right-hand side.

Although the b -bounded *stopdead*-semantics of N consists of four languages, we shall construct the LTS $BSD_b(N)$ with five state labels 0–4. The resulting languages $L_0(BSD_b(N))$ – $L_4(BSD_b(N))$ partition the set of all finite words Σ^* according to the right-hand side of Fig. 60; more precisely, we define $BSD_b(N)$ such that

- $L_4(BSD_b(N)) = \Sigma^* \setminus L_b(N)$,
- $L_3(BSD_b(N)) = L_b(N) \setminus stop_b(N)$,
- $L_2(BSD_b(N)) = stop_b(N) \setminus dead_b(N)$,
- $L_1(BSD_b(N)) = dead_b(N) \setminus bound_b(N)$, and
- $L_0(BSD_b(N)) = bound_b(N)$.

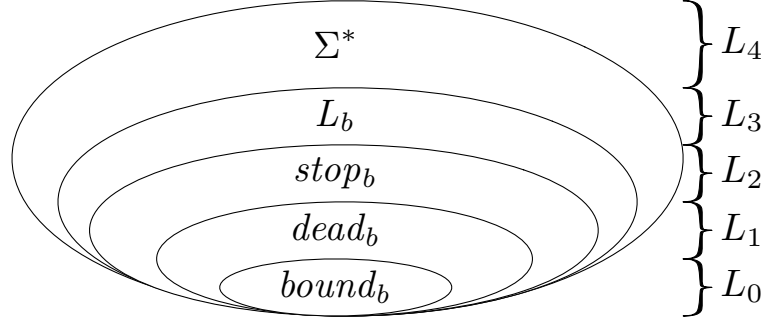


Figure 60: An illustration of the languages of the b -bounded *stopdead*-semantics from Def. 84 and their representation as languages of the LTS BSD_b from Def. 99.

Clearly, the four languages of N 's b -bounded *stopdead*-semantics can be easily derived from the five languages $L_0(BSD_b(N))$ – $L_4(BSD_b(N))$. Nevertheless, partitioning Σ^* into $L_0(BSD_b(N))$ – $L_4(BSD_b(N))$ eases the correctness proof of our construction.

As a starting point for $BSD_b(N)$, we construct the reachability graph of the labeled net $env(N)$, but stop the construction whenever we reach a marking m that violates bound b . That way, we generate only finitely many markings of $env(N)$ and guarantee finiteness of the constructed LTS. The initial state of our constructed LTS is the initial marking $m_{env(N)}$. Each state m that violates bound b gets the state label 0 and a loop for each transition label in Σ . Every trace w of $env(N)$ reaching m is a strict $bound_b$ -violation of N . The added loops include every suffix of w into the language $L_0(BSD_b(N))$ of the constructed LTS, which, intuitively, coincides with the inclusion of all continuations of a strict $bound_b$ -violation into the language $bound_b(N)$ by Def. 84. We now label every unlabeled state m of the constructed LTS (i.e., every b -bounded marking m of $env(N)$) with a label 1–3, indicating whether m is dead except for inputs (label 1), a stop but not dead except for inputs (label 2), or not a stop except for inputs (label 3) in $env(N)$. Intuitively, this yields the languages $L_1(BSD_b(N))$ – $L_3(BSD_b(N))$ we already mentioned above.

The idea behind a language inclusion check in automata theory [224, 123] is to make the automaton of the larger language deterministic (by constructing the powerset automaton) and then construct the unique simulation relation between the two automata. Consequently, we make the LTS constructed above deterministic using a variant of the powerset construction that also removes all τ -transitions (i.e., ε -transitions in automata theory). The resulting LTS is τ -free, deterministic, and finite; we refer to it as $BSD_b(N)$. Each state Q of $BSD_b(N)$ is a set of states of the nondeterministic LTS (i.e., a set of markings of $env(N)$). We label Q with the *minimum* label of its contained states. In addition, each state Q of $BSD_b(N)$ has an outgoing transition $Q \xrightarrow{x} Q'$ for each label in $x \in \Sigma$; if the corresponding trace was not present in the nondeterministic LTS, then Q' is an artificial state Q_\emptyset . The state Q_\emptyset corresponds to the empty set of states of the nondeterministic LTS, and we label Q_\emptyset with the label 4. As a final simplification, we identify all states of $BSD_b(N)$ with the label 0 into one error state \mathbb{U} .

Definition 99 [labeled transition system BSD_b]

Let N be an open net. We define the LTS $BSD_b(N) = (\mathcal{Q}, \delta, Q_{BSD_b(N)}, \Sigma^{in}, \Sigma^{out}, \lambda)$ with

- $\mathcal{Q} = \{\mathbb{U}\} \uplus \mathcal{Q}'$ with $\mathcal{Q}' \subseteq \mathcal{P}(M_{env(N)})$ and $\mathbb{U} \notin \mathcal{P}(M_{env(N)})$,
- $\delta = \{ (Q, x, Q') \in \mathcal{Q}' \times (I \uplus O) \times \mathcal{Q}' \mid \begin{array}{l} Q' = \{m' \mid \exists m \in Q : m \xrightarrow{x} m' \text{ in } env(N)\} \text{ and} \\ \text{for all } m' \in Q' : m' \text{ is } b\text{-bounded in } env(N)\} \\ \uplus \{ (Q, x, \mathbb{U}) \in \mathcal{Q}' \times (I \uplus O) \times \mathcal{Q} \mid \begin{array}{l} Q' = \{m' \mid \exists m \in Q : m \xrightarrow{x} m' \text{ in } env(N)\} \text{ and} \\ \text{there exists } m' \in Q' : m' \text{ is not } b\text{-bounded in } env(N)\} \\ \uplus \{ (\mathbb{U}, x, \mathbb{U}) \mid x \in I \uplus O \}, \end{array} \end{array}$
- $Q_{BSD_b(N)} = \begin{cases} \{m' \mid m_{env(N)} \xrightarrow{\epsilon} m'\}, & \text{if all these } m' \text{ are } b\text{-bounded} \\ \mathbb{U}, & \text{otherwise,} \end{cases}$
- $\Sigma^{in} = I$,
- $\Sigma^{out} = O$, and
- $\lambda(Q) = \begin{cases} 0, & \text{if } Q = \mathbb{U} \\ 1, & \text{if } \emptyset \neq Q \neq \mathbb{U} \\ & \wedge \exists m \in Q : m \text{ is dead except for inputs in } env(N) \\ 2, & \text{if } \emptyset \neq Q \neq \mathbb{U} \\ & \wedge \nexists m \in Q : m \text{ is dead except for inputs in } env(N) \\ & \wedge \exists m \in Q : m \text{ is a stop except for inputs in } env(N) \\ 3, & \text{if } \emptyset \neq Q \neq \mathbb{U} \\ & \wedge \nexists m \in Q : m \text{ is a stop except for inputs in } env(N) \\ 4, & \text{if } Q = \emptyset. \end{cases}$

The state $Q = \emptyset$ is the *empty state* of $BSD_b(N)$, which we denote by Q_\emptyset .

The LTS $BSD_b(N)$ has three properties that directly follow from the construction in Def. 99. First, $BSD_b(N)$ is always finite, τ -free, and deterministic because of the powerset construction. Second, the language of $BSD_b(N)$ is Σ^* because every state (even Q_\emptyset and \mathbb{U}) has an outgoing x -labeled transition, for all $x \in \Sigma$. Third, the information captured in the label of a state m of the nondeterministic LTS we used to construct $BSD_b(N)$ (i.e., m is a marking of $env(N)$) is preserved by the label of the state $Q \ni m$ of $BSD_b(N)$. For example, assume that we labeled m with 2; that is, m is a stop but not dead except for inputs in $env(N)$. The information that we captured by labeling m with 2 is that the trace w leading to m in $env(N)$ is a *stop-trace* of N . This implies that $w \in stop_b(N) \subseteq L_b(N) \subseteq \Sigma^*$ by Def. 84. By the powerset construction applied above, all states that are reachable with w in $env(N)$ are merged into the state Q , and Q is the *only state that is reachable by w* in $BSD_b(N)$. Thus, by labeling Q with the minimum label of its contained states (here, $m \in Q$ and, therefore, $\lambda(Q) \leq 2$), the information that we captured about w is preserved: If $\lambda(Q) = 2$, then there exists an $m' \in Q$ (for example, $m' = m$) that is a stop but not dead except for inputs and reachable with w in $env(N)$, implying $w \in stop_b(N) \subseteq L_b(N) \subseteq \Sigma^*$. If $\lambda(Q) < 2$, then there even exists a *dead-trace* w or a *bound $_b$ -violator* w of N . Still, this implies $w \in stop_b(N) \subseteq L_b(N) \subseteq \Sigma^*$.

We summarize these three properties in the following corollary. For readability reasons, we write L_i instead of $L_i(BSD_b(N))$, for $i \in \{0, \dots, 4\}$.

Corollary 100 [languages of BSD_b]

For an open net N and $BSD_b(N)$, we have

1. $BSD_b(N)$ is finite, τ -free, and deterministic.
2. $L_0 = \text{bound}_b(N)$
3. $L_0 \uplus L_1 = \text{dead}_b(N)$
4. $L_0 \uplus L_1 \uplus L_2 = \text{stop}_b(N)$
5. $L_0 \uplus L_1 \uplus L_2 \uplus L_3 = L_b(N)$
6. $L_0 \uplus L_1 \uplus L_2 \uplus L_3 \uplus L_4 = \Sigma^* = L$
7. $L_1 \uplus L_2 \uplus L_3 \uplus L_4 = \text{co-bound}_b(N)$
8. $L_2 \uplus L_3 \uplus L_4 = \text{co-dead}_b(N)$
9. $L_3 \uplus L_4 = \text{co-stop}_b(N)$
10. $L_4 = \text{co-}L_b(N)$

Example 101 Consider again the open net D in Fig. 54a. For convenience, we recall D 's 1-bounded *stopdead*-semantics from Ex. 85; that is, the sets $\text{bound}_1(D)$, $\text{dead}_1(D) = \text{dead}(D) \cup \text{bound}_1(D)$, $\text{stop}_1(D) = \text{stop}(D) \cup \text{bound}_1(D)$, and $L_1(D) = L(D) \cup \text{bound}_1(D)$ arising from

$$\begin{aligned}
 L(D) &= \{w \in \{s, q, d\}^* \mid \forall v \sqsubseteq w : |v|_d \leq |v|_q\} \\
 &\quad \cup \{w f z \mid w, z \in \{s, q, d\}^* \wedge \forall v \sqsubseteq w : |v|_d \leq |v|_q \\
 &\quad \quad \quad \wedge |w|_s > 0 \wedge |z|_d \leq |w|_q - |w|_d\}, \\
 \text{bound}_1(D) &= \uparrow \{w \in L(D) \mid \exists v \sqsubseteq w : |v|_d + 1 < |v|_q\} \\
 &\quad \cup \uparrow \{w \in L(D) \mid \exists v \sqsubseteq w : |v|_f + 1 < |v|_s\}, \\
 \text{stop}(D) &= \{w \in \{q, d\}^* \mid \forall v \sqsubseteq w : |v|_d \leq |v|_q \wedge |w|_d = |w|_q\} \\
 &\quad \cup \{w f z \mid w, z \in \{s, q, d\}^* \wedge \forall v \sqsubseteq w : |v|_d \leq |v|_q \\
 &\quad \quad \quad \wedge |w|_s > 0 \wedge |z|_d \leq |w|_q - |w|_d\}, \text{ and} \\
 \text{dead}(D) &= \text{stop}(D).
 \end{aligned}$$

Figure 61 shows a part of the reachability graph of the labeled net $\text{env}(D)$. We do not show the complete reachability graph of $\text{env}(D)$ because it is too big; it consists of at least 24 relevant states for bound 1. The LTS $BSD_1(D)$ in Fig. 62 is finite, τ -free, and deterministic. Due to the powerset construction, a state of $BSD_1(D)$ is a set of states of $\text{RG}(\text{env}(D))$. For example, the markings m_4 , m_5 , and m_1 form the state Q_2 of $BSD_1(D)$. The contained markings of a state of $BSD_1(D)$ determine the state's label according to Def. 99. For example, the state Q_2 is labeled with 3 because m_4 , m_5 , and m_1 are no stops except for inputs.

$BSD_1(D)$ encodes the 1-bounded *stopdead*-semantics of D according to Cor. 100. For example, D never sends an f twice, thus, every trace of $BSD_1(D)$ with more than one f either leads to the empty state Q_\emptyset or is a continuation of a trace leading to state \mathbb{U} . Observe that the states Q_4 – Q_9

are only reachable from the states Q_0 – Q_3 with an f -labeled transition; either via transition $Q_1 \xrightarrow{f} Q_4$ or via transition $Q_3 \xrightarrow{f} Q_8$. An f -labeled transition corresponds to the firing of transition f in $env(D)$ or, equivalently, to D producing a token on the output place f . The states Q_4 – Q_9 also form a kind of a “sink”, meaning no transition from Q_4 – Q_9 leads back to the state Q_0 . In other words, the initial marking of D is no longer reachable once a message f was send.

Furthermore, no state Q of $BSD_1(D)$ is labeled with 2; thus, $L_2(BSD_1(D)) = \emptyset$ and $dead_b(D) = stop_b(D)$ by Cor. 100. We observed this already in Ex. 85 by concluding $stop(D) = dead(D)$ from D 's final marking. \diamond

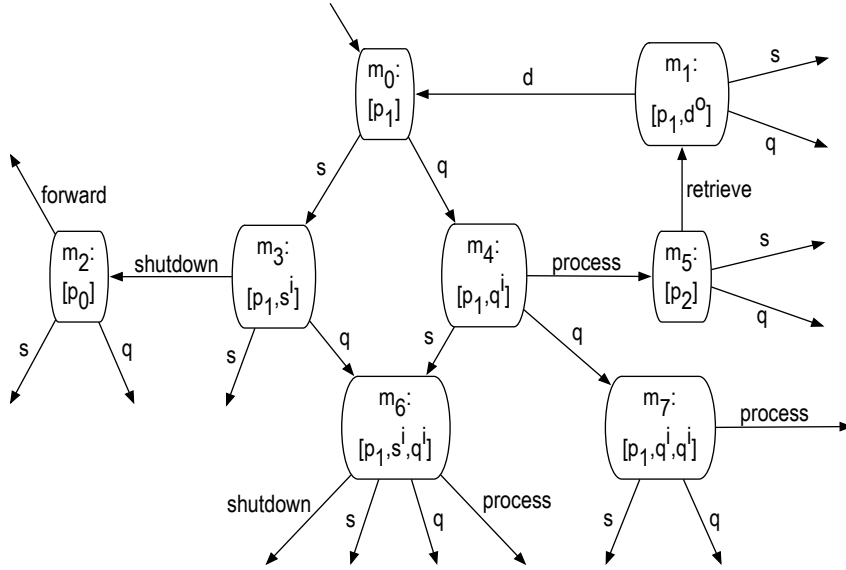


Figure 61: A part of the reachability graph of the environment of the open net D in Fig. 54a.

Proposition 88 gives a procedure to decide whether two composable open nets N_1 and N_2 are b -responsive based on intersections of their respective b -bounded *stopdead*-semantics. With Cor. 100, we showed that we can encode all four languages of the b -bounded *stopdead*-semantics of an open net N using one LTS with five state labels—that is, the LTS $BSD_b(N)$. We combine both results to develop an algorithm that decides b -responsiveness solely on the basis of $BSD_b(N_1)$ and $BSD_b(N_2)$: As both LTS are deterministic and $L(BSD_b(N_1)) = L(BSD_b(N_2)) = \Sigma^*$, there exists a unique least bisimulation relation ρ between them. Comparing the labels of all states that are related with ρ now gives the language intersections of Prop. 88 and, thus, b -responsiveness; this is not hard to see.

Theorem 102 [deciding b -responsiveness with BSD_b]

Let N_1 and N_2 be two composable open nets such that $N_1 \oplus N_2$ is a closed net. Then N_1 is a b -partner of N_2 if and only if $BSD_b(N_1)$ and $BSD_b(N_2)$ are bisimilar with the least bisimulation ρ such that for all $(Q_1, Q_2) \in \rho$,

$$\lambda_{BSD_b(N_1)}(Q_1) + \lambda_{BSD_b(N_2)}(Q_2) > 3.$$

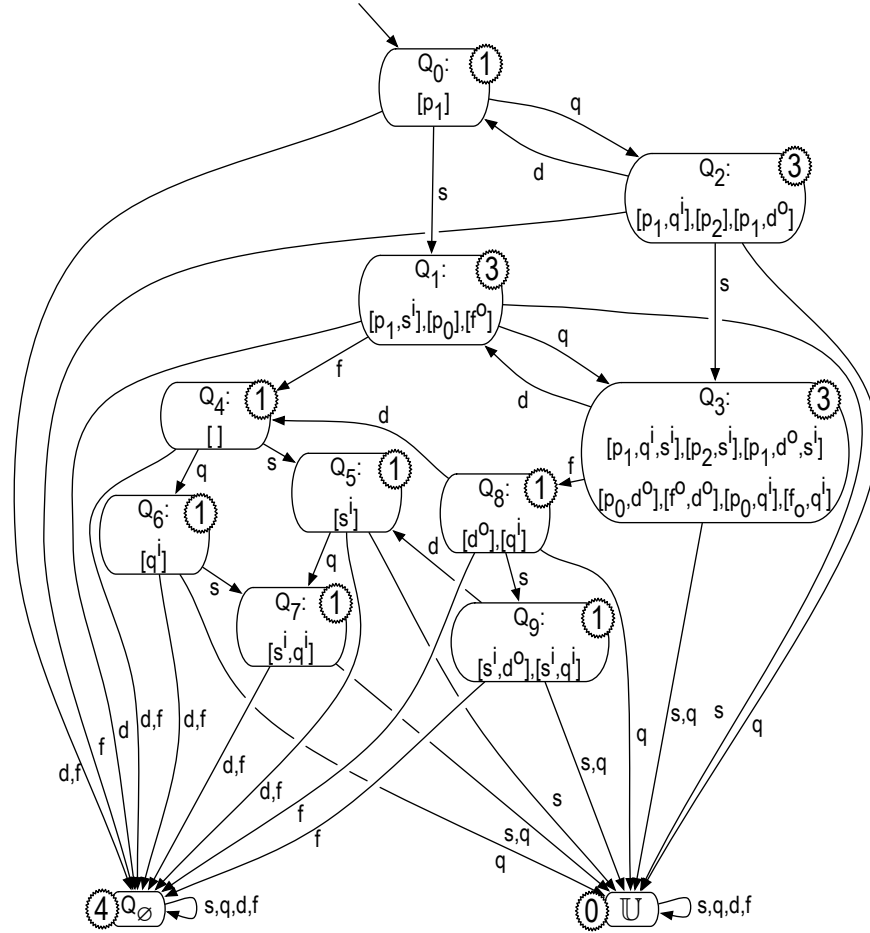


Figure 62: The LTS $BSD_1(D)$ encodes the 1-bounded *stopdead*-semantics of the open net D in Fig. 54a. We depict the label of each state as an encircled number in the upper right corner of that state, except for the states Q_\emptyset and U ; their respective label is shown on the left-hand side.

Proof. \Rightarrow : By assumption, N_1 and N_2 are composable and $N_1 \oplus N_2$ is a closed net. So with Cor. 100(6), we have $L(BSD_b(N_1)) = L(BSD_b(N_2)) = \Sigma^*$. As $BSD_b(N_1)$ and $BSD_b(N_2)$ are deterministic by Cor. 100(1), we conclude the existence of the unique least bisimulation ϱ . Applying Prop. 88 and Cor. 100(2)–(5), we conclude that $\lambda_{BSD_b(N_1)}(Q_1) + \lambda_{BSD_b(N_2)}(Q_2) \neq 3$ and, by the inclusion of the languages of the b -bounded *stopdead*-semantics, it cannot be that $\lambda_{BSD_b(N_1)}(Q_1) + \lambda_{BSD_b(N_2)}(Q_2) < 3$. Thus, we have $\lambda_{BSD_b(N_1)}(Q_1) + \lambda_{BSD_b(N_2)}(Q_2) > 3$.

\Leftarrow : By Def. 44, we have to show that N_1 and N_2 are b -responsive. By assumption and Cor. 100(2)–(5), we conclude that $bound_b(N_1) \cap L_b(N_2) = \emptyset$, $dead_b(N_1) \cap stop_b(N_2) = \emptyset$, $stop_b(N_1) \cap dead_b(N_2) = \emptyset$, and finally $L_b(N_1) \cap bound_b(N_2) = \emptyset$. Thus, N_1 and N_2 are b -responsive by Prop. 88. \square

Example 103 Consider again the open nets D and U in Fig. 54a and Fig. 54b. Figure 63 depicts the LTS $BSD_1(U)$. We already claimed in Ex. 45 that U is a 1-partner of D . Now, with Thm. 102, we verify this claim using

the LTS $BSD_1(D)$ and $BSD_1(U)$: $BSD_1(D)$ and $BSD_1(U)$ are bisimilar with the following bisimulation

$$\varrho = \{(Q_0, A_0), (Q_2, A_2), (Q_0, A_4), (Q_\emptyset, \mathbb{U}), (\mathbb{U}, Q_\emptyset), (Q_\emptyset, Q_\emptyset), \\ (Q_\emptyset, A_1), (Q_\emptyset, A_3), (Q_\emptyset, A_5), (Q_1, Q_\emptyset), (Q_3, Q_\emptyset), (Q_4, Q_\emptyset), \\ (Q_5, Q_\emptyset), (Q_6, Q_\emptyset), (Q_7, Q_\emptyset), (Q_8, Q_\emptyset), (Q_9, Q_\emptyset)\}.$$

To see that ϱ is truly a bisimulation, consider the following trivial statement: If a state Q of $BSD_1(D)$ is related to Q_\emptyset or \mathbb{U} of $BSD_1(U)$, then every state reachable from Q in $BSD_1(D)$ is related to Q_\emptyset or \mathbb{U} , respectively, and vice versa. In other words, once we relate Q to Q_\emptyset or \mathbb{U} , it is straight forward to see which pairs we have to add to a bisimulation relation between $BSD_1(D)$ and $BSD_1(U)$.

Now, we compare the labels of each pair of states that are related by ϱ according to Thm. 102. Because $\lambda(Q_\emptyset) = 4$ by Def. 99, a pair in ϱ containing Q_\emptyset trivially fulfills Thm. 102. Thus, it suffices to check all pairs of ϱ without Q_\emptyset —that is, the pairs $(Q_0, A_0), (Q_2, A_2), (Q_0, A_4) \in \varrho$. We have $\lambda_{BSD_1(D)}(Q_0) + \lambda_{BSD_1(U)}(A_0) = 3 + 1 > 3$, $\lambda_{BSD_1(D)}(Q_2) + \lambda_{BSD_1(U)}(A_2) = 1 + 3 > 3$, and $\lambda_{BSD_1(D)}(Q_0) + \lambda_{BSD_1(U)}(A_4) = 1 + 3 > 3$. Therefore, U is a 1-partner of D by Thm. 102. \diamond

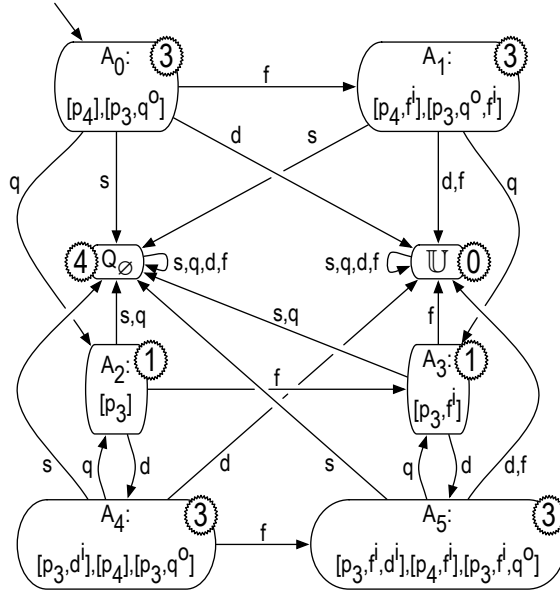


Figure 63: The LTS $BSD_1(U)$ encodes the 1-bounded *stopdead*-semantics of the open net U in Fig. 54b. We depict the label of each state as an encircled number in the upper right corner of that state, except for the state Q_\emptyset , whose label is shown on the left-hand side.

5.2.2 Deciding b -conformance using the LTS CSD_b

In the following, we develop a decision procedure for b -conformance. Although the LTS BSD_b represents the b -bounded *stopdead*-semantics of an open net in a finite manner, it is not sufficient to consider $BSD_b(Impl)$ and $BSD_b(Spec)$ of two composable open nets $Impl$ and $Spec$ to decide whether $Impl$ b -conforms to $Spec$: We already explained in Ex. 91 and Ex. 92 that

the converse of Thm. 90 does not hold. As b -conformance and b -coverable *stopdead*-inclusion coincide by Thm. 97, we construct an LTS CSD_b similar to BSD_b that encodes the b -coverable *stopdead*-semantics.

As a starting point, we take the LTS $BSD_b(N)$ that represents the b -bounded *stopdead*-semantics of an open net N by Cor. 100. The b -bounded *stopdead*-semantics consists of the languages $bound_b(N)$, $dead_b(N)$, $stop_b(N)$, and $L_b(N)$. The b -coverable *stopdead*-semantics of N consists of the languages $uncov_b(N)$, $udead_b(N)$, $ustop_b(N)$, and $uL_b(N)$; these languages include the languages $bound_b(N)$, $dead_b(N)$, $stop_b(N)$, and $L_b(N)$, respectively, by Cor. 95. We sketch the main idea for representing $uncov_b(N)$, $udead_b(N)$, $ustop_b(N)$, and $uL_b(N)$ by a finite LTS using the language $udead_b(N)$ as an example: The difference between $dead_b(N)$ and $udead_b(N)$ results solely from b -uncoverable traces w of N that are not included in $bound_b(N)$ —that is, $w \in co-bound_b(N)$. In $BSD_b(N)$, $dead_b(N)$ is represented by the disjoint languages $L_0(BSD_b(N)) = bound_b(N)$ and $L_1(BSD_b(N)) = dead_b(N) \setminus bound_b(N)$. If we consecutively identify such b -uncoverable traces $w \in co-bound_b(N)$ and shift them into the “error” language $L_0(BSD_b(N))$, then, eventually, $L_0(BSD_b(N))$ represents $uncov_b(N)$. That way, we may also shift traces from $L_1(BSD_b(N))$ to $L_0(BSD_b(N))$. However, these traces are identified as b -uncoverable traces and, therefore, also traces in $udead_b(N)$. In other words, the language $L_0(BSD_b(N)) \uplus L_1(BSD_b(N))$ eventually represents the language $udead_b(N)$. This strategy works equally well for $L_0(BSD_b(N)) \uplus L_1(BSD_b(N)) \uplus L_2(BSD_b(N)) = stop_b(N)$ and $L_0(BSD_b(N)) \uplus L_1(BSD_b(N)) \uplus L_2(BSD_b(N)) \uplus L_3(BSD_b(N)) = L_b(N)$, eventually transforming them into $ustop_b(N)$ and $uL_b(N)$. Figure 64 illustrates the idea of shifting b -uncoverable traces into the language $L_0(BSD_b(N))$ of BSD_b .

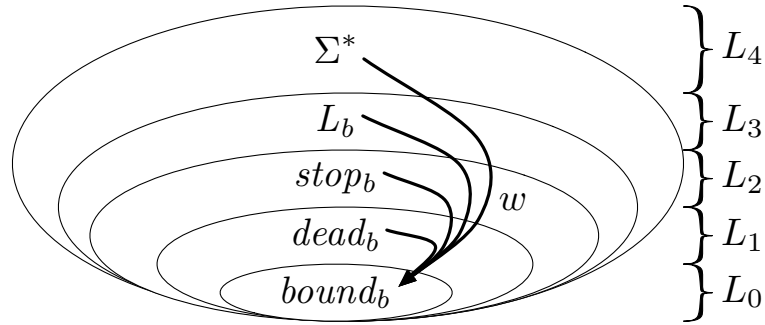


Figure 64: Shifting b -uncoverable traces w from $co-bound_b$ into the “error” language $L_0(BSD_b(N))$. The right-hand side illustrates the languages $L_0(BSD_b(N))$ – $L_4(BSD_b(N))$ of the LTS BSD_b .

Figure 65 illustrates the five languages after shifting all b -uncoverable traces to $L_0(BSD_b(N))$: We have $L_0(BSD_b(N)) \subseteq L_0(CSD_b(N))$ because $L_0(CSD_b(N))$ additionally contains all b -uncoverable traces. Because the language $L_0(BSD_b(N))$ is included in $dead_b(N)$, $stop_b(N)$, and $L_b(N)$, they equally grow, yielding the languages $udead_b(N)$, $ustop_b(N)$, and $uL_b(N)$. The language $L_0(BSD_b(N)) \uplus L_1(BSD_b(N)) \uplus L_2(BSD_b(N)) \uplus L_3(BSD_b(N)) \uplus L_4(BSD_b(N)) = \Sigma^*$ does not grow because it represents all possible traces.

In the following, we describe how to find b -uncoverable traces and how we shift them into $L_0(BSD_b(N))$. In $BSD_b(N)$, the language $L_0(BSD_b(N))$ initially represents $bound_b(N)$. From this, we iteratively identify a b -uncoverable trace w in $BSD_b(N)$ by one of the following two cases:

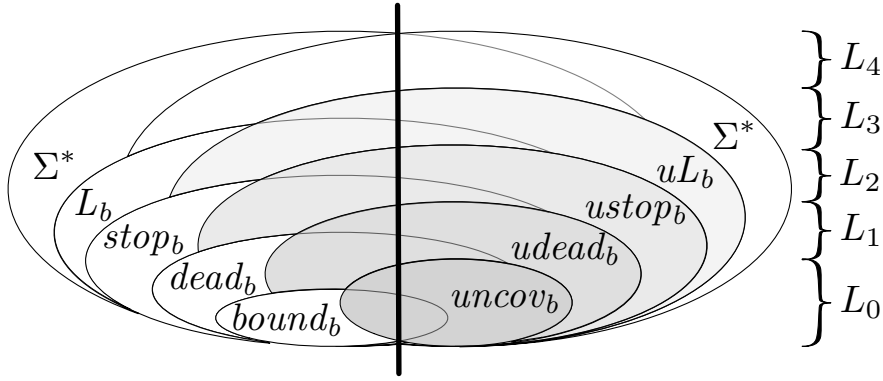


Figure 65: An illustration of how the languages of the b -bounded *stopdead*-semantics (left-hand Euler diagram) compare to the languages of the b -coverable *stopdead*-semantics (right-hand Euler diagram). The right-hand Euler diagram derives from the left-hand Euler diagram by shifting b -uncoverable traces into the “error” language $L_0(BSD_b(N))$, thereby eventually transforming $bound_b$ into $uncov_b$. The far right-hand side of Fig. 65 illustrates the languages $L_0(CSD_b(N))$ – $L_4(CSD_b(N))$ of the LTS CSD_b .

1. If w is a *dead*-trace of N , then every b -partner C of N with $w \in L(C)$ *must* be able to send a message to N after executing w ; otherwise, the composition $N \oplus C$ is in a nonresponsive marking: N cannot send a message to C , C cannot send a message to N , and no final marking is reachable in $N \oplus C$ because $w \in dead(N)$ (i.e., there is no final marking reachable in N). However, if every continuation wx of w with $x \in I_N$ leads to \mathbb{U} —that is, wx is a b -uncoverable trace—no b -partner of N sends a message to N after w . Thus, w must be a b -uncoverable trace of N , too.
2. If w is a trace of N and there exists a continuation wx of w with $x \in O_N$ leading to \mathbb{U} —that is, wx is a b -uncoverable trace of N —then no b -partner C of N may perform w , because transition $x \in I_C$ is always enabled in $env(C)$. Thus, w must be a b -uncoverable trace of N as well. Something similar is also known as output-pruning, where w is considered erroneous if wx reaches an error state; compare, for example, to [81, 18].

The idea for shifting b -uncoverable traces w into $L_0(BSD_b(N))$ is to merge the state of $BSD_b(N)$ reachable by w into the “error state” \mathbb{U} , yielding a new LTS $S^1(N)$. That way, all continuations of w also lead to \mathbb{U} in $S^1(N)$, just like all continuations of a b -uncoverable trace are b -uncoverable as well. We iteratively merge states of b -uncoverable traces into \mathbb{U} until we reach a fix point. We refer to the resulting LTS as $CSD_b(N)$. The state \mathbb{U} of $CSD_b(N)$ encodes the b -uncoverable traces of N in the same way as \mathbb{U} in $BSD_b(N)$ encodes the b -violators of N . The languages $L_0(CSD_b(N))$ – $L_4(CSD_b(N))$ of $CSD_b(N)$ represent the b -coverable *stopdead*-semantics of N as illustrated at the right-hand side of Fig. 65.

In Thm. 115, we will prove the correctness of our construction in Def. 104; that is, $L_0(CSD_b(N))$ represents $uncov_b(N)$ of an open net N and, thus, $CSD_b(N)$ characterizes N ’s b -coverable *stopdead*-semantics. For the proof, we also introduce an LTS $MP_b(N)$ in Def. 104 that results from $CSD_b(N)$ with the state \mathbb{U} removed.

Definition 104 [labeled transition systems CSD_b and MP_b]

Let N be an open net. For $i \in \mathbb{N}^+$, we recursively define the LTS $S^i(N)$ as follows:

- (Base): $S^1(N) = BSD_b(N)$.
- (Step): Let $Q \in \mathcal{Q}_{S^i(N)} \setminus \{\mathbb{U}\}$ be a state of $S^i(N)$ such that
 1. $\lambda(Q) = 1$ and for all $x \in \Sigma^{in}$, $Q \xrightarrow{x} \mathbb{U}$; or
 2. there exists an $x \in \Sigma^{out}$, $Q \xrightarrow{x} \mathbb{U}$.

We obtain the LTS $S^{i+1}(N)$ from $S^i(N)$ as follows:

- If Q is the initial state of $S^i(N)$, define \mathbb{U} as the initial state of $S^{i+1}(N)$ and remove Q .
- If Q is not the initial state of $S^i(N)$, redirect every incoming transition of Q to \mathbb{U} and remove Q .

Thereby, the removal of Q includes the removal of its outgoing transitions and all states and transitions that become unreachable from the initial state of $S^{i+1}(N)$.

We define $CSD_b(N) = S^j(N)$ for the smallest $j \in \mathbb{N}^+$ with $S^j(N) = S^{j+1}(N)$. We obtain the LTS $MP_b(N)$ from $CSD_b(N)$ by removing state \mathbb{U} and its outgoing transitions.

Example 105 We illustrate Def. 104 with the construction of $CSD_1(D)$ of the open net D in Fig. 54a. Figure 62 depicts the initial labeled transition system $S^1(D) = BSD_1(D)$. Figure 66 depicts the LTS $S^7(D)$ that we obtained from the LTS $S^1(D)$ by iteratively removing the states Q_9 , Q_8 , Q_7 , Q_6 , Q_5 , and Q_4 in this order. The states Q_4 – Q_9 were removed because of Def. 104(1). There are two states in $S^7(D)$ —state Q_1 and state Q_3 —with an outgoing f -labeled transition to state \mathbb{U} . These states must be removed according to Def. 104(2): No 1-partner of D can perform a trace containing s —that is, sending a message s —as this eventually leads to a nonfinal and nonresponsive marking in the composition with D . We already explained this fact in detail in Ex. 45 and Ex. 94. Removing the states Q_1 and Q_3 from $S^7(D)$ results in the LTS $S^9(D) = CSD_1(D)$, which we depict in Fig. 67. \diamond

In the following, we show that for any open net N , the language L_0 of $CSD_b(N)$ represents the set $uncov_b(N)$ —that is, the set of all b -uncoverable traces of N . From this, we can easily conclude the correctness of our construction in Def. 104; that is, $CSD_b(N)$ represents the b -coverable *stopdead*-semantics of an open net N . As the proof is quite complex, we prove each inclusion in a separate subsection.

5.2.2.1 Every trace that reaches the state \mathbb{U} is a b -uncoverable trace

The next lemma shows that, for every state Q that is removed from $BSD_b(N)$ according to Def. 104, the traces leading to Q (and their continuations) are b -uncoverable traces of N .

Lemma 106

For an open net N , we have $L_0(CSD_b(N)) \subseteq uncov_b(N)$.

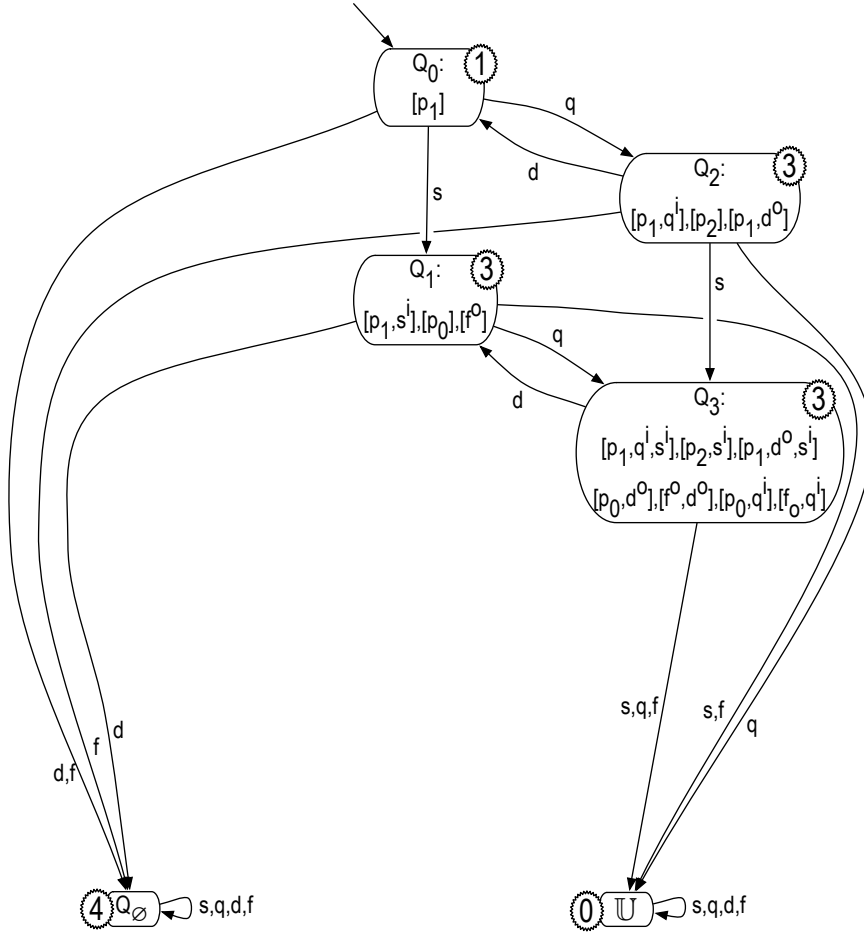


Figure 66: The LTS $S^7(D)$ which we obtained from $BSD_1(D)$ in Fig. 62 by iteratively removing the states Q_9, Q_8, Q_7, Q_6, Q_5 , and Q_4 in this order according to Def. 104(1). We depict the label of each state as an encircled number in the upper right corner of that state, except for the states Q_\emptyset and U , whose labels are shown on the left-hand side.

Proof. Induction over the sequence of labeled transition systems $BSD_b(N) = S^1(N), \dots, S^j(N) = CSD_b(N)$ with $j \in \mathbb{N}^+$ in Def. 104.

(Base): $L_0(S^1(N)) = bound_b(N)$ by Cor. 100(2) and $bound_b(N) \subseteq uncov_b(N)$ by Cor. 95; thus, $L_0(S^1(N)) \subseteq uncov_b(N)$.

(Step): Assume $L_0(S^i) \subseteq uncov_b(N)$ for $i \in \mathbb{N}^+$, and let Q be the state that is removed from $S^i(N)$ to obtain $S^{i+1}(N)$. Let $w \in L_0(S^{i+1}(N)) \setminus L_0(S^i(N))$, which exists by the construction of $S^{i+1}(N)$. The trace w is uniquely defined because $S^i(N)$ is deterministic. By the construction of $S^{i+1}(N)$ from $S^i(N)$ and the choice of w , the run underlying w visits the state Q in $S^i(N)$ —that is, $Q_{S^i(N)} \xrightarrow{u} Q \xrightarrow{v} Q'$ in $S^i(N)$ such that $w = uv$. We show that $u \in uncov_b(N)$, which implies $w \in uncov_b(N)$ by Def. 93. By Def. 104, we distinguish two cases for the choice of Q in S^i :

1. Let $\lambda(Q) = 1$ and for all $x \in \Sigma^{in}$, $Q \xrightarrow{x} U$:

$\lambda(Q) = 1$ implies $u \in dead_b(N)$ by Cor. 100, thus for all b -partners C of N , $u \notin stop_b(C)$ by Prop. 88. Then, for all b -partners C of N , either $u \notin L_b(C)$ or there exists an $x \in \Sigma^{in} = O_C$ such that $ux \in L_b(C)$. The latter cannot be the case, because for all $x \in \Sigma^{in}$, $Q \xrightarrow{x} U$ implies (by

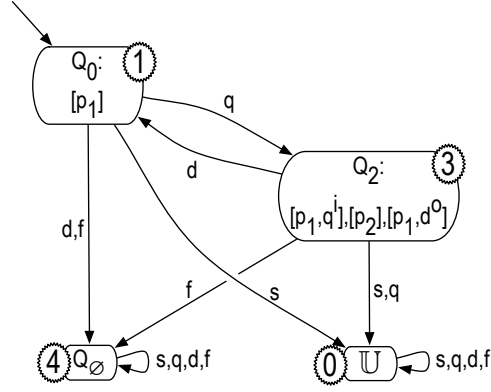


Figure 67: The LTS $CSD_1(D)$ encodes the 1-coverable *stopdead*-semantics of the open net D in Fig. 54a. We depict the label of each state as an encircled number in the upper right corner of that state, except for the states Q_\emptyset and \mathbb{U} , whose labels are shown on the left-hand side.

induction argument) that for all $x \in \Sigma^{in}$, $ux \in \text{uncov}_b(N)$, violating the assumption the C is a b -partner of N . Thus, for all b -partners C of N , $u \notin L_b(C)$, and, therefore, $u \in \text{uncov}_b(N)$.

2. Assume there exists an $x \in \Sigma^{out}$, $Q \xrightarrow{x} \mathbb{U}$:

By induction argument, $ux \in \text{uncov}_b(N)$ and for all b -partners C of N , $ux \notin L_b(C)$. As $x \in \Sigma^{out} = I_C$ is enabled at every marking in $\text{env}(C)$ for a b -partner C of N , $u \notin L_b(C)$, and, therefore, $u \in \text{uncov}_b(N)$. \square

Example 107 We already explained in Ex. 94 that no 1-partner of the open net D in Fig. 54a sends a message s ; for example, $s \in \text{uncov}_b(S)$. This is reflected in $CSD_1(S)$ in Fig. 67, as every s -labeled transition from a state other than Q_\emptyset has the state \mathbb{U} as its target. \diamond

5.2.2.2 Every b -uncoverable trace reaches the state \mathbb{U}

With Lem. 106, we have shown that every trace that reaches the state \mathbb{U} of $CSD_b(N)$ is a b -uncoverable trace of an open net N . In this subsection, we show the converse: Every b -uncoverable trace of N is a trace of $CSD_b(N)$ that reaches the state \mathbb{U} . This implies—together with Lem. 106—that $L_0(CSD_b(N))$ and $\text{uncov}_b(N)$ coincide, which in turn proves the correctness of our construction in Def. 104.

The proof idea for showing the converse of Lem. 106 is simple: First, we remove the state \mathbb{U} from $CSD_b(N)$ and show that the remaining LTS—that is, the LTS $MP_b(N)$ which we already defined in Def. 104—induces an open net $mp_b(N)$ that is composable to N . Then, we show that $mp_b(N)$ has two properties:

1. $mp_b(N)$ is a b -partner of N .
2. Every trace of $MP_b(N)$ is also a trace in the language of $mp_b(N)$.

That way, we conclude that every trace of $MP_b(N)$ is certainly not a b -uncoverable trace of N . Intuitively, the LTS $MP_b(N)$ over-approximates the language of any b -partner of an open net N . By contraposition, our original statement—every b -uncoverable trace of N reaches the state \mathbb{U} —follows.

For an open net N , the LTS $MP_b(N)$ may not necessarily exist: If $CSD_b(N)$ contains solely the state \mathbb{U} , $MP_b(N)$ is not defined by Def. 104. However, if $MP_b(N)$ exists, then we can relate the reachability graph of the inner of an open net C to $MP_b(N)$, capturing the essence of [258, Lem. 1]: If $inner(C)$ is in a certain marking in the composition $N \oplus C$, then $env(N)$ is in a marking contained in a state Q that is related to the marking of $inner(C)$.

Lemma 108 [weak simulation vs. MP_b]

Let N and C be two composable open nets such that $N \oplus C$ is a closed net and $MP_b(N)$ exists. If $RG(inner(C))$ is weakly simulated by $MP_b(N)$ using relation ϱ , then for every reachable marking m of $N \oplus C$, there exists a state Q of $MP_b(N)$ such that $(m|_{inner(C)}, Q) \in \varrho$ and $m|_{env(N)} \in Q$.

Proof. We prove this lemma by induction over the reachable markings of $N \oplus C$.

(Base): The initial marking of $N \oplus C$ is $m_{env(N)} + m_{inner(C)}$. By Def. 5, we have $(m_{inner(C)}, Q_{MP_b(N)}) \in \varrho$ and, by Def. 99 and Def. 104, $m_{env(N)} \in Q_{MP_b(N)}$.

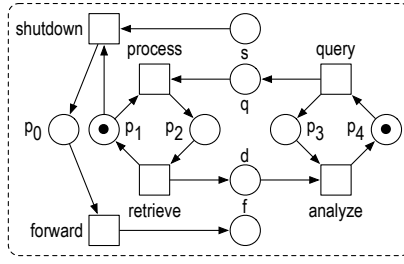
(Step): Let m be reachable in $N \oplus C$ and Q be a state of $MP_b(N)$ such that $(m|_{inner(C)}, Q) \in \varrho$ and $m|_{env(N)} \in Q$. We assume $m \xrightarrow{t} m'$ in $N \oplus C$ and distinguish three cases:

1. Let $t \in T_N$: Then $m|_{inner(C)} = m'|_{inner(C)}$ and $(m'|_{inner(C)}, Q) \in \varrho$. By Def. 17, we have $l(t) = \tau$ in $env(N)$, thus $m'|_{env(N)} \in Q$ by Def. 99.
2. Let $t \in T_{inner(C)}$ and $l(t) = \tau$: Then we have $m|_{inner(C)} \xrightarrow{\tau} m'|_{inner(C)}$ in $RG(inner(C))$, and $(m'|_{inner(C)}, Q) \in \varrho$ since $MP_b(N)$ is τ -free. By Def. 17, $m|_{env(N)} = m'|_{env(N)}$, thus $m'|_{env(N)} \in Q$.
3. Let $t \in T_{inner(C)}$ and $l(t) \neq \tau$: Then $m|_{inner(C)} \xrightarrow{l(t)} m'|_{inner(C)}$ in $RG(inner(C))$ and $Q \xrightarrow{l(t)} Q'$ in $MP_b(N)$ and $(m'|_{inner(C)}, Q') \in \varrho$. By Def. 99, $m'|_{env(N)} \in Q'$. \square

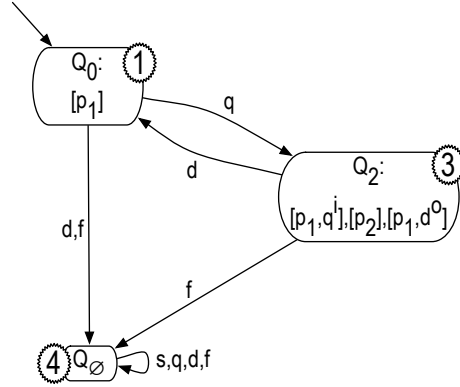
Example 109 Because $CSD_1(D)$ consists of more states than only state \mathbb{U} , the LTS $MP_1(D)$ exists. We obtained the LTS $MP_1(D)$ in Fig. 68b from $CSD_1(D)$ in Fig. 67 by omitting the state \mathbb{U} . Now consider the open net U in Fig. 54b, which is a b -partner of D by Ex. 45. Figure 68a depicts the open net $D \oplus U$, and Fig. 68c depicts the inner net of U . The reachability graph of $inner(U)$ (whose states are markings of $inner(U)$) is weakly simulated by $MP_1(D)$ with the weak simulation relation

$$\varrho = \{([p_4], Q_0), ([p_3], Q_2)\}.$$

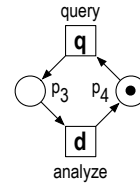
We illustrate Lem. 108 with all reachable markings of $D \oplus U$ —that is, the initial marking $[p_1, p_4]$, and the markings $[p_1, p_3, q]$, $[p_2, p_3]$, and $[p_1, p_3, d]$. If $inner(U)$ is in the marking $[p_4]$, then $env(D)$ is in the marking $[p_1]$. Accordingly, we have $([p_4], Q_0) \in \varrho$ and $[p_1] \in Q_0$. If $inner(U)$ is in the marking $[p_3]$, then $env(D)$ is in the marking $[p_1, q^i]$, $[p_2]$, or $[p_1, d^o]$. Accordingly, we have $([p_3], Q_2) \in \varrho$ and $[p_1, q^i], [p_2], [p_1, d^o] \in Q_2$. \diamond



(a) Open net $D \oplus U$



(b) LTS $MP_1(D)$



(c) Labeled net $inner(U)$

Figure 68: The open net $D \oplus U$ from Fig. 31c, the LTS $MP_1(D)$ derived from $CSD_1(D)$ in Fig. 67, and the inner net of the open net U from Fig. 54b. We depict the label of each state as an encircled number in the upper right corner of that state, except for the state Q_\emptyset , whose label is shown on the left-hand side. In addition to the figures, we have $\Omega_{D \oplus U} = \{[p_0]\}$ and $\Omega_{inner(U)} = \{[]\}$.

In the following, we define the open net $mp_b(N)$ of an open net N which we already motivated at the beginning of this subsection. If $MP_b(N)$ exists, then we derive $mp_b(N)$ from a labeled net that is induced by $MP_b(N)$.

Definition 110 [open net mp_b]

Let N be an open net such that the LTS $MP_b(N) = (Q, \delta, Q_{MP_b(N)}, \Sigma^{in}, \Sigma^{out}, \lambda)$ exists. Then $MP_b(N)$ induces a labeled net $N' = (Q, \delta, F', m_{N'}, \Omega_{N'}, \Sigma^{out}, \Sigma^{in}, l)$ with

- $F' = \{(Q, (Q, x, Q')), ((Q, x, Q'), Q') \mid Q, Q' \in Q \wedge (Q, x, Q') \in \delta\}$,
- $m_{N'} = [Q_{MP_b(N)}]$,
- $\Omega_{N'} = \{[Q] \mid Q \in Q \wedge \lambda(Q) = 2\}$, and
- $l((Q, x, Q')) = x$.

We define $mp_b(N)$ as the open net whose inner net is the labeled net N' .

Example 111 Figure 69 illustrates the construction in Def. 110. The left part of Fig. 69 sketches a part of $MP_b(N)$ for an open net N with three states Q, R, S , an o -labeled transition from Q to R , and an i -labeled transition from Q to S . We have $o \in O_N = \Sigma^{out}$ and $i \in I_N = \Sigma^{in}$, and $\lambda(Q) = 2$,

$\lambda(R) = 1$, and $\lambda(S) = 3$. The right part of Fig. 69 sketches the resulting part of the inner net of $mp_b(N)$: Each state induces a place, and each transition in $MP_b(N)$ induces a transition connecting two places in (the inner of) $mp_b(N)$. Thereby, $[Q]$ is a final marking of (the inner of) $mp_b(N)$, and the labels of the inner of $mp_b(N)$ are reversed compared to N —that is, $o \in \Sigma^{out} = I_{mp_b(N)}$ and $i \in \Sigma^{in} = O_{mp_b(N)}$. \diamond

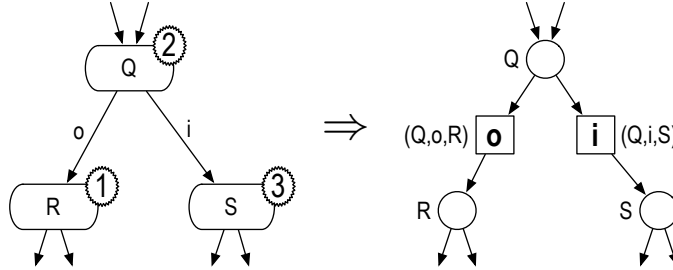


Figure 69: A sketch of the construction in Def. 110. We depict the label of each state as an encircled number in the upper right corner of that state.

If $mp_b(N)$ exists for an open net N , then $mp_b(N)$ is a b -partner of N .

Lemma 112 [mp_b is a b -partner]

Let N be an open net such that $MP_b(N)$ exists. Then $mp_b(N)$ is a b -partner of N .

Proof. By the construction of $mp_b(N)$ in Def. 110, N and $mp_b(N)$ are composable, $N \oplus mp_b(N)$ is a closed net, and $RG(inner(mp_b(N)))$ and $MP_b(N)$ are bisimilar with the bisimulation ϱ . It remains to show that every reachable marking $m = m_{|env(N)} + m_{|inner(mp_b(N))}$ of $N \oplus mp_b(N)$ is b -bounded and responsive.

As $RG(inner(mp_b(N)))$ and $MP_b(N)$ are bisimilar, ϱ is also a (weak) simulation of $RG(inner(mp_b(N)))$ by $MP_b(N)$. With Lem. 108, there exists a state Q of $MP_b(N)$ such that $(m_{|inner(mp_b(N))}, Q) \in \varrho$ and $m_{|env(N)} \in Q$. Each node of $MP_b(N)$ is a set of b -bounded markings of $env(N)$ by Def. 104, thus $m_{|env(N)}$ is b -bounded in $env(N)$. The marking $m_{|inner(mp_b(N))}$ is even 1-bounded by the construction of $mp_b(N)$ in Def. 110. Thus, m is b -bounded in $N \oplus mp_b(N)$.

We show that m is responsive by distinguishing two cases:

1. Let $m_{|env(N)}$ be a stop except for inputs in $env(N)$. We can exclude $\lambda(Q) = 0$ by b -boundedness, thus $\lambda(Q) = 1$ or $\lambda(Q) = 2$ by Def. 99.
 - a) If $\lambda(Q) = 1$: Then there exist an $x \in \Sigma^{in}$ and a state Q' such that $Q \xrightarrow{x} Q'$ in $MP_b(N)$ by the construction of $MP_b(N)$ in Def. 104. Then $m_{|inner(mp_b(N))} \xrightarrow{x}$ in $inner(mp_b(N))$ by the construction of $mp_b(N)$ in Def. 110 and m is responsive in $N \oplus mp_b(N)$ because $x \in O_{mp_b(N)}$.
 - b) If $\lambda(Q) = 2$: Then $m_{|env(N)}$ is not dead except for inputs in $env(N)$ by Def. 99, thus there exists a final marking m' of $env(N)$ such that $m_{|env(N)} \xrightarrow{\varepsilon} m'$. In addition, $m_{|inner(mp_b(N))}$ is a final marking of $inner(mp_b(N))$ by Def. 110. Therefore, the final marking $m' + m_{|inner(mp_b(N))}$ of $N \oplus mp_b(N)$ is reachable from m in $N \oplus mp_b(N)$ and, thus, m is responsive.

2. If $m|_{env(N)}$ is not a stop except for inputs in $env(N)$: Then there exists an $x \in O_N = \Sigma^{out}$ such that $m|_{env(N)} \xrightarrow{x}$ in $env(N)$ by Def. 55, which implies there exists a state Q' such that $Q \xrightarrow{x} Q'$ in $MP_b(N)$ by the construction of $MP_b(N)$ in Def. 104. Thus, $m|_{inner(mp_b(N))} \xrightarrow{x}$ in $inner(mp_b(N))$ by the construction of $mp_b(N)$ in Def. 110. We can repeat this step only a finite number of times, as the number of all tokens on former interface places of N is bounded in $m|_{env(N)}$. Thus, there is a marking m' reachable from m in $N \oplus mp_b(N)$ such that $m'|_{env(N)}$ is a stop except for inputs in $env(N)$, and m' is responsive in $N \oplus mp_b(N)$ by the previously handled case. As m' is reachable from m , m is responsive as well.

We showed that every reachable marking m of $N \oplus C$ is b -bounded and responsive, thus C is a b -partner of N . \square

Example 113 Consider again the LTS $MP_1(D)$ from Fig. 68b, which we rearranged in Fig. 70a to emphasize its similarity to $mp_1(D)$. Figure 70b shows its induced open net $mp_1(D)$ according to Def. 110, and Fig. 70c shows the composition $D \oplus mp_1(D)$. The place p_3 corresponds to the state Q_\emptyset , the place p_4 corresponds to the state Q_0 , and the place p_5 corresponds to the state Q_2 . The place p_4 is initially marked because Q_0 is the initial state of $MP_1(D)$. The set of final markings of $mp_1(D)$ is empty, because no state of $MP_1(D)$ is labeled with 2.

As the place p_3 corresponds to the state Q_\emptyset of $MP_1(D)$, no reachable marking of $D \oplus mp_1(D)$ marks p_3 by Lem. 108. Thus, the transitions t_3 to t_8 never fire in $D \oplus mp_1(D)$. Clearly, $D \oplus mp_1(D)$ is 1-bounded and D and $mp_1(D)$ are responsive, perpetually communicating over the places q and d . Thus, $mp_1(D)$ is a 1-partner of D . \diamond

Knowing that the open net $mp_b(N)$ is a b -partner of N , we have the ingredients to prove the converse of Lem. 106.

Lemma 114

For an open net N , we have $L_0(CSD_b(N)) \supseteq uncov_b(N)$.

Proof. We distinguish two cases:

1. If $MP_b(N)$ does not exist, then $CSD_b(N)$ consists solely of state \mathbb{U} with an x -labeled self-loop for each $x \in I \uplus O$ by Def. 104. Thus, $L_0(CSD_b(N)) = \Sigma^* \supseteq uncov_b(N)$.
2. If $MP_b(N)$ exists, we have $L(MP_b(N)) = L(inner(mp_b(N)))$ by Def. 110. As each trace of $inner(mp_b(N))$ is a trace of $env(mp_b(N))$ by Def. 15, we have $L(MP_b(N)) \subseteq L_b(mp_b(N))$. The open net $mp_b(N)$ is a b -partner of N by Lem. 112, thus $uncov_b(N) \subseteq \Sigma^* \setminus L(MP_b(N)) = L_0(CSD_b(N))$ by Def. 104. \square

With Lem. 106 and Lem. 114, we have shown that the language L_0 of $CSD_b(N)$ coincides with $uncov_b(N)$.

Theorem 115 [CSD_b represents $uncov_b$]

For an open net N , we have $L_0(CSD_b(N)) = uncov_b(N)$.

The next corollary states that the LTS $CSD_b(N)$ of an open net N represents N 's b -coverable *stopdead*-semantics. It follows directly from Thm. 115

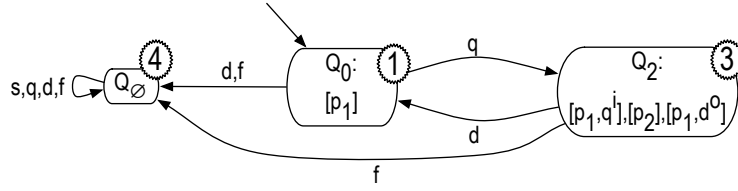
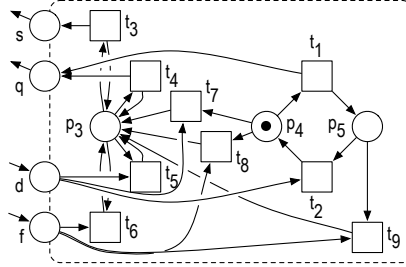
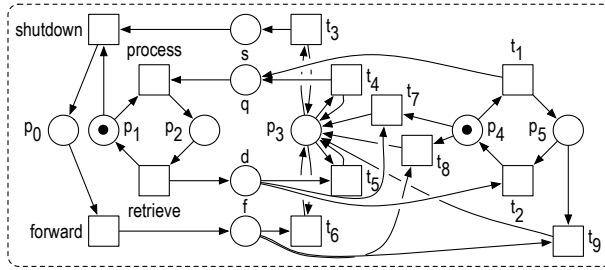
(a) LTS $MP_1(D)$ (b) Open net $mp_1(D)$ (c) Open net $D \oplus mp_1(D)$

Figure 70: The rearranged LTS $MP_1(D)$ from Fig. 68b, the open net $mp_1(D)$ that we derived from $MP_1(D)$ in Fig. 68b according to Def. 110, and its composition $D \oplus mp_1(D)$ with the open net D from Fig. 54a. We depict the label of each state of $MP_1(D)$ as an encircled number in the upper right corner of that state. In addition to the figures, we have $\Omega_{mp_1(D)} = \Omega_{D \oplus mp_1(D)} = \emptyset$.

and the construction of $CSD_b(N)$ in Def. 104. For readability reasons, we write L_i instead of $L_i(CSD_b(N))$, for $i \in \{0, \dots, 4\}$.

Corollary 116 [languages of CSD_b]

For an open net N and $CSD_b(N)$, we have

1. $CSD_b(N)$ is finite, τ -free, and deterministic.
2. $L_0 = \text{uncov}_b(N)$
3. $L_0 \uplus L_1 = \text{udead}_b(N)$
4. $L_0 \uplus L_1 \uplus L_2 = \text{ustop}_b(N)$
5. $L_0 \uplus L_1 \uplus L_2 \uplus L_3 = \text{uL}_b(N)$
6. $L_0 \uplus L_1 \uplus L_2 \uplus L_3 \uplus L_4 = \Sigma^* = L$
7. $L_1 \uplus L_2 \uplus L_3 \uplus L_4 = \text{co-uncov}_b(N)$
8. $L_2 \uplus L_3 \uplus L_4 = \text{co-udead}_b(N)$

$$9. L_3 \uplus L_4 = \text{co-ustop}_b(N)$$

$$10. L_4 = \text{co-uL}_b(N)$$

Finally, we can prove the main claim of this section: We can decide whether an open net *Impl* *b*-conforms to an open net *Spec* on their labeled transition systems $CSD_b(\text{Impl})$ and $CSD_b(\text{Spec})$. The idea is to check for the language inclusions from Thm. 97 using the least bisimulation relation ϱ between $CSD_b(\text{Impl})$ and $CSD_b(\text{Spec})$: If two states Q_{Impl} and Q_{Spec} are related by ϱ , then intuitively they represent the same set of words; we can decide which language of the *b*-coverable *stopdead*-semantics contains these words using the state labels of Q_{Impl} and Q_{Spec} and Cor. 116.

Theorem 117 [deciding *b*-conformance with CSD_b]

For any two interface-equivalent open nets *Impl* and *Spec*, *Impl* *b*-conforms to *Spec* if and only if $CSD_b(\text{Impl})$ and $CSD_b(\text{Spec})$ are bisimilar with the least bisimulation ϱ such that for all $(Q_{\text{Impl}}, Q_{\text{Spec}}) \in \varrho$,

$$\lambda_{CSD_b(\text{Impl})}(Q_{\text{Impl}}) \geq \lambda_{CSD_b(\text{Spec})}(Q_{\text{Spec}}).$$

Proof. \Rightarrow : By interface-equivalence of *Impl* and *Spec* and Cor. 116(6), we have $L(CSD_b(\text{Impl})) = L(CSD_b(\text{Spec})) = \Sigma^*$. As $CSD_b(\text{Impl})$ and $CSD_b(\text{Spec})$ are deterministic by Cor. 116(1), we conclude the existence of the bisimulation ϱ . With Thm. 97 and Cor. 116(2)–(5), we conclude that $\lambda_{CSD_b(\text{Impl})}(Q_{\text{Impl}}) \geq \lambda_{CSD_b(\text{Spec})}(Q_{\text{Spec}})$.

\Leftarrow : By assumption and Cor. 116(2)–(5), we conclude that $\text{uncov}_b(\text{Impl}) \subseteq \text{uncov}_b(\text{Spec})$, $\text{udead}_b(\text{Impl}) \subseteq \text{udead}_b(\text{Spec})$, $\text{ustop}_b(\text{Impl}) \subseteq \text{ustop}_b(\text{Spec})$, and $\text{uL}_b(\text{Impl}) \subseteq \text{uL}_b(\text{Spec})$. Thus, *Impl* *b*-conforms to *Spec* by Thm. 97. \square

Example 118 Consider again the patched database D' from Fig. 56. We already showed in Ex. 98 that D' *b*-conforms to D from Fig. 54a, but D does not *b*-conform to D' . Figure 67 depicts the LTS $CSD_1(D)$ and Fig. 71 depicts the LTS $CSD_1(D')$. The difference of $CSD_1(D')$ to $CSD_1(D)$ is caused by the absence of transition *forward* in D' : At the initial marking $[p_1]$, D' may fire transition *shutdown* after receiving an *s*, yielding the final marking $[\]$. In contrast to D , D' can leave its final marking only by receiving another *s* or a *q*. $CSD_1(D')$ reflects this with the 2-labeled state Q'_1 —a state not present in $CSD_1(D)$.

There exists a least bisimulation relation ϱ between $CSD_1(D)$ and $CSD_1(D')$

$$\varrho = \{(Q_0, Q'_0), (Q_2, Q'_2), (Q_\emptyset, Q_\emptyset), (\mathbf{U}, Q'_1), (\mathbf{U}, \mathbf{U}), (\mathbf{U}, Q_\emptyset)\},$$

which is uniquely defined because both LTS are deterministic. We have

$$\begin{aligned} \lambda_{CSD_1(D)}(Q_0) = 1 &= 1 = \lambda_{CSD_1(D')}(Q'_0), \\ \lambda_{CSD_1(D)}(Q_2) = 3 &= 3 = \lambda_{CSD_1(D')}(Q'_2), \\ \lambda_{CSD_1(D)}(Q_\emptyset) = 4 &= 4 = \lambda_{CSD_1(D')}(Q_\emptyset), \\ \lambda_{CSD_1(D)}(\mathbf{U}) = 0 &< 2 = \lambda_{CSD_1(D')}(Q'_1), \\ \lambda_{CSD_1(D)}(\mathbf{U}) = 0 &= 0 = \lambda_{CSD_1(D')}(\mathbf{U}), \text{ and} \\ \lambda_{CSD_1(D)}(\mathbf{U}) = 0 &< 4 = \lambda_{CSD_1(D')}(Q_\emptyset). \end{aligned}$$

Thus, D' b -conforms to D and D does not b -conform to D' according to Thm. 117, which we already showed in Ex. 98 arguing about their b -coverable *stopdead*-semantics'. \diamond

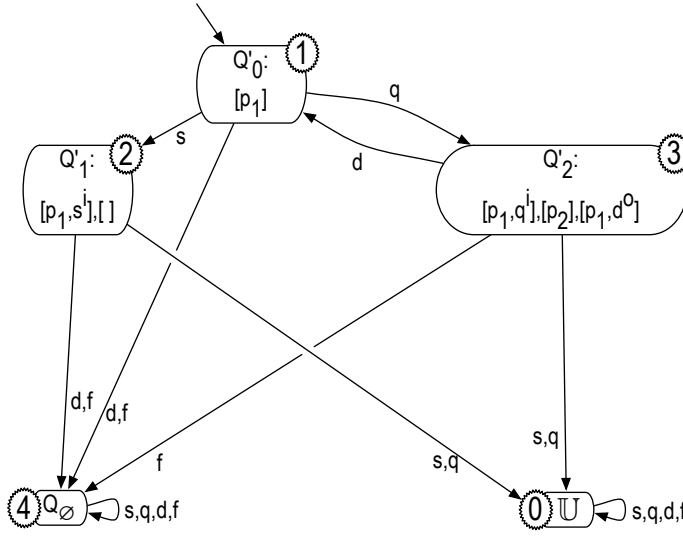


Figure 71: The LTS $CSD_1(D')$ encoding the 1-coverable *stopdead*-semantics of the open net D' in Fig. 56. We depict the label of each state as an encircled number in the upper right corner of that state, except for the states Q_\emptyset and \mathbb{U} , whose labels are shown on the left-hand side.

For the sake of completeness, we also derive the following easy corollary: Every open net with at least one b -partner has a b -partner whose inner net is τ -free. This fact directly follows from Lem. 112, because the inner net of $mp_b(N)$ is τ -free by construction.

Corollary 119

If there exists a b -partner of an open net N , then there exists a b -partner of N whose inner net is τ -free.

We complete this section with a short complexity analysis of the decision procedures for b -responsiveness and b -conformance outlined in Sect. 5.2.1 and Sect. 5.2.2.

5.2.3 Analyzing the computational complexity

Theorem 102 induces an algorithm for deciding whether two given composable open nets N_1 and N_2 are b -partners: First, we compute $BSD_b(N_1)$ and $BSD_b(N_2)$. Second, we build the least bisimulation for $BSD_b(N_1)$ and $BSD_b(N_2)$ and check if the labels of related states satisfy the condition in Thm. 102. Figure 72 illustrates this algorithm.

Algorithm 2 lists an algorithm to compute the LTS $BSD_b(N)$ from a given open net N , which is a straight-forward implementation of Def. 99. In the following, we analyze its computational complexity. Let I and O be fixed with $s = |I \uplus O|$ and let $N = (P, T, F, m_N, \Omega, I, O)$ be an open net. Let n be the number of reachable b -bounded markings in $env(N)$. The construction of the LTS S in line 1 yields at most $n + 1 = O(n)$ states—one state for each b -bounded marking of $env(N)$ and the state \mathbb{U} .

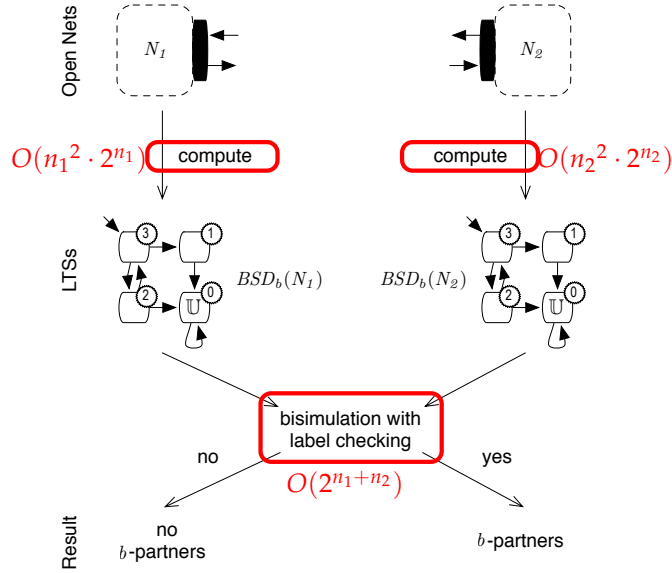


Figure 72: Using BSD_b to decide if two composable open nets N_1 and N_2 are b -partners. We highlighted the complexity of each part of the decision algorithm.

We can compute the *closure* sets in line 2 by employing the Floyd-Warshall-algorithm [95] onto S , thereby setting the weights of all τ -labeled transitions in S to 0 and the weights of all other transitions to 1: The set $closure(m)$ consists of all states within distance 0 from m . The runtime of the Floyd-Warshall-algorithm on S is $O(n^3)$. As $closure(m)$ consists of $O(n)$ states for each state m , every lookup in $closure$ (e.g., lines 3, 6, 7, etc.) takes $O(n)$ time. Therefore, we can compute the status of each marking (lines 4–11) in $O(n^3)$.

The powerset construction in lines 15 to 36 yields at most $2^n + 1 = O(2^n)$ states—one state for each subset of $M_{env(N),b}$ and the state U —and $s \cdot (2^n + 1) = O(2^n)$ transitions. For each transition (Q, x, Q') of $BSD_b(N)$, Q consists of at most $O(n)$ markings $m \in Q$ (line 20). Because $env(N)$ has exactly one x -labeled transition, there exists at most one marking m' that is reachable from m in S via an x -labeled transition (line 21). For each marking m' in turn, we have to consider its closure $closure(m')$ (lines 22–29), which are at most $O(n)$ markings. Thus, the time complexity for each transition of $BSD_b(N)$ is $O(n^2)$. Notice that the loop in line 18 is constant because $s = |\Sigma|$ is constant. We touch each of the $O(2^n)$ transition at most once, yielding a worst case complexity of $O(n^3) + O(n^2 \cdot 2^n) = O(n^2 \cdot 2^n)$ for computing $BSD_b(N)$.

Algorithm 3 lists an algorithm to compute the least bisimulation between two LTS $BSD_b(N_1)$ and $BSD_b(N_2)$ and check their related state-labels according to Thm. 102. Let n_i be the number of reachable b -bounded markings in $env(N_i)$ for $i \in \{1, 2\}$. In the worst case, we have to consider each pair of states of $BSD_b(N_1)$ and $BSD_b(N_2)$ —that is, $O(2^{n_1} \cdot 2^{n_2}) = O(2^{n_1 + n_2})$ states. The check for each pair requires $O(s)$. Therefore, the algorithm in Alg. 3 has a worst case complexity of $O(2^{n_1 + n_2})$.

By Thm. 102, we can combine Alg. 2 and Alg. 3 to decide whether two open nets N_1 and N_2 are b -partner. Let n_i be the number of reachable b -bounded markings in $env(N_i)$ for $i \in \{1, 2\}$. Then, the worst case complexity to decide b -responsiveness is $O(n_1^2 \cdot 2^{m_1}) + O(n_2^2 \cdot 2^{m_2}) + O(2^{n_1 + n_2})$. Figure 72 also illustrates the complexity of the parts of the decision algorithm.

```

Input : open net  $N$ 
Output : LTS  $BSD_b(N)$ 

1 construct LTS  $S = (Q_S, \delta_S, q_S, I_N, O_N, \Omega_S)$  from  $RG(Env(N))$  but stop
  at each bound-violation and merge them into the state  $\mathbb{U} \in Q_S$ 
2 compute  $closure(m) = \{m' \mid m \xrightarrow{\varepsilon} m'\}$  in  $S$  for all  $m \in Q_S$ 
3 if  $\mathbb{U} \in closure(q_S)$  then return  $(\{\mathbb{U}\}, \emptyset, \mathbb{U}, I_N, O_N, \lambda)$ 
4 foreach  $m \in Q_S$  do
5   if  $\forall m' \in closure(m) : \forall x \in O_N : m' \not\xrightarrow{x}$  in  $S$  then
6     if  $\forall m' \in closure(m) : m' \notin \Omega_S$  then set  $status(m) = \text{dead}$ 
7     else set  $status(m) = \text{stop}$ 
8   else
9     set  $status(m) = \text{no-stop}$ 
10  end
11 end
12 let  $BSD_b(N) = (Q, \delta, Q_{BSD_b(N)}, \Sigma^{in}, \Sigma^{out}, \lambda)$  where
    $Q = \{Q_{BSD_b(N)}, \mathbb{U}, Q_\emptyset\}$ ,  $\delta = \emptyset$ ,  $Q_{BSD_b(N)} = closure(q_S)$ ,  $\Sigma^{in} = I_N$ ,
    $\Sigma^{out} = O_N$ , and  $\lambda(Q_{BSD_b(N)}) = 3$ ,  $\lambda(\mathbb{U}) = 0$ ,  $\lambda(Q_\emptyset) = 4$ 
13 if  $\exists m \in Q_{BSD_b(N)} : status(m) = \text{stop}$  then set  $\lambda(Q_{BSD_b(N)}) = 2$ 
14 if  $\exists m \in Q_{BSD_b(N)} : status(m) = \text{dead}$  then set  $\lambda(Q_{BSD_b(N)}) = 1$ 
15 push  $Q_{BSD_b(N)}$  onto empty Stack
16 repeat
17   pop  $Q$  from Stack
18   foreach  $x \in \Sigma$  do
19     set  $Q' = \emptyset$  and  $\lambda(Q') = 4$ 
20     foreach  $m \in Q$  do
21       if  $\exists m'$  with  $m \xrightarrow{x} m'$  in  $S$  then
22         if  $\mathbb{U} \in closure(m')$  then
23           add transition  $(Q, x, \mathbb{U})$  to  $\delta$ 
24           continue with next outer foreach loop
25         else
26           add  $closure(m')$  to  $Q'$ 
27           if  $\exists m'' \in closure(m') : status(m'') = \text{stop}$  then set
              $\lambda(Q') = 2$ 
28           if  $\exists m'' \in closure(m') : status(m'') = \text{dead}$  then set
              $\lambda(Q') = 1$ 
29         end
30       end
31     end
32     if  $Q' \neq \emptyset$  and  $\lambda(Q') = 4$  then set  $\lambda(Q') = 3$ 
33     if  $Q' \notin Q$  then add  $Q'$  to  $Q$  and push  $Q'$  on Stack
34     add transition  $(Q, x, Q')$  to  $\delta$ 
35   end
36 until Stack is empty

```

Algorithmus 2 : Computing $BSD_b(N)$ from N .

```

Input : LTS  $BSD_b(N_1)$  and LTS  $BSD_b(N_2)$ 
Output : true or false
1 if  $\lambda_{BSD_b(N_1)}(Q_{BSD_b(N_1)}) + \lambda_{BSD_b(N_2)}(Q_{BSD_b(N_2)}) \leq 3$  then
2   | return false
3 end
4 mark  $(Q_{BSD_b(N_1)}, Q_{BSD_b(N_2)})$  as visited and push on empty Stack
5 repeat
6   | pop pair  $(P, Q)$  from Stack
7   | foreach  $x$  in  $\Sigma$  do
8     | let  $P \xrightarrow{x} P'$  in  $BSD_b(N_1)$  and  $Q \xrightarrow{x} Q'$  in  $BSD_b(N_2)$ 
9     | if pair  $(P', Q')$  was not visited then
10    |   | if  $\lambda_{BSD_b(N_1)}(P') + \lambda_{BSD_b(N_2)}(Q') \leq 3$  then
11    |   |   | return false
12    |   |   | end
13    |   |   | mark  $(P', Q')$  as visited and push on Stack
14    |   |   | end
15    |   |   | end
16 until Stack is empty
17 return true

```

Algorithmus 3 : Deciding b -responsiveness using BSD_b .

Proposition 120 [complexity of deciding b -responsiveness with BSD_b]

Let N_1 and N_2 be two composable open nets such that $N_1 \oplus N_2$ is a closed net. Let n_i be the number of reachable b -bounded markings in $env(N_i)$, for $i \in \{1, 2\}$. Then, we can decide whether N_1 is a b -partner of N_2 in $O(n_1^2 \cdot 2^{n_1}) + O(n_2^2 \cdot 2^{n_2}) + O(2^{n_1+n_2})$.

We proceed with a short complexity analysis of the decision procedure in Thm. 117. Theorem 117 induces an algorithm for deciding whether an open net $Impl$ b -conforms to an interface-equivalent open net $Spec$, which we already illustrated in Fig. 59: First, we compute $CSD_b(Impl)$ and $CSD_b(Spec)$. Second, we check if $CSD_b(Impl)$ and $CSD_b(Spec)$ are bisimilar and if the labels of related states satisfy the condition in Thm. 117. This decision algorithm is similar to the algorithm to decide b -responsiveness in Fig. 72 except that we use CSD_b instead of BSD_b and check the state labels differently.

Let I and O be fixed with $s = |I \uplus O|$ and let $N = (P, T, F, m_N, \Omega, I, O)$ be an open net. Let n be the number of reachable b -bounded markings in $env(N)$. We already showed that we can compute $BSD_b(N)$ with $O(2^n)$ states and $O(2^n)$ transitions in time $O(n^2 \cdot 2^n)$. To compute $CSD_b(N)$ from $BSD_b(N)$ according to Def. 104, we iteratively remove states from $BSD_b(N)$ until we reach a fixed point (i.e., the LTS $CSD_b(N)$). Algorithm 4 lists an algorithm for computing $CSD_b(N)$, which is, in essence, an inverse breadth-first-search. Checking whether we have to remove a state in Alg. 4 (lines 4 and 14) can be done in $O(s) = O(1)$ and we have to remove at most $O(2^n)$ states from $BSD_b(N)$. Therefore, the worst case complexity of Alg. 4 is $O(2^n)$. By combining Alg. 2 and Alg. 4, we can compute $CSD_b(N)$ (and $MP_b(N)$ by Def. 104) from N in time proportional to $O(n^2 \cdot 2^n) + O(2^n) = O(n^2 \cdot 2^n)$.

By Thm. 117, deciding b -conformance for two interface-equivalent open nets $Impl$ and $Spec$ boils down to computing the least bisimulation relation between $CSD_b(Impl)$ and $CSD_b(Spec)$ and check the related state labels. Algorithm 5 lists the corresponding algorithm, which is a minor modification

```

Input : LTS  $BSD_b(N) = (Q, \delta, Q_{BSD_b(N)}, \Sigma^{in}, \Sigma^{out}, \lambda)$ 
Output : LTS  $CSD_b(N)$ 

1 let Queue be empty
2 foreach  $(Q, x, \mathbb{U}) \in \delta$  do
3   if  $(Q$  is not in Queue) then
4     if  $(\lambda(Q) = 1$  and  $\forall y \in \Sigma^{in} : Q \xrightarrow{y} \mathbb{U})$  or  $(x \in \Sigma^{out})$  then
5       enqueue  $Q$  in Queue
6     end
7   end
8 end
9 repeat
10  dequeue  $Q'$  from Queue
11  foreach  $(Q, x, Q') \in \delta$  do
12    if  $Q$  is not in Queue then
13      remove  $(Q, x, Q')$  from  $\delta$  and add  $(Q, x, \mathbb{U})$  to  $\delta$ 
14      if  $(\lambda(Q) = 1$  and  $\forall y \in \Sigma^{in} : Q \xrightarrow{y} \mathbb{U})$  or  $(x \in \Sigma^{out})$  then
15        enqueue  $Q$  in Queue
16      end
17    end
18  end
19 until Queue is empty
20 remove unreachable states, transitions, and labels
21 return  $(Q, \delta, Q_{BSD_b(N)}, \Sigma^{in}, \Sigma^{out}, \lambda)$ 

```

Algorithmus 4 : Computing the LTS $CSD_b(N)$ from the LTS $BSD_b(N)$.

of Alg. 3: We adjust lines 1 and 10 to the condition in Thm. 117. Let n_1 be the number of reachable b -bounded markings in $env(Impl)$ and let n_2 be the number of reachable b -bounded markings in $env(Spec)$. Then, the algorithm in Alg. 5 has, like Alg. 3, a worst case complexity of $O(2^{n_1+n_2})$.

We can decide b -conformance for two given interface-equivalent open nets $Impl$ and $Spec$ by combining Alg. 2, Alg. 4, and Alg. 5. Let n_1 be the number of reachable b -bounded markings in $env(Impl)$ and let n_2 be the number of reachable b -bounded markings in $env(Spec)$. Then, we can decide b -conformance in $O(n_1^2 \cdot 2^{n_1}) + O(n_2^2 \cdot 2^{n_2}) + O(2^{n_1+n_2})$. Figure 73 illustrates the complexity of the parts of the decision algorithm.

Proposition 121 [complexity of deciding b -conformance with CSD_b]

Let $Impl$ and $Spec$ be two interface-equivalent open nets. Let n_1 be the number of reachable b -bounded markings in $env(Impl)$ and let n_2 be the number of reachable b -bounded markings in $env(Spec)$. Then, we can decide whether $Impl$ b -conforms to $Spec$ in $O(n_1^2 \cdot 2^{n_1}) + O(n_2^2 \cdot 2^{n_2}) + O(2^{n_1+n_2})$.

In the following section, we motivate and elaborate an alternative decision procedure for b -partner and b -conformance.

5.3 AN ALTERNATIVE DECISION PROCEDURE FOR b -CONFORMANCE

The decision procedure for b -conformance that we illustrated in Fig. 59 always requires to compute the LTS CSD_b for both given open nets $Impl$ and $Spec$. In this section, we elaborate on a decision procedure for b -conformance

```

Input : LTS  $CSD_b(Impl)$  and LTS  $CSD_b(Spec)$ 
Output : true or false
1 if  $\lambda_{CSD_b(Impl)}(Q_{CSD_b(Impl)}) < \lambda_{CSD_b(Spec)}(Q_{CSD_b(Spec)})$  then
2   | return false
3 end
4 mark ( $Q_{CSD_b(Impl)}, Q_{CSD_b(Spec)}$ ) as visited and push on empty Stack
5 repeat
6   | pop pair ( $P, Q$ ) from Stack
7   | foreach  $x$  in  $\Sigma$  do
8     | let  $P \xrightarrow{x} P'$  in  $CSD_b(Impl)$  and  $Q \xrightarrow{x} Q'$  in  $CSD_b(Spec)$ 
9     | if pair ( $P', Q'$ ) was not visited then
10    |   | if  $\lambda_{CSD_b(Impl)}(P') < \lambda_{CSD_b(Spec)}(Q')$  then
11    |   |   | return false
12    |   |   | end
13    |   |   | mark ( $P', Q'$ ) as visited and push on Stack
14    |   |   | end
15    |   | end
16 until Stack is empty
17 return true

```

Algorithmus 5 : Deciding b -conformance using CSD_b . We highlight the difference to Alg. 3.

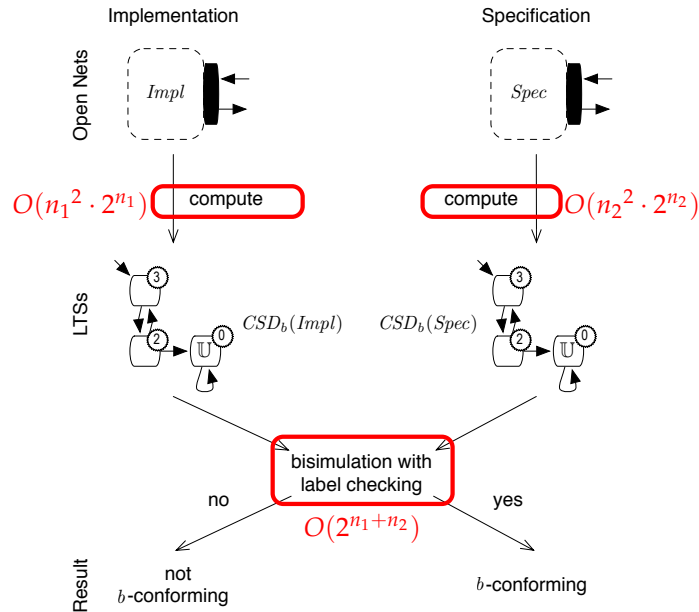


Figure 73: Using CSD_b to decide if an open net *Impl* b -conforms to an interface-equivalent open net *Spec*. We highlighted the complexity of each part of the decision algorithm.

that requires to compute only one LTS from *Spec*, but no LTS from *Impl*. The alternative decision procedure checks b -conformance, with a method called *matching*, directly on the state-space of *Impl* and the LTS that we computed from *Spec*. Regarding worst case complexity, matching is slightly more expensive than computing the bisimulation with label checking in Fig. 59 but

avoids computing $CSD_b(Impl)$ and operates directly on the state-space of $Impl$. This state-space is in general much smaller than $CSD_b(Impl)$.

In Sect. 5.3.1, we first demonstrate the idea of matching by elaborating an alternative decision procedure for b -responsiveness. Then, we extend this decision procedure for b -responsiveness to an alternative decision procedure for b -conformance in Sect. 5.3.2.

5.3.1 Deciding b -responsiveness using matching

In Thm. 102, we showed that we can decide whether two given open nets N and C are b -partners by comparing their LTS $BSD_b(N)$ and $BSD_b(C)$. The idea for an alternative decision procedure is to compute BSD_b (or, more precisely, an artifact derived from BSD_b) only for one open net instead of for both. This artifact shall be the LTS $MP_b(N)$ from Def. 104: We already established in Lem. 108 a relation between $MP_b(N)$ and the inner of a composable open net C of N . In addition, $MP_b(N)$ represents (parts of) the b -coverable *stopdead*-semantics of N by Cor. 116, which, in turn, over-approximates the b -bounded *stopdead*-semantics of N by Cor. 95. Figure 74 illustrates this idea.

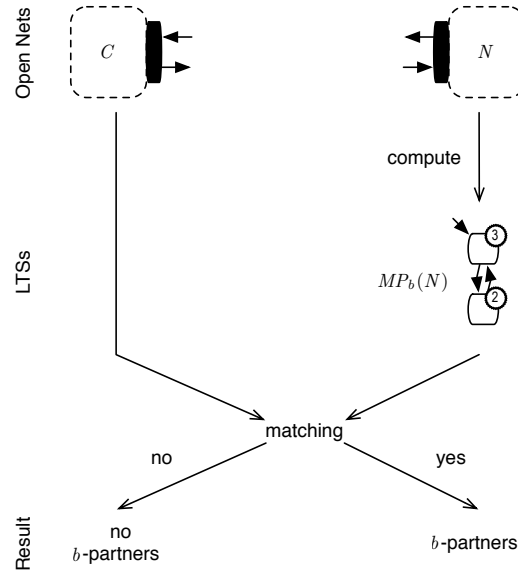


Figure 74: Using matching to decide if two given open nets N and C are b -partners.

For deciding whether C is a b -partner of N , we shall introduce a weak simulation relation with additional requirements, called *matching*, between (the inner of) C and $MP_b(N)$. The idea of matching exploits that $MP_b(N)$ is an over-approximation of the b -bounded *stopdead*-semantics of a b -partner C :

Lemma 122 [MP_b over-approximates b -bounded *stopdead*-semantics]

Let N and C be two composable open nets. If N and C are b -partners, then

- $bound_b(C) \subseteq co-uL_b(N)$,
- $dead_b(C) \subseteq co-ustop_b(N)$,
- $stop_b(C) \subseteq co-udead_b(N)$, and
- $L_b(C) \subseteq co-uncov_b(N)$.

Proof. We have $bound_b(C) \subseteq dead_b(C) \subseteq stop_b(C) \subseteq L_b(C) \subseteq co-uncov_b(N)$ by Def. 84 and Def. 93. Then the inclusions follow from Prop. 88. \square

The four languages $co-uL_b(N)$, $co-ustop_b(N)$, $co-udead_b(N)$, and $co-uncov_b(N)$ in Lem. 122 are represented in $MP_b(N)$ by Cor. 116(7–10). However, these languages do not always correspond to the b -bounded *stopdead*-semantics of a b -partner of N ; they are only over-approximations. Therefore, we have to exclude some of the open nets C whose b -bounded *stopdead*-semantics is characterized by $MP_b(N)$: The first inclusion in Lem. 122 implies that every $bound_b$ -violation of C (and, therefore, of $inner(C)$) must reach the empty state of $MP_b(N)$, as $co-uL_b(N)$ is solely represented by $L_4(MP_b(N))$. The second inclusion in Lem. 122 restricts the *dead*-traces of C : A *dead*-trace w must reach a state Q of $MP_b(N)$ such that $\lambda(Q) \neq 1$ and $\lambda(Q) \neq 2$. There must exist an outgoing x -labeled transition of Q with $x \in O_N$ and C must be able to receive that message x ; otherwise, N and C cannot “progress”. The third inclusion in Lem. 122 restricts the *stop*-traces of C in a similar way as the second inclusion, but additionally allows for mutual termination by reaching a final marking. In other words, whenever C internally stops (i.e., reaches a stop except for inputs in $inner(C)$), then either C must reach a marking from which it can receive a message that was sent by N or N and C internally reach a final marking. In other words, The last inclusion in Lem. 122 implies that $env(C)$ (and, therefore, $inner(C)$) must be weakly simulated by $L(MP_b(N))$.

Definition 123 [matching]

Let N be an open net such that $MP_b(N)$ exists. An open net C *matches* with $MP_b(N)$ if

1. $I_C = \Sigma^{out}$ and $O_C = \Sigma^{in}$, and
2. $RG(inner(C))$ is weakly simulated by $MP_b(N)$ with the least weak simulation relation ϱ such that for all $(m, Q) \in \varrho$:
 - a) If m is not b -bounded in $inner(C)$, then $Q = Q_\emptyset$.
 - b) If m is a stop except for inputs in $inner(C)$, then for all $m_Q \in Q$:
 - i. there exists a final marking m'_Q of $env(N)$ such that $m_Q \xrightarrow{\varepsilon} m'_Q$ in $env(N)$, or
 - ii. there exists an $x \in O_N$ such that $m \xrightarrow{x}$ in $inner(C)$ and $m_Q \xrightarrow{x}$ in $env(N)$.
 - c) If m is dead except for inputs in $inner(C)$, then for all $m_Q \in Q$, there exists an $x \in O_N$ such that $m \xrightarrow{x}$ in $inner(C)$ and $m_Q \xrightarrow{x}$ in $env(N)$.

We refer to ϱ as the *matching relation*.

Example 124 The open net U in Fig. 54b matches with $MP_1(D)$ of the open net D in Fig. 54a: The reachability graph of the inner net of U (depicted in Fig. 68c) is weakly simulated by $MP_1(D)$ (depicted in Fig. 68b) with the weak simulation relation

$$\varrho = \{([p_4], Q_0), ([p_3], Q_2)\},$$

which we already detailed in Ex. 109. Every marking of $inner(U)$ is b -bounded, thus Def. 123(2a) holds trivially. The only final marking of

$inner(U)$ is $[\]$, thus $[p_3]$ is dead except for inputs in $inner(U)$. Nevertheless, we have $d \in O_D$ such that $[p_3] \xrightarrow{d} in inner(U)$ and $m \xrightarrow{d} in env(D)$ for every $m \in Q_2$, thus Def. 123(2) holds and U matches with D . \diamond

With the next theorem, we show that matching gives a necessary and sufficient condition for deciding whether an open net C is a b -partner of an open net N . For the proof, we frequently employ that, for all open nets N , the b -bounded *stopdead*-semantics of $inner(N)$ is included in the b -bounded *stopdead*-semantics of $env(N)$. Recall that the b -bounded *stopdead*-semantics of $inner(N)$ is well-defined by Def. 84, as $inner(N)$ is a labeled net. Therefore, we directly conclude the following corollary from Def. 15, Def. 17, and Def. 84.

Corollary 125

For an open net N , we have

- $bound_b(inner(N)) \subseteq bound_b(N)$,
- $L_b(inner(N)) \subseteq L_b(N)$,
- $stop_b(inner(N)) \subseteq stop_b(N)$, and
- $dead_b(inner(N)) \subseteq dead_b(N)$.

Theorem 126 [characterizing all b -partners]

Let N be an open net such that $MP_b(N)$ exists. Then an open net C matches with $MP_b(N)$ iff C is a b -partner of N .

Proof. \Rightarrow : By Def. 123(1), N and C are composable and $N \oplus C$ is a closed net. Let $m = m_{|env(N)} + m_{|inner(C)}$ be a reachable marking in $N \oplus C$, and let ϱ denote the matching relation. Then there exists a state Q of $MP_b(N)$ such that $(m_{|inner(C)}, Q) \in \varrho$ and $m_{|env(N)} \in Q$ by Lem. 108. It remains to show that m is b -bounded and responsive.

Each state of $MP_b(N)$ is a (possibly empty) set of b -bounded markings of $env(N)$ by Def. 104, thus $m_{|env(N)} \in Q$ is b -bounded in $env(N)$. Assume $m_{|inner(C)}$ is not b -bounded in $inner(C)$. Then $Q = Q_\emptyset$ by Def. 123(2a), which contradicts $m_{|env(N)} \in Q$. Thus, $m_{|inner(C)}$ is b -bounded in $inner(C)$ and m is b -bounded in $N \oplus C$.

We show by Noetherian induction on the number of tokens on O_N that m is responsive by distinguishing all possible cases in depth:

1. If $m_{|inner(C)}$ is not a stop except for inputs in $inner(C)$:
Then m is trivially responsive in $N \oplus C$ by Def. 15 and Def. 41.
2. If $m_{|inner(C)}$ is a stop except for inputs in $inner(C)$:
 - a) Assume that $m_{|inner(C)}$ is not dead except for inputs in $inner(C)$ and there exists a final marking m_1 of $env(N)$ such that $m_{|env(N)} \xrightarrow{\varepsilon} m_1$ in $env(N)$:
Because $m_{|inner(C)}$ is not dead except for inputs in $inner(C)$, there exists a final marking m_2 of $inner(C)$ such that $m_{|inner(C)} \xrightarrow{\varepsilon} m_2$ in $inner(C)$ by Def. 84. Then $m_1 + m_2$ is a final marking of $N \oplus C$ and $m_1 + m_2$ is reachable from m in $N \oplus C$, thus m is responsive by Def. 41.

- b) If $m|_{inner(C)}$ is dead except for inputs in $inner(C)$ or there does not exist a final marking m_1 of $env(N)$ such that $m|_{env(N)} \xrightarrow{\varepsilon} m_1$ in $env(N)$:

In either case, there exists an $x \in O_N$ and a marking m' of $N \oplus C$ such that $m|_{inner(C)} \xrightarrow{x} m'|_{inner(C)}$ in $inner(C)$ and $m|_{env(N)} \xrightarrow{x} m'|_{env(N)}$ in $env(N)$ by Def. 123(2b) and Def. 123(2c). These can be combined to show $m \xrightarrow{\varepsilon} m'$ in $N \oplus C$. Either, a token was put onto O_N along $m|_{env(N)} \xrightarrow{x} m'|_{env(N)}$ and we are done, or a token is removed from x and we are done by induction.

We showed that every reachable marking m of $N \oplus C$ is b -bounded and responsive, thus C is a b -partner of N .

\Leftarrow : C is a b -partner of N , thus Def. 123(1) holds trivially. With Lem. 122 and Cor. 116(7), we have $L_b(C) \subseteq L(MP_b(N))$ and hence the weak simulation ρ of $RG(inner(C))$ by $MP_b(N)$ exists; ρ is uniquely defined because $MP_b(N)$ is deterministic. For the rest of the proof, let $(m, Q) \in \rho$. We show that items (2a), (2b), and (2c) of Def. 123 hold. Let $m|_{inner(C)} \xrightarrow{w} m$ in $inner(C)$.

- Assume m is not b -bounded in $inner(C)$, thus $w \in bound_b(C)$ by Def. 84 and Cor. 125. Then $w \in co-uL_b(N)$ by Lem. 122 and $Q = Q_\emptyset$ by Cor. 116(10), which implies Def. 123(2a).
- Assume m is a stop except for inputs in $inner(C)$, thus $w \in stop_b(C) \subseteq co-udead_b(N)$ by Cor. 125 and Lem. 122, and $\lambda(Q) = 2$, $\lambda(Q) = 3$, or $\lambda(Q) = 4$ by Cor. 116(8). Assume there exists a marking $m_Q \in Q$ of $env(N)$ violating Def. 123(2b).
 1. In the case m_Q is a stop except for inputs in $env(N)$:
Then m_Q is dead except for inputs in $env(N)$ as Def. 123(2bi) is violated, thus $\lambda(Q) = 1$ by Def. 99, a contradiction.
 2. In the case m_Q is not a stop except for inputs in $env(N)$:
Then there exists an $x \in O_N$ such that $m_Q \xrightarrow{x}$ in $env(N)$, but for all $x \in O_N$ with $m_Q \xrightarrow{x}$ in $env(N)$, $m \not\xrightarrow{x}$ in $inner(C)$ as Def. 123(2bii) is violated. Then there exists a word $w' \in \{x \in O_N \mid m_Q \xrightarrow{x} \text{ in } env(N)\}^+$ such that $ww' \in stop_b(N)$ (because the number of tokens on former output places of N in m_Q is bounded and the firing of an x -transition in $env(N)$ does not enable other transitions) and $ww' \in dead_b(C)$ (because m is a stop except for inputs, $env(C)$ cannot use/remove the tokens produced along w' since $m \not\xrightarrow{x}$ in $inner(C)$, and all final markings of $env(C)$ have empty interface places). This contradicts Prop. 88, thus no marking $m_Q \in Q$ violates Def. 123(2b).
- Assume m is dead except for inputs in $inner(C)$, thus $w \in dead_b(C) \subseteq co-ustop_b(N)$ by Cor. 125 and Lem. 122, and $\lambda(Q) = 3$ or $\lambda(Q) = 4$ by Cor. 116(9). Thus, any $m_Q \in Q$ is not a stop except for inputs in $env(N)$, and we are done by the previous paragraph. \square

Example 127 We already showed in Ex. 124 that the open net U in Fig. 54b matches with $MP_1(D)$ of the open net D in Fig. 54a. Thus, by Thm. 126, U is a 1-partner of D , which we already detailed in Ex. 89. \diamond

5.3.2 Deciding b -conformance using matching

In this section, we extend the decision procedure for b -responsiveness from the previous section to a decision procedure for b -conformance. To this end, we develop a finite characterization of all b -conforming open nets. For characterizing all open nets that b -conform to an open net N , we introduce the notion of a maximal b -partner $\max_b(N)$ of N . We later show that every b -partner of $\max_b(N)$ b -conforms to N . Thus, matching with $MP_b(\max_b(N))$ characterizes all open nets that b -conform to N . Finally, we show how $\max_b(N)$ can be constructed from $MP_b(N)$.

Intuitively, a b -partner M of N is maximal if the trace sets of M 's b -bounded *stopdead*-semantics are maximal with respect to the trace-sets of all b -partners of N .

Definition 128 [maximal b -partner]

Let $X \in \{\text{bound}_b, \text{dead}_b, \text{stop}_b, L_b\}$ be a trace set of the b -bounded *stopdead*-semantics from Def. 84. A b -partner M of an open net N is X -maximal, if for all b -partners C of N : $X(C) \subseteq X(M)$. A b -partner M is maximal if M is X -maximal for all $X \in \{\text{bound}_b, \text{dead}_b, \text{stop}_b, L_b\}$.

In [258, 226], an L_b -maximal b -partner of N is called “most-permissive”, as it allows for the most behavior of all b -partners of N .

A maximal b -partner M of an open net N characterizes all open nets that b -conform to N ; that is, every b -partner of M b -conforms to N and every open net that b -conforms to N is a b -partner of M .

Theorem 129 [characterizing all b -conforming open nets]

Let M be a maximal b -partner of an open net $Spec$. Then for every open net $Impl$, $Impl$ is a b -partner of M if and only if $Impl$ b -conforms to $Spec$.

Proof. \Rightarrow : As $Impl$ is a b -partner of M and M is a b -partner of $Spec$, $Impl$ and $Spec$ are interface-equivalent by Def. 44. We show that $Impl$ b -conforms to $Spec$ by showing the inclusions of their b -coverable *stopdead*-semantics according to Thm. 97.

- Let $w \in \text{uncov}_b(Impl)$. Then $w \notin L_b(M)$ because M is a b -partner of $Impl$. As M is L_b -maximal, there does not exist a b -partner C of $Spec$ with $w \in L_b(C)$. Thus, $w \in \text{uncov}_b(Spec)$.
- Let $w \in \text{udead}_b(Impl) = \text{dead}(Impl) \cup \text{uncov}_b(Impl)$ by Def. 93. If $w \in \text{uncov}_b(Impl)$, then $w \in \text{uncov}_b(Spec) \subseteq \text{udead}_b(Spec)$ by the first item. Thus, we assume $w \in \text{dead}(Impl)$. Then $w \notin \text{stop}_b(M)$ by Prop. 88. As M is stop_b -maximal, there does not exist a b -partner C of $Spec$ with $w \in \text{stop}_b(C)$. Then either $w \notin L(Spec)$ or $w \in \text{udead}_b(Spec)$ by Lem. 96(3). If $w \in \text{udead}_b(Spec)$, we are done. Otherwise, $w \notin uL_b(Spec)$ because $w \notin \text{uncov}_b(Spec)$, and there exists a b -partner C of $Spec$ with $w \in \text{bound}_b(C)$ by Lem. 96(1) and $w \in \text{stop}_b(C)$. This contradicts that M is stop_b -maximal, thus $w \in \text{udead}_b(Spec)$.
- Let $w \in \text{ustop}_b(Impl) = \text{stop}(Impl) \cup \text{uncov}_b(Impl)$ by Def. 93. If $w \in \text{uncov}_b(Impl)$, then $w \in \text{uncov}_b(Spec) \subseteq \text{ustop}_b(Spec)$ by the first item. Thus, we assume $w \in \text{stop}(Impl)$. Then $w \notin \text{dead}_b(M)$ by Prop. 88. As M is dead_b -maximal, there does not exist a b -partner C of $Spec$ with $w \in \text{dead}_b(C)$. Then either $w \notin L(Spec)$ or $w \in \text{ustop}_b(Spec)$ by Lem. 96(2). If $w \in \text{ustop}_b(Spec)$, we are done. Otherwise, $w \notin uL_b(Spec)$ because

$w \notin \text{uncov}_b(\text{Spec})$, and there exists a b -partner C of Spec with $w \in \text{bound}_b(C)$ by Lem. 96(1) and $w \in \text{dead}_b(C)$. This contradicts that M is dead_b -maximal, thus $w \in \text{ustop}_b(\text{Spec})$.

- Let $w \in uL_b(\text{Impl}) = L(\text{Impl}) \cup \text{uncov}_b(\text{Impl})$ by Def. 93. If $w \in \text{uncov}_b(\text{Impl})$, then $w \in \text{uncov}_b(\text{Spec}) \subseteq uL_b(\text{Spec})$ by the first item. Thus, we assume $w \in L(\text{Impl})$. Then $w \notin \text{bound}_b(M)$ by Prop. 88. As M is bound_b -maximal, there does not exist a b -partner C of Spec with $w \in \text{bound}_b(C)$. Then $w \in uL_b(\text{Spec})$ by Lem. 96(1).

\Leftarrow : As Impl b -conforms to Spec and M is a b -partner of Spec , M is a b -partner of Impl by Def. 47 and Impl is a b -partner of M by Def. 44. \square

In the rest of this section, we construct a maximal b -partner of an open net N . As the starting point, we take the b -partner $mp_b(N)$ from Def. 110, which is already bound_b -maximal and L_b -maximal.

Lemma 130 [mp_b is bound_b -maximal and L_b -maximal]

Let N be an open net such that $MP_b(N)$ exists. Then $mp_b(N)$ is a bound_b -maximal and L_b -maximal b -partner of N .

Proof. As $MP_b(N)$ exists, the open net $mp_b(N)$ is a b -partner of N by Lem. 112. In addition, the state Q_\emptyset exists in $MP_b(N)$ by the construction of $MP_b(N)$. Let C be a b -partner of N . Then

- $\text{bound}_b(C) \subseteq \text{co-}uL_b(N) = L_4(MP_b(N)) \subseteq \text{bound}_b(mp_b(N))$:
The first inclusion holds by Prop. 88 and $\text{bound}_b(C) \cap \text{uncov}_b(N) = \emptyset$, the equation by Cor. 116(10), and the second inclusion holds by the construction of $mp_b(N)$ in Def. 110: The open net $mp_b(N)$ has at least one output place o and an internal transition t such that $\bullet t = \{Q_\emptyset\}$ and $t^\bullet = \{Q_\emptyset, o\}$. In other words, once the place Q_\emptyset is marked, t may produce an unlimited number of tokens on o . Therefore, every trace w to state Q_\emptyset in $MP_b(N)$ (i.e., $w \in L_4(MP_b(N))$) is a bound_b -violation of $mp_b(N)$. Thus, $mp_b(N)$ is a bound_b -maximal b -partner of N .
- $L_b(C) \subseteq \text{co-uncov}_b(N) = L(MP_b(N)) \subseteq L_b(mp_b(N))$:
The first inclusion holds by Def. 93. By Cor. 116(7) and the construction of $MP_b(N)$ in Def. 104, we have $\text{co-uncov}_b(N) = L(\text{CSD}_b(N)) \setminus L_0(\text{CSD}_b(N)) = L(MP_b(N))$. The second inclusion holds by the construction of $mp_b(N)$ from $MP_b(N)$ in Def. 110 and by Cor. 125. Thus, $mp_b(N)$ is an L_b -maximal b -partner of N . \square

Because $mp_b(N)$ is already bound_b - and L_b -maximal, we slightly modify the construction of $mp_b(N)$ from $MP_b(N)$ to get a maximal b -partner $max_b(N)$ of N . The idea for the construction of $max_b(N)$ is to shift every non- dead -trace w of N to the dead -traces of $max_b(N)$: If $m_{mp_b(N)} \xrightarrow{w} [Q]$ in $\text{env}(mp_b(N))$ (recall that every state of $MP_b(N)$ is a place of $mp_b(N)$), we introduce an internal transition that shifts the token from Q to a new place Q' such that every output transition enabled at $[Q]$ is not enabled at $[Q']$. In addition, if w is also a stop -trace of N , we add $[Q']$ to the final markings of $max_b(N)$; that way, w is not a dead - but a stop -trace of $max_b(N)$.

Definition 131 [open net max_b]

Let N be an open net such that $MP_b(N)$ exists. We modify the induced labeled net of $MP_b(N)$ (see Def. 110) as follows: For every place Q where

1. $\lambda(Q) \neq 1$, and
2. there exists an $x \in I_N$ with $Q \xrightarrow{x}$ in $MP_b(N)$,

we add a fresh place Q' and a fresh τ -labeled transition t such that $\bullet t = \{Q\}$ and $t^\bullet = \{Q'\}$. We add $[Q']$ to the final markings if $\lambda(Q) = 2$. In addition, for every x -labeled transition $t \in Q^\bullet$ with $x \in O_N$, we add a fresh transition t' such that $\bullet t' = \{Q'\}$ and $t'^\bullet = t^\bullet$.

We define $max_b(N)$ as the open net whose inner net is the modified labeled net induced by $MP_b(N)$.

Example 132 Figure 75 illustrates the construction in Def. 131. The left part of Fig. 75 sketches a part of $MP_b(N)$ for an open net N : That part of $MP_b(N)$ has three states Q, R, S ; an o -labeled transition from Q to R ; and an i -labeled transition from Q to S . We have $o \in I_N$ and $i \in O_N$, and $\lambda(Q) = 2$, $\lambda(R) = 1$, and $\lambda(S) = 1$. The right part of Fig. 75 sketches the resulting part of the inner net of $max_b(N)$: As in Def. 110, each state induces a place, each transition in $MP_b(N)$ induces a transition connecting two places in (the inner of) $max_b(N)$, and $o \in O_{max_b(N)}$ and $i \in I_{max_b(N)}$.

Because $\lambda(Q) \neq 1$ (i.e., no marking in Q is dead except for inputs in Q by Def. 99) and $Q \xrightarrow{o}$, we add the place Q' and the transitions t and t' . That way, we shift every trace w that reaches the state Q in $MP_b(N)$ (i.e., w is not a *dead-trace* of N) to the set of *dead-traces* of $max_b(N)$, thereby maximizing the set of *dead-traces* of $max_b(N)$.

By applying this construction only in the case of $\lambda(Q) \neq 1$, we guarantee that the shifted trace w is not a *dead-trace* of N . However, w may be a *stop-trace* of N . In this case, shifting w to the *dead-traces* of $max_b(N)$ fails to produce a b -partner by Prop. 88. If w could be a *stop-trace* of N , then $\lambda(Q) = 2$ like in our example in Fig. 75. Therefore, we add the final marking $[Q']$ to $max_b(N)$, which implies $w \in stop(max_b(N)) \setminus dead(max_b(N))$. In other words, we shift w to the set of *stop-traces* of $max_b(N)$ instead. \diamond

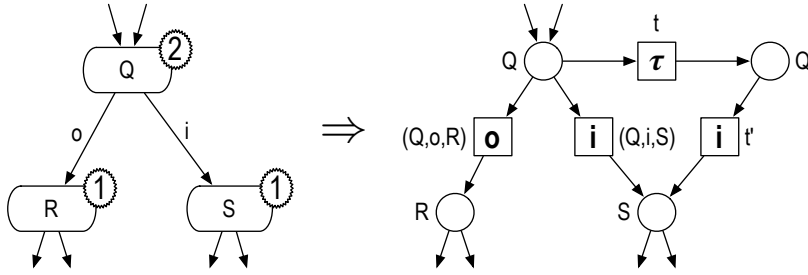


Figure 75: A sketch of the construction in Def. 131. We depict the label of each state as an encircled number in the upper right corner of that state. The marking $[Q']$ is a final marking of the resulting open net.

Note that the modification in Def. 131 applies to all states Q of $MP_b(N)$ that fulfill both requirements (i.e., $\lambda(Q) \neq 1$ and there exists an $x \in I_N$ with $Q \xrightarrow{x}$), including the empty state Q_\emptyset . Also note that the second requirement in Def. 131 is not compulsory to construct a maximal b -partner; it merely hinders the duplication of places for traces that are already *stop-traces* in $mp_b(N)$: Recall that every outgoing transition of a state Q in $MP_b(N)$ becomes a transition in $mp_b(N)$ by Def. 110. A transition of $mp_b(N)$ that derives from $Q \xrightarrow{x}$ with $x \in I_N = O_{mp_b(N)}$ has the output place x in its

postset. If no such transition exists for the place Q in $mp_b(N)$ (as required by Def. 131(2)), then every trace w of $mp_b(N)$ that marks place Q is trivially a *stop*-trace of $mp_b(N)$. Therefore, w is also a *stop*-trace of $max_b(N)$ even without the modified construction in Def. 131.

The next theorem shows that the construction in Def. 131 yields a maximal b -partner.

Theorem 133 [max_b is a maximal b -partner]

Let N be an open net such that $MP_b(N)$ exists. Then $max_b(N)$ is a maximal b -partner of N .

Proof. As $MP_b(N)$ exists, the open net $mp_b(N)$ is a b -partner of N by Lem. 112 and $mp_b(N)$ is $bound_b$ -maximal and L_b -maximal by Lem. 130. The construction of $max_b(N)$ in Def. 131 preserves every trace of $mp_b(N)$, and no additional trace or $bound_b$ -violation is introduced. Thus, we have $bound_b(mp_b(N)) = bound_b(max_b(N))$ and $L_b(mp_b(N)) = L_b(max_b(N))$. The construction of $max_b(N)$ ensures that

- every trace w of $max_b(N)$ is a *stop*-trace of $max_b(N)$ except w is a *dead*-trace of N (i.e., w leads to a state Q with $\lambda(Q) = 1$ in $MP_b(N)$ by Cor. 116(2)), and
- every *stop*-trace w of $max_b(N)$ is a *dead*-trace of $max_b(N)$ except w is a *stop*-trace of N (i.e., w leads to a state Q with $\lambda(Q) = 2$ in $MP_b(N)$ by Cor. 116(3))

Thus, $max_b(N)$ is a b -partner of N by Prop. 88, and $max_b(N)$ is even maximal by the construction of $MP_b(N)$. \square

Example 134 The open net $max_1(D)$ in Fig. 76a is a maximal b -partner of the open net D in Fig. 54a. We obtained $max_1(D)$ from $MP_1(D)$ in Fig. 70a according to Def. 131: place p_3 is induced by Q_\emptyset , place p_4 is induced by Q_0 , and place p_5 is induced by Q_2 . The nine transitions t_1 – t_9 derive from the nine transitions of $MP_1(D)$. The place p'_3 is a duplicate of p_3 because $\lambda(Q_\emptyset) \neq 1$ and $Q_\emptyset \xrightarrow{s}$ with $s \in I_D$ in $MP_1(D)$ —that is, Q_\emptyset fulfills the criterion in Def. 131. Likewise, we also added the three transitions t_{10} , t_{11} , and t_{12} .

For the state Q_2 in $MP_1(D)$, we have $\lambda(Q_2) \neq 1$ but there does not exist an $x \in I_D$ such that $Q_2 \xrightarrow{x}$ in $MP_1(D)$: The only two outgoing transitions of Q_2 are labeled with d and f , respectively, and we have $d, f \in O_D$. Thus, Q_2 does not fulfill the second requirement in Def. 131 and every trace of $mp_1(D)$ that marks the place p_5 (i.e., the place induced by Q_2) is already a *stop*-trace of $mp_1(D)$ in Fig. 70b. Therefore, w is also a *stop*-trace of $max_1(D)$.

Compared to the b -partner $mp_1(D)$ of D , $max_1(D)$ differs only in the place p'_3 and the transitions t_{10} , t_{11} , and t_{12} . We illustrate this difference in Fig. 76b. These four nodes add additional *stop*- and *dead*-traces to $max_1(D)$: For example, $d \notin stop_b(mp_1(D))$ because of the transitions t_3 and t_4 , but $d \in stop_b(max_1(D))$ and $d \in dead_b(max_1(D))$. \diamond

Finally, we combine Thm. 126, Thm. 129 and Thm. 133 to finitely characterize all b -conforming open nets for a given open net N .

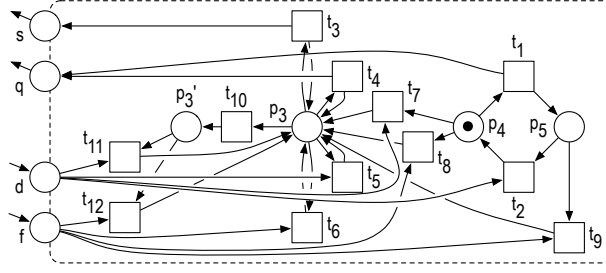
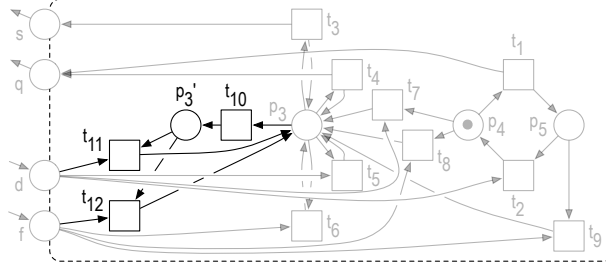
(a) Open net $max_1(D)$ (b) Difference of $max_1(D)$ to $mp_1(D)$ from Fig. 70b

Figure 76: The open net $max_1(D)$ that we obtained from the LTS $MP_1(D)$ in Fig. 70a according to Def. 131, and its difference to the open net $mp_1(D)$ from Fig. 70b. In addition to the figure, we have $\Omega_{max_1(D)} = \emptyset$.

Proposition 135 [characterizing all b -conforming open nets]

Let $Spec$ be an open net such that $MP_b(Spec)$ exists. For every open net $Impl$, $Impl$ matches with $MP_b(max_b(Spec))$ if and only if $Impl$ b -conforms to $Spec$.

Example 136 Figure 77a depicts the LTS $MP_1(max_1(D))$, which characterizes all open nets that b -conform to the open net D in Fig. 54a due to Prop. 135. We already claimed in Ex. 49 that the open net D' in Fig. 56 1-conforms to D . According to Prop. 135, we have to check whether D' matches with $max_1(D)$.

The reachability graph of the inner net of D' (depicted in Fig. 77b) is weakly simulated by $MP_1(max_1(D))$ with the weak simulation relation

$$q = \{([p_1], Q_0), ([p_2], Q_2), ([p_1], Q_3), ([], Q_\emptyset)\}.$$

Every marking of $inner(D')$ is b -bounded, thus Def. 123(1) holds trivially. The only final marking of $inner(D')$ is $[\]$, thus $[p_1]$ and $[\]$ are stops except for inputs and $[p_1]$ is dead except for inputs in $inner(D')$. For $[\]$, Def. 123(2) holds trivially. For $[p_1]$, we have $q \in O_{MP_1(max_1(D))}$ such that $[p_1] \xrightarrow{q}$ in $inner(D')$ and $m \xrightarrow{q}$ in $env(max_1(D))$ for every $m \in Q_0 \cup Q_3$, thus Def. 123(2) holds and D' matches with $MP_1(max_1(D))$ and D' 1-conforms to D . \diamond

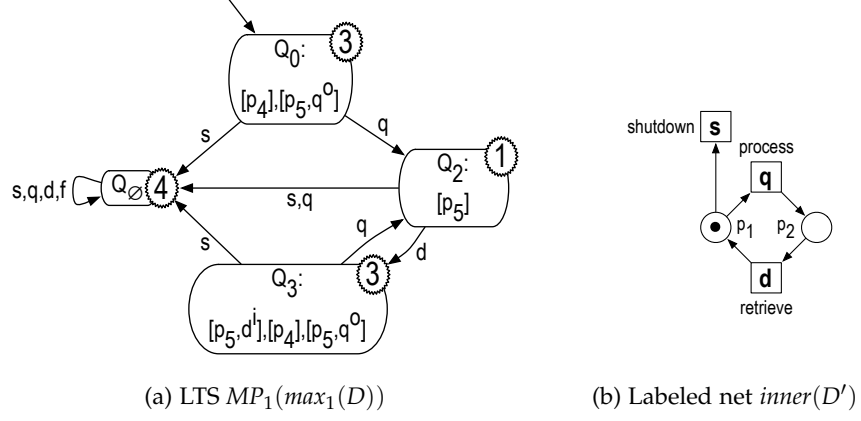


Figure 77: The LTS $MP_1(max_1(D))$ derived from $max_1(D)$ in Fig. 76a, and the inner net of the open net D' from Fig. 56. We depict the label of each state as an encircled number in the upper right corner of that state. In addition to the figures, we have $\Omega_{inner(D')} = \{\{\}\}$.

5.3.3 Analyzing the computational complexity

We complete this section with a short complexity analysis of the alternative decision procedures. Theorem 126 induces an algorithm for deciding whether two given open nets C and N are b -partners. Let n_1 be the number of reachable b -bounded markings in $inner(C)$ and let n_2 be the number of reachable b -bounded markings in $env(N)$. First, we compute $MP_b(N)$ in $O(n_2^2 \cdot 2^{n_2})$. Second, we check if C matches with $MP_b(N)$. We already illustrated this algorithm in Fig. 74.

Algorithm 6 lists an algorithm to check if C matches with $MP_b(N)$, which is a straight-forward implementation of Def. 123. Let I and O be fixed with $s = |I \uplus O|$ and let $N = (P, T, F, m_N, \Omega, I, O)$ be an open net. The construction of the LTS S in line 2 yields at most $O(n_1)$ states. As in Alg. 2, we can compute the *closure* sets in line 3 by applying the Floyd-Warshall-algorithm [95] to S with runtime $O(n_1^3)$. Every lookup in *closure* (e.g., lines 5 and 6) takes $O(n_1)$ time. Therefore, we can compute the status of each marking (lines 4–11) in $O(n_1^3)$. The LTS $MP_b(N)$ has $O(2^{n_2})$ states, as we already described in Sect. 5.2.3. Thus, the least weak simulation relation ρ of $RG(inner(C))$ (i.e., S) by $MP_b(N)$ consists of at most $O(n_1 \cdot 2^{n_2})$ pairs. We consider each pair (m, Q) at most once (lines 13–31). Notice that whenever $Q = Q_\emptyset$, Q will not change in the reachable pairs due to the self-loops of Q_\emptyset in $MP_b(N)$ and, thus, Def. 123(2) will be vacuously true. Therefore, we never consider a pair (m, Q_\emptyset) : For the initial pair $(m_{inner(C)}, Q_{MP_b(N)})$ in line 12, $Q_{MP_b(N)} \neq Q_\emptyset$ because of the initial marking of $env(N)$, and we never enqueue an unvisited pair (m', Q_\emptyset) in line 27. For each pair $(m, Q) \in \rho$ with $Q \neq Q_\emptyset$, we immediately abort if m is not b -bounded in $inner(C)$ (line 15) because of Def. 123(2a). If m is b -bounded in $inner(C)$ in turn, we have to check each marking $m_Q \in Q$ (lines 17–19 and lines 21–23) according to Def. 123(2b) and Def. 123(2c)—that is, at most $O(n_2)$ markings. Thereby, the checks in lines 18 and 22 can be done in constant time: We can already compute the reachability of a final marking from m_Q in $env(N)$ and the possible outputs of m_Q while computing $MP_b(N)$. In addition, the possible inputs of m in $inner(C)$ can be computed during the Floyd-Warshall-algorithm in line 3. Therefore, we can decide whether an open net C matches with $MP_b(N)$ in $O(n_1^3) + O(n_1 \cdot n_2 \cdot 2^{n_2})$.

```

Input : open net  $C$  and LTS  $MP_b(N) = (Q, \delta, Q_{MP_b(N)}, \Sigma^{in}, \Sigma^{out}, \lambda)$ 
Output : true or false

1 if  $I_C \neq \Sigma^{out}$  or  $O_C \neq \Sigma^{in}$  then return false
2 construct LTS  $S = (Q_S, \delta_S, q_S, I_C, O_C, \Omega_S)$  from  $RG(inner(C))$  but
   stop at each bound-violation
3 compute  $closure(m) = \{m' \mid m \xrightarrow{\varepsilon} m'\}$  in  $S$  for all  $m \in Q_S$ 
4 foreach  $m \in Q_S$  do
5   if  $\forall m' \in closure(m) : \forall x \in O_N : m' \not\xrightarrow{x}$  in  $S$  then
6     if  $\forall m' \in closure(m) : m' \notin \Omega_S$  then set  $status(m) = \text{dead}$ 
7     else set  $status(m) = \text{stop}$ 
8   else
9     set  $status(m) = \text{no-stop}$ 
10  end
11 end
12 mark  $(q_S, Q_{MP_b(N)})$  as visited and enqueue in empty Queue
13 repeat
14   dequeue  $(m, Q)$  from Queue
15   if  $m$  not  $b$ -bounded in  $inner(C)$  then return false
16   if  $status(m) = \text{stop}$  then
17     foreach  $m_Q \in Q$  do
18       if  $\forall m'_Q$  with  $m_Q \xrightarrow{\varepsilon} m'_Q$  in  $env(N) : m'_Q \notin \Omega_{env(N)}$  and
19          $\{x \in \Sigma^{out} \mid m \xrightarrow{x}$  in  $inner(C) \wedge m_Q \xrightarrow{x}$  in  $env(N)\} = \emptyset$ 
20         then return false
21     end
22   else if  $status(m) = \text{dead}$  then
23     foreach  $m_Q \in Q$  do
24       if  $\{x \in \Sigma^{out} \mid m \xrightarrow{x}$  in  $inner(C) \wedge m_Q \xrightarrow{x}$  in  $env(N)\} = \emptyset$ 
25       then return false
26     end
27   end
28   foreach  $m' \in Q_S$  and  $x \in \Sigma \cup \{\tau\}$  with  $m \xrightarrow{x} m'$  in  $S$  do
29     if  $x = \tau$  then let  $Q' = Q$  else let  $Q' : Q \xrightarrow{x} Q'$  in  $MP_b(N)$ 
30     if  $Q' \neq Q_\emptyset$  and  $(m', Q')$  not visited then
31       mark  $(m', Q')$  as visited and enqueue in Queue
32     end
33   end
34 until Queue is empty
35 return true

```

Algorithmus 6 : Deciding b -responsiveness using matching.

By combining Alg. 2, Alg. 4, and Alg. 6, we can decide whether two given open nets C and N are b -partners in $O(n_2^2 \cdot 2^{n_2}) + O(n_1^3) + O(n_1 \cdot n_2 \cdot 2^{n_2}) = O(n_1^3) + O((n_1 + n_2) \cdot n_2 \cdot 2^{n_2})$. Figure 78 also illustrates the complexity of the parts of the decision algorithm.

Proposition 137 [complexity of deciding b -responsiveness with matching]
 Let N_1 and N_2 be two composable open nets such that $N_1 \oplus N_2$ is a closed net. Let n_1 (n_2) be the number of reachable b -bounded markings in $inner(N_1)$ ($env(N_2)$). Then, we can decide whether N_1 is a b -partner of N_2 in $O(n_1^3) + O((n_1 + n_2) \cdot n_2 \cdot 2^{n_2})$.

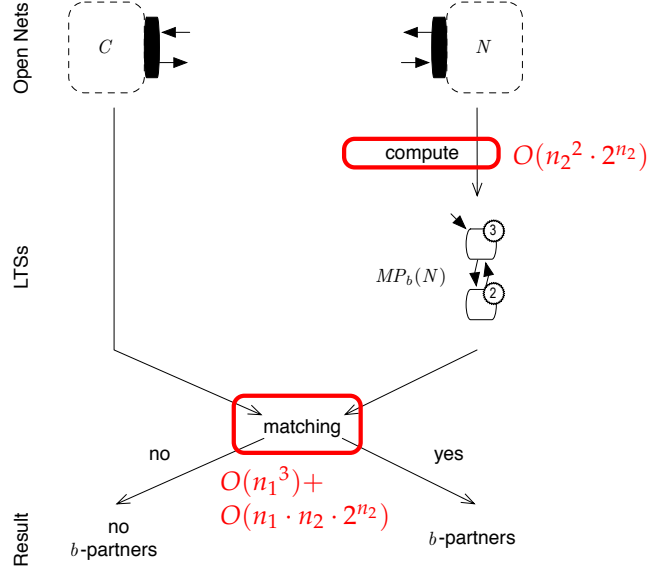


Figure 78: The complexity of the parts of the decision procedure illustrated in Fig. 74.

Deciding b -responsiveness with BSD_b takes $O(n_1^2 \cdot 2^{n_1}) + O(n_2^2 \cdot 2^{n_2}) + O(2^{n_1+n_2}) = O(n_1^2 \cdot 2^{n_1}) + O((2^{n_1} + n_2^2) \cdot 2^{n_2})$ time by Prop. 120. Thus, deciding b -responsiveness using matching is computationally more efficient. Note that n_1 in Prop. 120 refers to the number of reachable b -bounded markings in $env(N_1)$. In contrast, n_1 in Prop. 137 refers to the number of reachable b -bounded markings in $inner(N_1)$, which is in general much smaller than n_1 in Prop. 120, making the decision procedure using matching even more efficient.

We can use matching also to decide whether a given open net $Impl$ b -conforms to an interface-equivalent open net $Spec$: First, we compute $max_b(Spec)$. Then, we compute $MP_b(max_b(Spec))$ and check whether $Impl$ matches with $MP_b(max_b(Spec))$. Figure 79 illustrates this algorithm.

Let I and O be fixed with $s = |I \uplus O|$ and let $Impl$ and $Spec$ be two interface-equivalent open nets with s interface places, respectively. Let n_1 be the number of reachable b -bounded markings in $inner(Impl)$ and let n_2 be the number of reachable b -bounded markings in $env(Spec)$. We already showed in Sect. 5.2.3 that we can compute $MP_b(Spec)$ with $O(2^{n_2})$ states and $O(2^{n_2})$ transitions in time $O(n_2^2 \cdot 2^{n_2})$. By Def. 131, the labeled net $env(max_b(Spec))$ has at most $(b+1)^s \cdot O(2^{n_2}) = O(2^{n_2})$ reachable b -bounded markings. Thus, we can compute $MP_b(max_b(Spec))$ with $O(2^{2n_2})$ states and $O(2^{2n_2})$ transitions in $O(2^{n_2^2} \cdot 2^{2n_2})$ and decide whether $Impl$ matches with $MP_b(max_b(Spec))$ in $O(n_1^3) + O(n_1 \cdot 2^{n_2} \cdot 2^{2n_2})$.

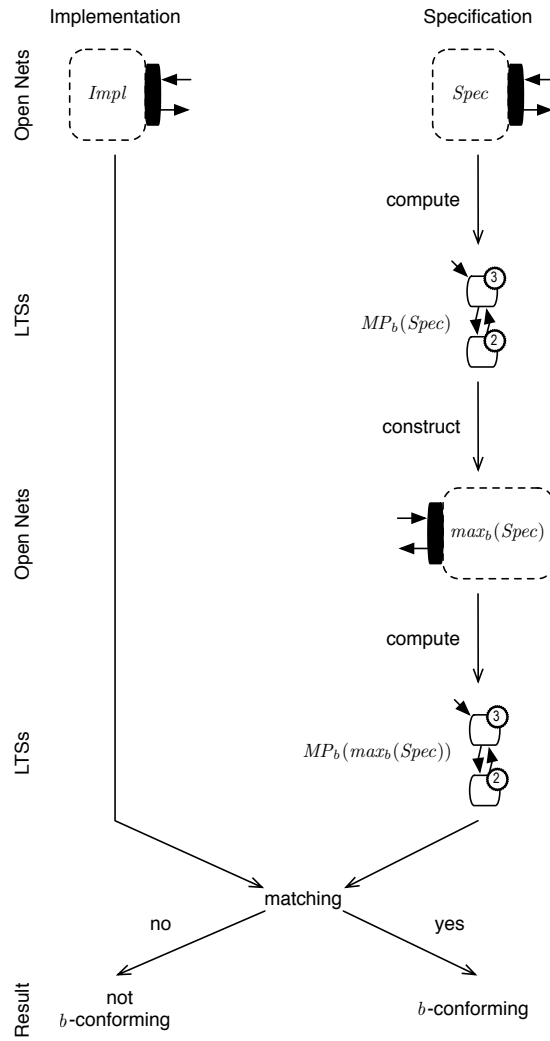


Figure 79: Using the maximal partner and matching to decide if a given open net $Impl$ b -conforms to an interface-equivalent open net $Spec$.

In contrast to the alternative decision procedure for b -responsiveness, the alternative decision procedure for b -conformance is practically much worse than the decision procedure in Sect. 5.2 regarding worst case complexity. However, n_1 refers to the much smaller LTS $inner(Impl)$ compared to the approach with Thm. 117, where n_1 refers to $env(Impl)$. Consequently, the alternative decision procedure for b -conformance might be more feasible in practice for an implementation $Impl$ with a very large state-space and a specification $Spec$ with a very small state-space.

5.4 IMPLEMENTATION AND EXPERIMENTAL RESULTS

We showed how to decide whether an open net C is a b -partner of an open net N and whether an open net $Impl$ b -conforms to an open net $Spec$ using the LTSs BSD_b and CSD_b in Sect. 5.2 and Sect. 5.3. Both the construction algorithm for BSD_b in Alg. 2 and the construction algorithm for CSD_b in Alg. 4 have been implemented in the tool Chloe [115], which we developed in the course of this thesis. For implementing the construction algorithm for $BSD_b(N)$, Chloe relies on the tool LoLa [257]—a general-purpose Petri net

model checking tool—to generate the bounded state-space of $env(N)$ (see line 1 in Alg. 2). The algorithms to check whether two given open nets are b -partners in Alg. 3 or b -conforming in Alg. 5 have been implemented in the tool Delain [78]. The tools Chloe and Delain are free open source software and were implemented in C++. Both tools were developed following a “one tool - one purpose” policy, which has been proven helpful in implementing a theory of correctness for open systems [155].

As a proof of concept, we calculate the LTSs BSD_b and CSD_b of our running examples D and U (Fig. 54), D' (Fig. 56), U' (Fig. 55), and five open systems of industrial size. These open systems are services [201]. Each process was modeled in WS-BPEL [130] and models a communication protocol or a business process. The services “Loan Approval” and “Purchase Order” are taken from the WS-BPEL specification [130], and the other three examples are industrial service models provided by a consulting company. To apply the algorithms of this chapter, we first translated the WS-BPEL processes into open nets using the compiler BPEL2OWFN [149]. Table 3 gives an overview of the characteristics of the derived open nets. We see that the open nets derived from the WS-BPEL processes have up to 90 places and 123 transitions. The interfaces of the open nets consist of up to 11 interface places.

open net (abbreviation)	$ P $	$ I $	$ O $	$ T $	$ F $
Database (D)	3	2	2	4	11
Patched Database (D')	2	2	2	3	8
First User (U)	2	2	2	2	6
Second User (U')	2	2	2	3	7
Contract Negotiation (CN)	76	4	7	98	294
Loan Approval (LA)	34	3	3	17	60
Purchase Order (PO)	74	4	6	96	290
Reservations (RS)	38	2	8	33	83
Ticket Reservation (TR)	90	3	6	123	363

Table 3: The size of the derived open nets.

We compute BSD_b and CSD_b for each of the nine open nets and check for b -conformance with itself (which should hold naturally), using the standard options of Chloe and Delain and bound values of 1 and 2. All computations in this section are conducted on a MacBook Air model A1466 [21] with one Intel Core i5 1.3 GHz CPU with 2 independent processor cores and 8 GiB of memory. The MacBook Air is weak in terms of computing power compared to existing business servers; for example, a standard IBM Blade server model HX5 [125] has up to four Intel Xeon 2.4 GHz CPUs with 10 independent processor cores each (i.e., 20 times as many cores as the MacBook Air which are, in addition, faster) and 256 GiB of memory (i.e., 32 times as many memory as the MacBook Air). However, we conduct all experiments on the MacBook Air to demonstrate the feasibility of our implementation on today’s average (personal) computers.

Table 4 shows the size of the resulting LTSs BSD_1 and BSD_2 , the time for computing them, and the memory consumption. We can see that computing BSD_1 is feasible with time consumptions from 0 to 13 seconds and memory consumption of up to 107 MiB (which is approx. $\frac{1}{75}$ of the available 8 GiB memory). Even computing the much larger LTS BSD_2 (which is up to six times larger than BSD_1) is feasible with time consumptions from 0 to 63 sec-

onds and memory consumption of up to 501 MiB (which is still only approx. $\frac{1}{16}$ of the available 8 GiB memory). However, in cases where $b > 2$, computing BSD_b might be too time consuming if we have to compute it twice for every b -responsiveness check using the algorithm in Fig. 72. Therefore, the alternative algorithm in Fig. 74 might perform better for those examples.

LTS	$ Q $	$ \delta $	$ \Sigma^{in} $	$ \Sigma^{out} $	time (s)	memory (KiB)
$BSD_1(D)$	12	48	2	2	0	1,424
$BSD_1(D')$	6	24	2	2	0	1,412
$BSD_1(U)$	8	32	2	2	0	1,412
$BSD_1(U')$	12	48	2	2	0	1,420
$BSD_1(CN)$	2,050	22,550	4	7	13	106,784
$BSD_1(LA)$	66	396	3	3	0	2,028
$BSD_1(PO)$	1,026	10,260	4	6	3	33,028
$BSD_1(RS)$	1,026	10,260	2	8	0	4,352
$BSD_1(TR)$	460	4,140	3	6	2	24,256
<hr/>						
$BSD_2(D)$	29	116	2	2	0	1,476
$BSD_2(D')$	11	44	2	2	0	1,440
$BSD_2(U)$	17	68	2	2	0	1,432
$BSD_2(U')$	32	128	2	2	0	1,464
$BSD_2(CN)$	10,370	114,070	4	7	63	501,344
$BSD_2(LA)$	218	1,308	3	3	0	3,140
$BSD_2(PO)$	5,186	51,860	4	6	15	133,644
$BSD_2(RS)$	2,306	23,060	2	8	0	7,344
$BSD_2(TR)$	2,594	23,346	3	6	9	89,236

Table 4: The size of BSD_1 and BSD_2 generated with the tool Chloe, including the used memory and time.

The tool Chloe outputs the computed LTS in the graph description language DOT [101]. We can easily visualize DOT files with Graphviz [90, 91], which is a heterogeneous collection of open source graph drawing tools. As an example, Fig. 80 shows the LTS $CSD_1(D)$ as computed by the tool Chloe and visualized by the tool dot from the Graphviz collection. The LTS in Fig. 80 coincides with the LTS we derived by hand in Fig. 67.

Table 5 shows the size of the resulting LTSs CSD_1 and CSD_2 , the time for computing them, and the memory consumption. We can see that CSD_1 has considerably fewer states than BSD_1 , ranging from 4 to 578 compared to 6 to 2,050. This difference in size of BSD_b and CSD_b becomes even more apparent if we consider CSD_2 and BSD_2 : The number of states of CSD_2 ranges from 5 to 578 whereas the number of states of BSD_2 ranges from 11 to 10,370. In addition, constructing CSD_1 and CSD_2 results in nearly no additional time and memory consumption compared to the time and memory we need to construct BSD_1 and BSD_2 , respectively. Still, computing CSD_1 and CSD_2 is feasible with time consumptions from 0 to 61 seconds and memory consumption of up to 501 MiB. Interestingly, four out of five industrial open systems (namely CN , LA , PO , and RS) seem to be designed for a bound of 1, as for them CSD_1 is identical to CSD_2 . Only for the industrial open system TR , CSD_1 differs from CSD_2 . In all cases, the respective checks for 1-conformance and 2-conformance performed instantly, therefore we did not include them in Tab. 5.

We conclude this section by comparing our approach to compute a maximal b -partner with two existing approaches from literature. Mooij et al. [180,

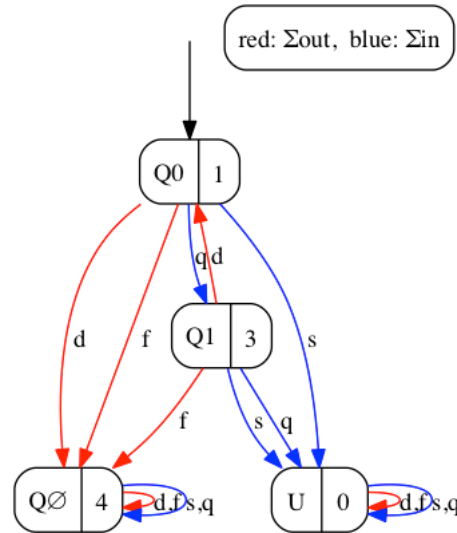


Figure 80: The LTS $CSD_1(D)$ as computed by the tool Chloe and visualized by the tool dot from the Graphviz collection. It coincides with the LTS in Fig. 67 which derived manually.

LTS	$ Q $	$ \delta $	$ \Sigma^{in} $	$ \Sigma^{out} $	time (s)	memory (KiB)
$CSD_1(D)$	4	16	2	2	0	1,424
$CSD_1(D')$	5	20	2	2	0	1,412
$CSD_1(U)$	8	32	2	2	0	1,412
$CSD_1(U')$	6	24	2	2	0	1,420
$CSD_1(CN)$	578	6,358	4	7	13	106,784
$CSD_1(LA)$	22	132	3	3	0	2,028
$CSD_1(PO)$	170	1,700	4	6	4	33,028
$CSD_1(RS)$	371	3,710	2	8	0	4,352
$CSD_1(TR)$	112	1,008	3	6	2	24,256
<hr/>						
$CSD_2(D)$	5	20	2	2	0	1,476
$CSD_2(D')$	6	24	2	2	0	1,444
$CSD_2(U)$	17	68	2	2	0	1,432
$CSD_2(U')$	6	24	2	2	0	1,464
$CSD_2(CN)$	578	6,358	4	7	61	501,344
$CSD_2(LA)$	22	132	3	3	0	3,140
$CSD_2(PO)$	170	1,700	4	6	15	133,644
$CSD_2(RS)$	371	3,710	2	8	0	7,376
$CSD_2(TR)$	182	1,638	3	6	9	89,236

Table 5: The size of CSD_1 and CSD_2 generated with the tool Chloe, including the used memory and time.

179] construct a finite maximal b -partner—called maximal controller—for a conformance relation—called accordance [226, 11]—that preserves b -bounded deadlock freedom. In essence, their construction algorithm “unfolds” the operating guideline of Lohmann et al. [153] into a single service automaton, which results in an exponential blowup of the size of the maximal b -partner compared to the size of the operating guideline. Parnjai [203] lifts this construction algorithm to a conformance relation based on a notion of responsiveness that is weaker than ours. In contrast, our construction of a maximal

b -partner max_b from the LTS CSD_b in Sect. 5.3 at most doubles the size of max_b compared to the size of CSD_b . As the size of CSD_b is comparable to the size of an operating guideline (see Sect. 5.2), our maximal b -partner is in general drastically smaller than the maximal controllers in [180, 179, 203]. In [203], Parnjai advocates to construct an additional “compact canonical” maximal b -partner: In essence, the compact maximal b -partner in [203] is a maximal b -partner from [179] reduced under some failure-semantics (called “responsive failures” in [203]). In contrast, we showed in Sect. 5.3 that a maximal b -partner needs to be maximal only with respect to a trace-based semantics. Therefore, our approach should still yield a smaller maximal b -partner in most cases than the approach in [203], while being more general in terms of the considered responsiveness notion.

In the following, we compare the approaches in [179] and [203] with our approach by computing a maximal 1-partner for each of the nine previously mentioned open nets. Although our setting is more general (i.e., the approaches in [179] and [203] only consider internally bounded open nets whereas we consider arbitrary open nets), the size of the resulting maximal 1-partners should give an impression of the approaches’ effectiveness. The tool Chloe [115] computes a maximal b -partner according to our approach; the tool Maxis [204] computes a maximal b -partner according to the approaches in [179] and [203]. Table 6 shows the size of the resulting maximal 1-partners. In eight cases, our approach results in a smaller maximal 1-partner than the approaches in [179] and [203]; only for the open net U , the maximal 1-partner of [203] is smaller than ours. In two cases, Maxis was only able to compute the maximal 1-partner as a service automaton but not as an open net. Therefore, we give the number of states of the respective service automaton as lower bound on the size of the resulting open net.

5.5 CONCLUSIONS

In this chapter, we investigated the b -conformance relation that arises from b -responsiveness—that is, a variant of responsiveness where the number of pending messages never exceeds a previously known bound b . We investigated b -conformance because conformance and compositional conformance turned out to be undecidable in Chap. 4. Although respecting a bound b may seem restricting, the notion of b -conformance is still practically relevant: Distributed systems operate on a middleware with buffers that are of bounded size. The actual buffer size can be the result of a static analysis of the underlying middleware or of the communication behavior of an open system, or simply be chosen sufficiently large.

We gave a trace-based characterization for b -conformance, thereby adapting and combining results from conformance in Chap. 4 and work on traces that cannot be used reliably by any partner [162]. Due to the latter traces, b -conforming systems may violate language inclusion. Giving an answer to an open question, we showed that b -conformance is not a precongruence.

In contrast to conformance, b -conformance is decidable. Thus, we elaborated a decision procedure for b -conformance. For a given open net, we additionally developed a finite characterization of all b -partners and all b -conforming open nets. These finite characterizations serve as an alternative decision procedure for b -conformance. We implemented the decision procedure for b -conformance in the tools Chloe [115] and Delain [78] and evaluated it using open nets of industrial size.

open net	using the approach in Sect. 5.3						using the approach in [179, 205]						using the approach in [203]					
	P	I	O	T	F		P	I	O	T	F		P	I	O	T	F	
$max_1(D)$	4	2	2	12	35		48	2	2	132	350		12	2	2	19	47	
$max_1(D')$	5	2	2	15	44		69	2	2	196	522		15	2	2	24	60	
$max_1(U)$	10	2	2	30	87		53	2	2	142	376		14	2	2	22	55	
$max_1(U')$	6	2	2	16	47		54	2	2	142	376		16	2	2	25	63	
$max_1(CN)$	1,011	7	4	8,331	24,559	> 497,500	7	4	> 497,500	> 497,500	3,003	7	4	7,275	19,409			
$max_1(LA)$	27	3	3	108	318	692	3	3	2,476	6,756	67	3	3	131	346			
$max_1(PO)$	259	6	4	1,812	5,346	49,830	6	4	252,902	709,054	741	6	4	1,745	4,672			
$max_1(RS)$	489	8	2	4,154	12,343	> 221,851	8	2	> 221,851	> 221,851	2,327	8	2	6,197	16,643			
$max_1(TR)$	150	6	3	1,004	2,973	20,937	6	3	101,034	282,284	455	6	3	1,147	3,105			

Table 6: The size of the resulting maximal 1-partners.

This chapter is based on results published in [248, 249].

IN Chap. 5, we analyzed the b -conformance relation that is—in contrast to conformance—decidable and therefore suitable for verification. However, we also showed in Chap. 5 that the b -conformance relation is still not preserved under the open net composition operator. Consequently, we use this chapter to investigate the coarsest precongruence that is contained in the b -conformance relation—that is, compositional b -conformance. Table 2 illustrates how this chapter fits into the structure of Part II.

relation	characterization	compositionality	decidability
conformance	Chap. 4	Chap. 4	Chap. 4
b -conformance	Chap. 5	Chap. 6	Chap. 5 & Chap. 6

Table 7: The structure of Part II without Chap. 7. We highlight the current chapter with a gray background.

In this chapter, we characterize compositional b -conformance and show that it is—in contrast to compositional conformance in Chap. 4—decidable. The highlighted part of Fig. 53 illustrates how we achieve this. We introduce a new denotational semantics for open nets based on failures in Sect. 6.1. The new semantics extends the \mathcal{F}_{fin}^+ -semantics from Chap. 4 by information about $bound_b$ -violations, which is why we refer to it as b -bounded \mathcal{F}_{fin}^+ -semantics. We show that a refinement relation build upon the b -bounded \mathcal{F}_{fin}^+ -semantics, to which we subsequently refer to as $\mathcal{F}_{b,fin}^+$ -refinement, characterizes compositional b -conformance. In essence, $\mathcal{F}_{b,fin}^+$ -refinement extends the \mathcal{F}_{fin}^+ -refinement from Chap. 4 by including (the inclusion of) the $bound_b$ -violators. We elaborate a decision procedure for compositional b -conformance in Sect. 6.2. To this end, we show that \mathcal{F}_{fin}^+ -refinement is decidable for two finite LTSs. Next, we reduce $\mathcal{F}_{b,fin}^+$ -refinement under the assumption of $bound_b$ -inclusion to \mathcal{F}_{fin}^+ -refinement on two finite LTSs. We already showed in Chap. 5 how to decide $bound_b$ -inclusion. That way, we conclude decidability of compositional b -conformance. We finish this chapter with a conclusion in Sect. 6.3.

6.1 CHARACTERIZING COMPOSITIONAL b -CONFORMANCE

To characterize compositional b -conformance, we introduce in Sect. 6.1.1 a failure-based semantics for open nets similar to the \mathcal{F}_{fin}^+ -semantics in Sect. 4.2. We refer to the new semantics as b -bounded \mathcal{F}_{fin}^+ -semantics. We show that the b -bounded \mathcal{F}_{fin}^+ -semantics of a composition of two composable open nets can be derived from the b -bounded \mathcal{F}_{fin}^+ -semantics of the composed open nets. In Sect. 6.1.2, we show that a refinement relation build upon the b -bounded \mathcal{F}_{fin}^+ -semantics coincides with compositional b -

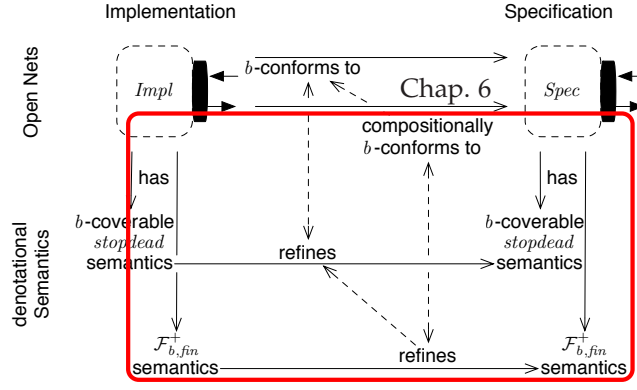


Figure 81: Characterizing compositional b -conformance using a denotational semantics for open nets. A solid arc illustrates the relation described by the corresponding arc label. Dashed arcs illustrate logical implication or logical equivalence, depending on their number of heads.

conformance. In other words, we provide a failure-based characterization of compositional b -conformance.

6.1.1 The b -bounded \mathcal{F}_{fin}^+ -semantics for open nets

In Sect. 4.2, we used the \mathcal{F}_{fin}^+ -semantics to characterize the coarsest precongruence that is contained in the conformance relation. So it suggests itself that the \mathcal{F}_{fin}^+ -semantics is closely related to the new semantics. To characterize the coarsest precongruence that is contained in b -conformance, we need to cope with the restriction to a b -bounded composition $N \oplus C$ of two open nets N and C in Def. 41. We therefore add information about $bound_b$ -violators to the \mathcal{F}_{fin}^+ -semantics in Def. 63. The resulting b -bounded \mathcal{F}_{fin}^+ -semantics consists of the set of b -violators from Def. 84 and the \mathcal{F}_{fin}^+ -semantics extended with all fintree failures (w, X, Y) , where w is a trace of $bound_b$ and X and Y are (almost) arbitrary languages over the alphabet. That way, we are flooding the set of fintree failures \mathcal{F}_{fin}^+ in the same way we have been flooding the sets of $stop$ - and $dead$ -traces in the b -bounded $stopdead$ -semantics in Def. 84.

Definition 138 [b -bounded \mathcal{F}_{fin}^+ -semantics]

Let N be a labeled net. We define the set of $finbound_b$ -violators of N by $finbound_b(N) = bound_b(N) \times \mathcal{P}(\Sigma^+) \times \mathcal{P}(\Sigma^*)$. The b -bounded \mathcal{F}_{fin}^+ -semantics of N is defined by the sets

- $bound_b(N)$, and
- $\mathcal{F}_{b,fin}^+(N) = \mathcal{F}_{fin}^+(N) \cup finbound_b(N)$.

Example 139 Consider the open net D in Fig. 82a and the open net D' in Fig. 82b, which we already used as running examples in Chap. 5. We detailed their 1-bounded $stopdead$ -semantics in Ex. 85 and Ex. 91, which

contains their set of $bound_1$ -violators and their (flooded) language. The language and the $bound_1$ -violators of D are

$$\begin{aligned} L(D) &= \{w \in \{s, q, d\}^* \mid \forall v \sqsubseteq w : |v|_d \leq |v|_q\} \\ &\quad \cup \{w f z \mid w, z \in \{s, q, d\}^* \wedge \forall v \sqsubseteq w : |v|_d \leq |v|_q \\ &\quad \quad \quad \wedge |w|_s > 0 \wedge |z|_d \leq |w|_q - |w|_d\}, \\ bound_1(D) &= \uparrow \{w \in L(D) \mid \exists v \sqsubseteq w : |v|_d + 1 < |v|_q\} \\ &\quad \cup \uparrow \{w \in L(D) \mid \exists v \sqsubseteq w : |v|_f + 1 < |v|_s\}, \end{aligned}$$

and the language and the $bound_1$ -violators of D' are

$$\begin{aligned} L(D') &= \{w \in \{s, q, d\}^* \mid \forall v \sqsubseteq w : |v|_d \leq |v|_q\}, \\ bound_1(D') &= \uparrow \{w \in L(D') \mid \exists v \sqsubseteq w : |v|_d + 1 < |v|_q\} \\ &\quad \cup \uparrow \{w \in L(D') \mid \exists v \sqsubseteq w : |v|_s > 1\}. \end{aligned}$$

Thus, the set of $finbound_1$ -violators of D is

$$finbound_1(D) = bound_1(D) \times \mathcal{P}(\{s, q, d, f\}^+) \times \mathcal{P}(\{s, q, d, f\}^*),$$

and the set of $finbound_1$ -violators of D' is

$$finbound_1(D') = bound_1(D') \times \mathcal{P}(\{s, q, d, f\}^+) \times \mathcal{P}(\{s, q, d, f\}^*).$$

As an example for the fintree failures of the 1-bounded \mathcal{F}_{fin}^+ -semantics of D and D' , consider the trace s . After executing the trace s , $env(D)$ reaches the marking $[p_1, s^i]$, $[p_0]$, or $[f^o]$. In all three markings, $env(D)$ cannot refuse the trace f . In contrast, after executing the trace s , $env(D')$ reaches either the marking $[p_1, s^i]$ or the marking $[\]$. In both markings, it can refuse the trace f because no reachable marking of $env(D')$ enables transition f . The trace s is neither a $bound_1$ -violator of D nor a $bound_1$ -violator of D' , and the empty set \emptyset is a fin-refusal set of every marking of $env(D)$ and $env(D')$. Thus, we can distinguish D and D' by their 1-bounded \mathcal{F}_{fin}^+ -semantics: We have $(s, \{f\}, \emptyset) \notin \mathcal{F}_{1, fin}^+(D)$ but $(s, \{f\}, \emptyset) \in \mathcal{F}_{1, fin}^+(D')$, for instance. \diamond

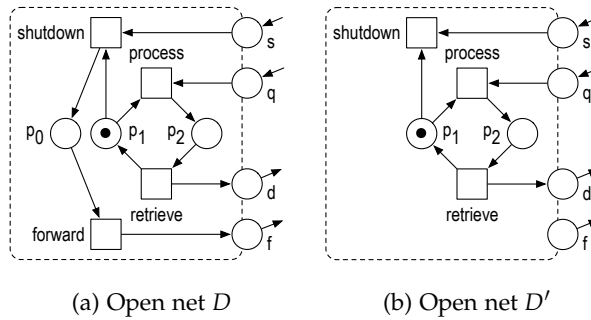


Figure 82: The open nets D and D' from Fig. 54a and Fig. 56 modeling a database server and a patched database server. In addition to the figures, we have $\Omega_D = \{[p_0]\}$ and $\Omega_{D'} = \{[\]\}$.

Like the \mathcal{F}_{fin}^+ -semantics in Lem. 65, the composition of two composable open nets N_1 and N_2 has the same b -bounded \mathcal{F}_{fin}^+ -semantics as the parallel composition of their respective environments, $env(N_1) \uparrow env(N_2)$.

Lemma 140 [b -bounded \mathcal{F}_{fin}^+ -semantics for open net composition]

For two composable open nets N_1 and N_2 , we have

$$\mathcal{F}_{b,fin}^+(env(N_1 \oplus N_2)) = \mathcal{F}_{b,fin}^+(env(N_1) \uparrow env(N_2)).$$

Proof. Follows directly from Lem. 30: If one net has a $bound_b$ -violator w due to marking m , then the other net can reach an agreeing marking m' with trace w ; thus w is also a $bound_b$ -violator for the other net. Likewise, by applying the same argumentation as in the proof of Lem. 65, we conclude that if one net has a fintree failure (w, X, Y) so does the other net. \square

In the remainder of this section, we shall show that the b -bounded \mathcal{F}_{fin}^+ -semantics is compositional; that is, the b -bounded \mathcal{F}_{fin}^+ -semantics of a composition $N_1 \oplus N_2$ of two composable open nets N_1 and N_2 can be derived solely from the b -bounded \mathcal{F}_{fin}^+ -semantics of N_1 and N_2 . To this end, we characterize the b -bounded \mathcal{F}_{fin}^+ -semantics for labeled net composition and hiding and finally combine these results to determine the b -bounded \mathcal{F}_{fin}^+ -semantics for open net composition. First, we consider the b -bounded \mathcal{F}_{fin}^+ -semantics for the composition of two labeled nets.

Lemma 141 [b -bounded \mathcal{F}_{fin}^+ -semantics for labeled net composition]

For two composable labeled nets N_1 and N_2 , we have

$$\begin{aligned} \mathcal{F}_{b,fin}^+(N_1 \parallel N_2) = & \{ (w, X, Y) \mid \exists (w_i, X_i, Y_i) \in \mathcal{F}_{b,fin}^+(N_i) \text{ for } i = 1, 2 : \\ & w \in w_1 \parallel w_2 \wedge \forall x \in X, y \in Y : \\ & \quad (x \in x_1 \parallel x_2 \text{ implies } x_1 \in X_1 \vee x_2 \in X_2) \\ & \quad \wedge (y \in y_1 \parallel y_2 \text{ implies } y_1 \in Y_1 \vee y_2 \in Y_2) \} \\ & \cup finbound_b(N_1 \parallel N_2). \end{aligned}$$

Proof. We write E for $N_1 \parallel N_2$.

\subseteq : Let $(w, X, Y) \in \mathcal{F}_{b,fin}^+(E)$. If w is not a $bound_b$ -violator, then $(w, X, Y) \in \mathcal{F}_{fin}^+(E)$ by Def. 138, and we conclude with Lem. 66 and Def. 138 that it is contained in the first set on the right hand side. If w is a $bound_b$ -violator of E , then $(w, X, Y) \in finbound_b(N_1 \parallel N_2)$ by Def. 138.

\supseteq : Let (w, X, Y) arise from $(w_i, X_i, Y_i) \in \mathcal{F}_{b,fin}^+(N_i)$ for $i = 1, 2$. If both $(w_i, X_i, Y_i) \in \mathcal{F}_{fin}^+(N_i)$, then $(w, X, Y) \in \mathcal{F}_{fin}^+(E)$ by Lem. 66 and $\mathcal{F}_{fin}^+(E) \subseteq \mathcal{F}_{b,fin}^+(E)$ by Def. 138. Assume now that at least one fintree failure (w_i, X_i, Y_i) is not contained in the respective \mathcal{F}_{fin}^+ -semantics. Then trace w_i is a $bound_b$ -violator by Def. 138 and so is w by Prop. 86(1), because $w_{3-i} \in L_b(N_{3-i})$ by Def. 138. Thus, $(w, X_1 \cup X_2, Y_1 \cup Y_2) \in finbound_b(E) \subseteq \mathcal{F}_{b,fin}^+(E)$ due to Def. 138. \square

Next, we consider hiding for the b -bounded \mathcal{F}_{fin}^+ -semantics on a labeled net.

Lemma 142 [b -bounded \mathcal{F}_{fin}^+ -semantics under hiding]

For a labeled net N and a label set $A \subseteq \Sigma^*$, we have

$$\mathcal{F}_{b,fin}^+(N/A) = \{(\phi(w), X, Y) \mid (w, \phi^{-1}(X), \phi^{-1}(Y)) \in \mathcal{F}_{b,fin}^+(N)\}.$$

Proof. Follows from Lem. 67. \square

We finally combine Lem. 140, Lem. 141, and Lem. 142 to show how the b -bounded \mathcal{F}_{fin}^+ -semantics for the composition $N_1 \oplus N_2$ of two open nets N_1 and N_2 can be determined from the b -bounded \mathcal{F}_{fin}^+ -semantics of N_1 and N_2 .

Proposition 143 [b -bounded \mathcal{F}_{fin}^+ -semantics for open net composition]

For two composable open nets N_1 and N_2 , we have

$$\begin{aligned} \mathcal{F}_{b,fin}^+(N_1 \oplus N_2) = & \{(w, X, Y) \mid \exists (w_i, X_i, Y_i) \in \mathcal{F}_{b,fin}^+(N_i) \text{ for } i = 1, 2: \\ & w \in w_1 \uparrow w_2 \wedge \forall x \in X, y \in Y : \\ & \quad (x \in x_1 \uparrow x_2 \text{ implies } x_1 \in X_1 \vee x_2 \in X_2) \\ & \quad \wedge (y \in y_1 \uparrow y_2 \text{ implies } y_1 \in Y_1 \vee y_2 \in Y_2)\} \\ & \cup \text{finbound}_b(N_1 \oplus N_2). \end{aligned}$$

Proof. Let $F(N_1, N_2)$ denote the first set on the right-hand side in Lem. 141. According to Lem. 140, we can consider $\mathcal{F}_{b,fin}^+(env(N_1) \uparrow env(N_2))$ instead of $\mathcal{F}_{b,fin}^+(N_1 \oplus N_2)$. Because \uparrow is \parallel followed by hiding, we can determine the set $\mathcal{F}_{b,fin}^+(env(N_1) \uparrow env(N_2))$ by applying hiding (according to Lem. 142) to the right-hand side of Lem. 141. As a result, $F(N_1, N_2)$ turns into the first set in the present proposition, just as Prop. 68 results from Lem. 66 with Lem. 67. More easily, $\text{finbound}_b(N_1 \parallel N_2)$ is analogously translated into $\text{finbound}_b(N_1 \oplus N_2)$ according to Prop. 86(3). \square

In this section, we presented the b -bounded \mathcal{F}_{fin}^+ -semantics for open nets, as an extension of the \mathcal{F}_{fin}^+ -semantics for open nets from Sect. 4.2. In the following section, we define a refinement relation between two open nets based on their b -bounded \mathcal{F}_{fin}^+ -semantics that characterizes compositional b -conformance.

6.1.2 Refinement on the b -bounded \mathcal{F}_{fin}^+ -refinement

In this section, we define a refinement relation between two open nets based on their b -bounded \mathcal{F}_{fin}^+ -semantics and prove that this refinement relation coincides with compositional b -conformance.

The $\mathcal{F}_{b,fin}^+$ -refinement relation combines bound_b -inclusion from Thm. 90 and the \mathcal{F}_{fin}^+ -refinement relation from Def. 69.

Definition 144 [$\mathcal{F}_{b,fin}^+$ -refinement]

For two action-equivalent labeled nets $Impl$ and $Spec$, $Impl$ $\mathcal{F}_{b,fin}^+$ -refines $Spec$, denoted by $Impl \sqsubseteq_{\mathcal{F}_{b,fin}^+} Spec$, if

1. $bound_b(Impl) \subseteq bound_b(Spec)$, and
2. $\forall (w, X, Y) \in \mathcal{F}_{b,fin}^+(Impl) :$

$$\exists x \in \{\varepsilon\} \cup \downarrow X \cup \downarrow Y : (wx, x^{-1}X, x^{-1}Y) \in \mathcal{F}_{b,fin}^+(Spec).$$

For two interface-equivalent open nets $Impl$ and $Spec$, we define $Impl \sqsubseteq_{\mathcal{F}_{b,fin}^+} Spec$, if $env(Impl) \sqsubseteq_{\mathcal{F}_{b,fin}^+} env(Spec)$.

Example 145 Consider again the open nets D and D' in Fig. 82. For any bound b , we have $(s, \{f\}, \emptyset) \in \mathcal{F}_{b,fin}^+(D')$: After trace s , $env(D')$ is either in marking $[p_1, s^i]$ or in marking $[\]$. In both cases, $env(D')$ can refuse f because $env(D')$ cannot fire transition f at all. However, we have $(s, \{f\}, \emptyset) \notin \mathcal{F}_{b,fin}^+(D)$ and $(sf, \{\varepsilon\}, \emptyset) \notin \mathcal{F}_{b,fin}^+(D)$, because $env(D)$ can never refuse f after the trace s , and $(sf, \{\varepsilon\}, \emptyset)$ is not a fintree failure by Def. 138. Therefore, D does not $\mathcal{F}_{b,fin}^+$ -refine D' according to Def. 144. \diamond

If two open nets are in the $\mathcal{F}_{b,fin}^+$ -refinement relation, then this implies inclusion of their flooded languages, $stop_b$ -traces, and $dead_b$ -traces.

Lemma 146 [$\mathcal{F}_{b,fin}^+$ -refinement implies L_b -, $stop_b$ -, $dead_b$ -inclusion]

For two action-equivalent labeled nets $Impl$ and $Spec$, we have

1. $Impl \sqsubseteq_{\mathcal{F}_{b,fin}^+} Spec$ implies $L_b(Impl) \subseteq L_b(Spec)$.
2. $Impl \sqsubseteq_{\mathcal{F}_{b,fin}^+} Spec$ implies $stop_b(Impl) \subseteq stop_b(Spec)$.
3. $Impl \sqsubseteq_{\mathcal{F}_{b,fin}^+} Spec$ implies $dead_b(Impl) \subseteq dead_b(Spec)$.

Proof. (1) Let $w \in L_b(Impl)$. Then $(w, \emptyset, \emptyset) \in \mathcal{F}_{b,fin}^+(Impl)$ by Def. 138 and $(w, \emptyset, \emptyset) \in \mathcal{F}_{b,fin}^+(Spec)$ by Def. 144, which immediately implies $w \in L_b(Spec)$ by Def. 138.

(2) Let $w \in stop_b(Impl)$. Then we use the proof of (the reverse implication of) Thm. 75 by replacing $stop$ by $stop_b$ to conclude that $w \in stop_b(Spec)$.

(3) Similar argumentation as for (2). \square

In the remainder of this section, we shall show that $\mathcal{F}_{b,fin}^+$ -refinement is a precongruence on open nets for the composition operator \oplus , just like \mathcal{F}_{fin}^+ -refinement is a precongruence on open nets for \oplus by Thm. 74. As for the proof of Thm. 74, we first show the precongruence result for labeled nets and operator \parallel (cf. Lem. 72). Then, we show that this result is also preserved under hiding (cf. Lem. 73). Finally, we combine these results to show the precongruence for open nets and the operator \oplus .

First, we show with Lem. 148 that $\mathcal{F}_{b,fin}^+$ -refinement is a precongruence for labeled nets and operator \parallel , thereby using the precongruence result for \mathcal{F}_{fin}^+ -refinement from Lem. 72. For the proof of Lem. 148, we shall use that

the following four saturation conditions, which hold for the \mathcal{F}_{fin}^+ -semantics (cf. Lem. 71), also hold for the b -bounded \mathcal{F}_{fin}^+ -semantics:

Lemma 147 [saturation conditions]

For a labeled net N , we have

1. $(w, X, Y) \in \mathcal{F}_{b,fin}^+(N), X' \subseteq X, Y' \subseteq Y$ implies $(w, X', Y') \in \mathcal{F}_{b,fin}^+(N)$
2. $(w, X, Y) \in \mathcal{F}_{b,fin}^+(N) \wedge \forall z \in Z : (wz, z^{-1}X, z^{-1}Y) \notin \mathcal{F}_{b,fin}^+(N)$ implies $(w, X \cup Z, Y \cup Z) \in \mathcal{F}_{b,fin}^+(N)$
3. $(w, X, Y) \in \mathcal{F}_{b,fin}^+(N)$ implies $(w, \uparrow X, Y) \in \mathcal{F}_{b,fin}^+(N)$
4. $(w, X, Y) \in \mathcal{F}_{b,fin}^+(N)$ implies $(w, X, X \cup Y) \in \mathcal{F}_{b,fin}^+(N)$

Proof. To see these conditions, consider first some $(w, X, Y) \in \mathcal{F}_{fin}^+(N) \subseteq \mathcal{F}_{b,fin}^+(N)$. Then, items (1)–(4) follow directly from Lem. 72. Now, consider a fintree failure (w, X, Y) with $w \in \text{bound}_b(N)$; here, all four conditions are immediate because $(w, X', Y') \in \mathcal{F}_{b,fin}^+(N)$ for any $X' \in \mathcal{P}((I \uplus O)^+)$ and $Y' \in \mathcal{P}((I \uplus O)^*)$. \square

Lemma 148

$\mathcal{F}_{b,fin}^+$ -refinement is a precongurence for labeled nets with respect to \parallel .

Proof. Let $F(N_1, N_2)$ denote the first set on the right-hand side in Lem. 141. The precongurence result for \mathcal{F}_{fin}^+ -refinement in Lem. 72 holds for general sets of fintree failures (see the remark below Lem. 72). We make use of this, although this defining equation does not match Lem. 141, but just gives $F(N_1, N_2)$. Now let $\text{Impl} \sqsubseteq_{\mathcal{F}_{b,fin}^+} \text{Spec}$ and C be a composable labeled net for Impl and Spec . We have to check the two items of Def. 144 to prove that $\text{Impl} \parallel C \sqsubseteq_{\mathcal{F}_{b,fin}^+} \text{Spec} \parallel C$.

The first item of Def. 144 follows from Prop. 86(1) (which holds for labeled nets in general) because our assumption implies $\text{bound}_b(\text{Impl}) \subseteq \text{bound}_b(\text{Spec})$ as well as—due to Lem. 146(1)— $L_b(\text{Impl}) \subseteq L_b(\text{Spec})$.

For the second item, we first consider some $(w, X, Y) \in F(\text{Impl}, C)$. We observe that, due to Def. 144, $\mathcal{F}_{b,fin}^+(\text{Impl})$ is related to $\mathcal{F}_{b,fin}^+(\text{Spec})$ in the sense of \mathcal{F}_{fin}^+ -refinement. So by Lem. 72, $\exists x \in \{\varepsilon\} \cup \downarrow X \cup \downarrow Y : (wx, x^{-1}X, x^{-1}Y) \in F(\text{Spec}, C) \subseteq \mathcal{F}_{b,fin}^+(\text{Spec} \parallel C)$. Second, we consider a fintree failure $(w, X, Y) \in \text{finbound}_b(\text{Impl} \parallel C)$. This time, due to $\text{bound}_b(\text{Impl} \parallel C) \subseteq \text{bound}_b(\text{Spec} \parallel C)$, we even have $(w, X, Y) \in \text{finbound}_b(\text{Spec} \parallel C) \subseteq \mathcal{F}_{b,fin}^+(\text{Spec} \parallel C)$; that is, Def. 144(2) is satisfied taking $x = \varepsilon$. \square

Next, we show that $\mathcal{F}_{b,fin}^+$ -refinement for labeled nets is preserved under hiding.

Lemma 149

$\mathcal{F}_{b,fin}^+$ -refinement for labeled nets is preserved under hiding.

Proof. Let Impl and Spec be two labeled nets such that $\text{Impl} \sqsubseteq_{\mathcal{F}_{b,fin}^+} \text{Spec}$, and $A \subseteq \Sigma^*$. Then Lem. 142 directly implies Def. 144(1) for Impl/A and

$Spec/A$. Furthermore, the characterization in Lem. 142 corresponds to the defining equation for hiding in Lem. 73, so Def. 144(2) is inherited from the precongruence result in Lem. 73 for \mathcal{F}_{fin}^+ -refinement and hiding. \square

Lemma 148 and Lem. 149 enable us to show the first main result of this section: $\mathcal{F}_{b,fin}^+$ -refinement is a precongruence for the open net composition operator \oplus . As for the precongruence result in Thm. 74, the proof idea is to translate the operator \oplus for open nets into the operator \uparrow for labeled nets followed by hiding.

Theorem 150 [$\mathcal{F}_{b,fin}^+$ -refinement is a precongruence]

$\mathcal{F}_{b,fin}^+$ -refinement is a precongruence for open nets with respect to \oplus .

Proof. This is now completely analogous to the proof of Thm. 74, except that here the semantics associates two sets with a net and we use Lem. 148 and Lem. 149 on the basis of Lem. 72 and Lem. 73. \square

With the next theorem, we show the second main result of this section: $\mathcal{F}_{b,fin}^+$ -refinement coincides with the coarsest precongruence that is contained in the b -conformance relation—that is, compositional b -conformance.

For the implication of the proof, we show, amongst others, that compositional b -conformance implies $bound_b$ - and L_b -inclusion. This illustrates an inherent difference to the characterization of the b -conformance preorder in Thm. 97, which does not imply L_b -inclusion but only uL_b -inclusion (see Thm. 90 and Ex. 91 and Ex. 92 for a more detailed explanation).

Lemma 151 [$\sqsubseteq_{b,conf}^c$ implies $bound_b$ - and L_b -inclusion]

For two interface-equivalent open nets $Impl$ and $Spec$, we have

1. $Impl \sqsubseteq_{b,conf}^c Spec$ implies $bound_b(Impl) \subseteq bound_b(Spec)$.
2. $Impl \sqsubseteq_{b,conf}^c Spec$ implies $L_b(Impl) \subseteq L_b(Spec)$.

Proof. (1) Let $Impl \sqsubseteq_{b,conf}^c Spec$. We show $bound_b(Impl) \subseteq bound_b(Spec)$ by contradiction. So assume there exists a trace $w \in bound_b(Impl) \setminus bound_b(Spec)$. Then we can construct an open A such that A and $Spec$ (and equivalently $Impl$) are composable, $w \in L_b(A)$, and $A \oplus Spec$ is a b -bounded closed net (like the open net N_w in the proof of Thm. 61). By Prop. 86(3), we have $\varepsilon \in bound_b(A \oplus Impl)$, thus $A \oplus Impl$ is not b -bounded. Then the construction in Fig. 83 shows that open net B in Fig. 83b is a b -partner of $Spec \oplus A'$ but not of $Impl \oplus A'$. This contradicts that $Impl \sqsubseteq_{b,conf}^c Spec$ by Def. 47.

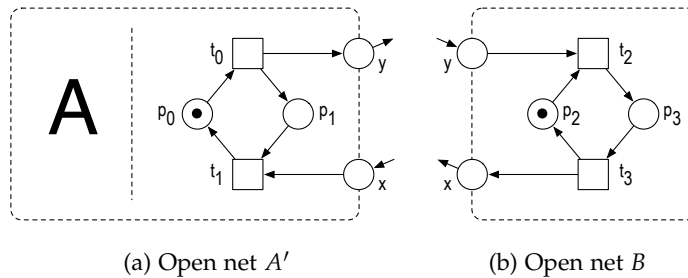


Figure 83: Construction of the open nets A' and B for the proof of Lem. 151.

(2) Similar argumentation as for (1), but we construct A such that $w \in \text{bound}_b(A)$ (like the open net C in the proof of Lem. 96(1)). \square

Because compositional b -conformance implies bound_b - and L_b -inclusion by Lem. 151, there is no need to incorporate b -uncoverable traces into the b -bounded \mathcal{F}_{fin}^+ -semantics in Def. 138. In contrast, we could characterize b -conformance only after incorporating b -uncoverable traces into the b -bounded stopdead -semantics in Def. 93.

For the reverse implication of the proof, we explicitly rely on Thm. 90 by showing that $\mathcal{F}_{b,fin}^+$ -refinement implies inclusion of their bounded languages, stop -traces, and dead -traces.

Theorem 152 [$\mathcal{F}_{b,fin}^+$ -refinement is the coarsest precongruence]

For two interface-equivalent open nets Impl and Spec , we have

$$\text{Impl} \sqsubseteq_{b,conf}^c \text{Spec} \quad \text{iff} \quad \text{Impl} \sqsubseteq_{\mathcal{F}_{b,fin}^+} \text{Spec}.$$

Proof. \Rightarrow : Let $\text{Impl} \sqsubseteq_{b,conf}^c \text{Spec}$. The first item of Def. 144 follows directly from Lem. 151(1).

For the second item of Def. 144, let $(w, X, Y) \in \mathcal{F}_{b,fin}^+(\text{Impl})$ such that $w \notin \text{bound}_b(\text{Impl})$. Otherwise, $w \in \text{bound}_b(\text{Impl}) \subseteq \text{bound}_b(\text{Spec})$ by the previously shown bound_b -inclusion, and $(w, X, Y) \in \mathcal{F}_{b,fin}^+(\text{Spec})$ by Def. 138. We use net N in Fig. 47 as in the proof of Thm. 75. Following the argumentation in the proof of Thm. 75, we have that if an open net C is not a b -partner of $\text{Impl} \oplus N$ so it is not a b -partner of $\text{Spec} \oplus N$. We distinguish three cases: If $w \in \text{bound}_b(\text{Spec})$, then $(w, X, Y) \in \mathcal{F}_{b,fin}^+(\text{Spec})$ by Def. 138. If $wu \in \text{bound}_b(\text{Spec})$ with $u \in \downarrow X \cup \downarrow Y$, then $(wu, u^{-1}X, u^{-1}Y) \in \mathcal{F}_{b,fin}^+(\text{Spec})$ by Def. 138. Otherwise, we use the argumentation in the proof of Thm. 75 to conclude that $(w, X, Y) \in \mathcal{F}_{b,fin}^+(\text{Spec})$.

\Leftarrow : Let $\text{Impl} \sqsubseteq_{\mathcal{F}_{b,fin}^+} \text{Spec}$. We conclude bound_b -inclusion by Def. 144(1) and L_b -, stop_b -, and dead_b -inclusion by Lem. 146. Now Thm. 90 implies $\text{Impl} \sqsubseteq_{b,conf} \text{Spec}$, and this, in turn, also shows that $\text{Impl} \sqsubseteq_{\mathcal{F}_{b,fin}^+} \text{Spec}$ implies $\text{Impl} \sqsubseteq_{b,conf}^c \text{Spec}$ with Thm. 150 and the definition of $\sqsubseteq_{b,conf}^c$. \square

Example 153 We already showed in Ex. 50 that for the open nets S and S' , $S' \sqsubseteq_{b,conf}^c S$ does not hold. We can now confirm this with Thm. 152, because S' does not $\mathcal{F}_{b,fin}^+$ -refine S by Ex. 145. \diamond

With Thm. 97, we have characterized the b -conformance relation for a variant of responsiveness (i.e., b -responsiveness) as introduced in Def. 41, and with Thm. 152 the coarsest precongruence that is contained in that relation (i.e., compositional b -conformance). In contrast to conformance and compositional conformance, b -conformance and compositional b -conformance are decidable. We already developed a decision procedure for b -conformance in Sect. 5.2. In the following section, we present a decision procedure for compositional b -conformance.

6.2 DECIDING COMPOSITIONAL b -CONFORMANCE

In this section, we show that compositional b -conformance is decidable. As a first step, we show in Sect. 6.2.1 how to decide \mathcal{F}_{fin}^+ -refinement for two

finite LTSs, thereby generalizing the construction of Rensink and Vogler [217, Theorem 61] for deciding \mathcal{F}^+ -refinement. In the second step in Sect. 6.2.2, we encode the set of fintree failures $\mathcal{F}_{b,fin}^+$ of a labeled net N into a finite LTS $BEH_b(N)$. To this end, we combine the decision procedure from Sect. 5.2 with deciding whether the finite LTS $BEH_b(Impl)$ \mathcal{F}_{fin}^+ -refines the finite LTS $BEH_b(Spec)$. That way, we can conclude decidability of $\mathcal{F}_{b,fin}^+$ -refinement.

6.2.1 Deciding \mathcal{F}_{fin}^+ -refinement for finite LTSs

For an LTS S with final states, we can define $\mathcal{F}_{fin}^+(S)$ and \mathcal{F}_{fin}^+ -refinement in the same way as for labeled nets in Def. 63 and Def. 69. We already showed in Sect. 4.3 that \mathcal{F}_{fin}^+ -refinement is in general undecidable for labeled nets. However, in this section, we show that \mathcal{F}_{fin}^+ -refinement is decidable for two finite LTSs.

We assume two finite LTSs $Impl$ and $Spec$ with identical alphabet Σ and initial states p_0 and q_0 such that $L(Impl) \subseteq L(Spec)$. For convenience, and only in the remainder of this chapter, we speak of a finite automaton instead of a finite LTS because the automata-theoretic notion of language coincides with our notion of the language of an LTS if we regard the transition label τ as the empty word ε and consider all states as accepting states (see Sect. 2.2 for more details). For an automaton A with some state s (we write $s \in A$), $L_A(s)$ denotes the language of the automaton if we change the initial state to s . We call a state *productive*, if it lies on a path from the initial state to some accepting state—that is, if it is used by the automaton when accepting a word.

As a first step, we extend $Impl$ to an automaton of automata pairs AA by adding a family of pairs of deterministic automata (A_p^1, A_p^2) , $p \in Impl$, such that for every $p \in Impl$ the language of A_p^1 is the set $\Sigma^* \setminus L_{Impl}(p)$ of traces that $Impl$ cannot perform from p , and the language of A_p^2 is the set $\Sigma^* \setminus L_{Impl}^\Omega(p)$ with $L_{Impl}^\Omega(p) = \{w \mid p \xrightarrow{w} p' \wedge p' \in \Omega_{Impl}\}$. The following holds by Def. 63:

$$(w, X, Y) \in \mathcal{F}_{fin}^+(Impl) \text{ iff } \exists p_0 \xrightarrow{w}_{AA} p : X \subseteq L(A_p^1) \wedge Y \subseteq L(A_p^2).$$

The automaton AA represents some fintree failures $(w, X, Y) \in \mathcal{F}_{fin}^+(Impl)$ directly in the sense that there is a $p \in Impl$ with $p_0 \xrightarrow{w}_{AA} p$ and $X = L(A_p^1)$ and $Y = L(A_p^2)$; in particular, it represents all *maximal* fintree failures—that is, all those $(w, X, Y) \in \mathcal{F}_{fin}^+(Impl)$ where extending X or Y cannot give another element in $\mathcal{F}_{fin}^+(Impl)$. Note that in a finite LTS $Impl$, there exists for each fintree failure $(v, X', Y') \in \mathcal{F}_{fin}^+(Impl)$ a maximal fintree failure $(v, X, Y) \in \mathcal{F}_{fin}^+(Impl)$ with $X' \subseteq X$ and $Y' \subseteq Y$.

Example 154 As an example, consider the LTS $Impl$ in Fig. 84a. It consists of five states p_0 to p_4 , among them the only final state p_4 , and four transitions. If we extend $Impl$ to the automaton AA described above, then AA has the same structure as $Impl$ except that every state p of AA is an accepting state and consists of two finite, deterministic automata A_p^1 and A_p^2 . For the initial state p_0 of AA , we depict its two automata $A_{p_0}^1$ and $A_{p_0}^2$ in Fig. 84b and Fig. 84c. The automaton AA encodes the set of fintree failures \mathcal{F}_{fin}^+ of $Impl$. For example, we have $(\varepsilon, \{aa\}, \{aa, ab\}) \in \mathcal{F}_{fin}^+(Impl)$: $Impl$ refuses

the word aa after ε (i.e., from its initial state p_0). In addition, neither aa nor ab leads to a final state of $Impl$ from p_0 ; while aa cannot be performed at all, ab leads to the nonfinal state p_3 . From the initial state p_0 , $Impl$ cannot refuse ab . Consequently, aa is an accepted word of both $A_{p_0}^1$ and $A_{p_0}^2$, but ab is only accepted by $A_{p_0}^2$ and not by $A_{p_0}^1$. \diamond

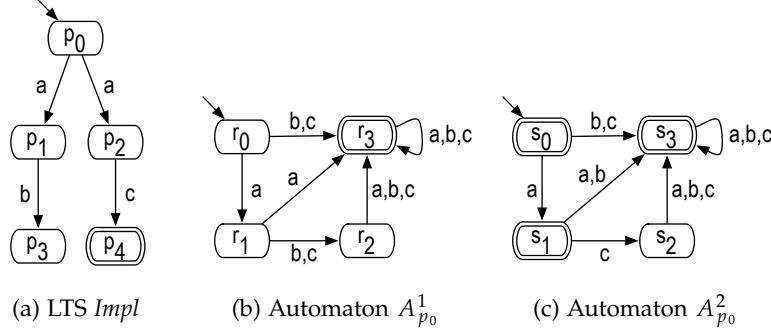


Figure 84: Sketch of the construction of the automaton AA for the LTS $Impl$. We depict a final state of an LTS and an accepting state of an automaton by a doubled frame.

Similarly, we construct an automaton of automata pairs for $Spec$, but this time, we additionally make $Spec$ deterministic more or less by the usual powerset construction on automata [224, 123]. This results in a deterministic automaton of automata pairs BB , which is a deterministic automaton extended with a family BB_Q , $Q \in BB$. For each state Q (being a set of states of $Spec$), BB_Q is a set of pairs of deterministic automata (B_q^1, B_q^2) , $q \in Q$, with $L(B_q^1) = \Sigma^* \setminus L_{Spec}(q)$ and $L(B_q^2) = \Sigma^* \setminus L_{Spec}^\Omega(q)$.

More in detail, the automaton part of BB is defined as follows: The initial state of BB is $Q_0 = \{q \mid q_0 \xrightarrow{\tau}_{Spec} q\}$; the transition relation is defined by $Q \xrightarrow{a}_{BB} Q'$ ($a \in \Sigma$) if $Q' = \{q' \mid \exists q \in Q : q \xrightarrow{a}_{Spec} q'\}$. We restrict BB to the nonempty states reachable from Q_0 and let each state of BB be accepting. As a consequence, all states of BB are productive and $L(BB) = L(Spec)$. This way, we have

$$Q_0 \xrightarrow{w}_{BB} Q \text{ iff } Q = \{q \mid q_0 \xrightarrow{w}_{Spec} q\} \text{ for all } w \in \Sigma^*$$

and

$$(w, X, Y) \in \mathcal{F}_{fin}^+(Spec) \text{ iff } \exists Q_0 \xrightarrow{w}_{BB} Q, (B_q^1, B_q^2) \in BB_Q : \\ X \subseteq L(B_q^1) \wedge Y \subseteq L(B_q^2).$$

First, we construct the following partial product automaton S , which can also be seen as the minimal simulation relation from AA to BB . It is well-defined because BB is deterministic by construction:

- $(p_0, Q_0) \in S$ is the initial state of S , and all states are accepting.
- If $(p, Q) \in S$, $a \in \Sigma$ and $p \xrightarrow{a}_{AA} p'$, then by language inclusion and definition of BB , there is a unique $Q' \in BB$ such that $Q \xrightarrow{a}_{BB} Q'$; we add (p', Q') and the transition $(p, Q) \xrightarrow{a} (p', Q')$ to S .
- If $(p, Q) \in S$ and $p \xrightarrow{\tau}_{AA} p'$, then we add (p', Q) and the transition $(p, Q) \xrightarrow{\tau} (p', Q)$ to S (recall that BB is τ -free).

Checking \mathcal{F}_{fin}^+ -refinement in Def. 69 on LTSs entails checking whether for all $(w, X, Y) \in \mathcal{F}_{fin}^+(Impl)$ with $X \cup Y \neq \emptyset$, we have $(wu, u^{-1}X, u^{-1}Y) \in \mathcal{F}_{fin}^+(Spec)$ for some $u \in \downarrow(X \cup Y)$. Recall that by language inclusion we do not have to check triples $(w, \emptyset, \emptyset)$. We have to check for each $(p, Q) \in S$ and each pair (X, Y) with $X \subseteq L(A_p^1)$, $Y \subseteq L(A_p^2)$, and $X \cup Y \neq \emptyset$ that

$$\begin{aligned} \exists u \in \downarrow(X \cup Y), Q' \in BB, (B_{q'}^1, B_{q'}^2) \in BB_{Q'} : \\ Q \xrightarrow{u}_{BB} Q' \wedge u^{-1}X \subseteq L(B_{q'}^1) \wedge u^{-1}Y \subseteq L(B_{q'}^2). \end{aligned} \quad (1)$$

Let us fix (p, Q) ; we now show how to check (1) for all suitable (X, Y) . This means that we have to compare runs in A_p^1 or A_p^2 , for u in (1), with runs of BB . To do this, we construct another (partial) product automaton P , similar to the previous one, but this time between the automata A_p^1, A_p^2 (whose initial states we also denote by p) and BB where the initial state is changed to Q . Another difference with the previous case is that, this time, we do not necessarily have $L(A_p^1) \cup L(A_p^2) \subseteq L_{BB}(Q)$ —that is, BB might not be able to simulate both, A_p^1 and A_p^2 —but still we want to represent all of $L(A_p^1) \cup L(A_p^2)$ to check the inclusion in (1). Furthermore, P (as also the derived subautomata R below) has two types of accepting states, called 1-accepting and 2-accepting. With $L^i(P)$, we denote the language of P when i -accepting states are considered to be accepting states, for $i = 1, 2$.

Therefore, P is constructed as follows (here $*$ is a dummy element, not appearing anywhere else):

- $(p, p, Q) \in P$ is the initial state;
- if $(p_1, p_2, Q') \in P$ and $p_1 \xrightarrow{a}_{A_p^1} p'_1$ or $p_2 \xrightarrow{a}_{A_p^2} p'_2$ (implying $p_1 \neq * \neq p'_1$ or $p_2 \neq * \neq p'_2$), we add state (p''_1, p''_2, Q'') and the transition $(p_1, p_2, Q') \xrightarrow{a} (p''_1, p''_2, Q'')$ where
 - p''_i is p'_i if $p_i \xrightarrow{a}_{A_p^i} p'_i$ and $*$ otherwise (in particular if $p_i = *$) for $i = 1, 2$
 - Q'' satisfies $Q' \xrightarrow{a}_{BB} Q''$ (in particular, $Q' \neq *$) or is $*$ otherwise
- (p_1, p_2, Q') is i -accepting if p_i is accepting in A_p^i , $i = 1, 2$.

Because the A_p^i and BB are deterministic, P is also deterministic, and we have $L^i(P) = L(A_p^i)$ for $i = 1, 2$ by construction. We will call R a *productive subautomaton* of P , if R is obtained from P by restricting all components (in particular also the accepting states) to a subset M of the state set such that each state of R is productive in R . We will show that (1) is satisfied for all suitable (X, Y) if and only if for each productive subautomaton R of P

$$\begin{aligned} \exists (p_1, p_2, Q') \in R, Q' \in BB, (B_{q'}^1, B_{q'}^2) \in BB_{Q'} : \\ L_R^i((p_1, p_2, Q')) \subseteq L(B_{q'}^i), \text{ for } i = 1, 2. \end{aligned} \quad (2)$$

The latter is clearly decidable because the number of productive subautomata R of P is finite and, thus, we have to check only finitely many inclusions on finite automata in (2). Because (2) is decidable, it then follows that \mathcal{F}_{fin}^+ -refinement of two finite LTSs is decidable, too. Note that $Q' \in BB$ in (2) is equivalent to $Q' \neq *$.

So assume (1) is satisfied for all suitable (X, Y) and, thus, in particular for $(L(A_p^1), L(A_p^2))$. If R is a productive subautomaton, then $L^1(R) \subseteq L^1(P)$

and $L^2(R) \subseteq L^2(P)$ and $L^1(R) \cup L^2(R) \neq \emptyset$. Hence, due to (1) and the construction of P , there exists a $u \in \downarrow L^1(R) \cup \downarrow L^2(R)$, $Q' \in BB, (B_{q'}^1, B_{q'}^2) \in BB_{Q'}$ such that $Q \xrightarrow{u}_{BB} Q'$ and $u^{-1}L^i(R) \subseteq L(B_{q'}^i)$, for $i = 1, 2$. Then $(p, p, Q) \xrightarrow{u}_R (p_1, p_2, Q')$ for some p_1, p_2 . Because R is deterministic, the state (p_1, p_2, Q') is uniquely determined by u and, therefore, $u^{-1}L^i(R) = L_R^i((p_1, p_2, Q'))$, for $i = 1, 2$. Thus, we have (p_1, p_2, Q') and $(B_{q'}^1, B_{q'}^2)$ are the state and automaton pair whose existence is asserted in (2).

Vice versa, assume that (2) holds for each productive subautomaton R and take some $X \subseteq L(A_p^1)$ and $Y \subseteq L(A_p^2)$ with $X \cup Y \neq \emptyset$. The set of states that are needed in P to accept the words of $X \cup Y$ (recall the construction of P) defines a productive subautomaton R with $X \subseteq L^1(R)$ and $Y \subseteq L^2(R)$. Take $(p_1, p_2, Q') \in R$ and $(B_{q'}^1, B_{q'}^2) \in BB_{Q'}$ that satisfy (2). Then there is some $u \in \downarrow X \cup \downarrow Y$ with $(p, p, Q) \xrightarrow{u}_R (p_1, p_2, Q')$ by choice of R and $Q \xrightarrow{u}_{BB} Q'$ by construction of P and because $Q' \in BB$. Now $u^{-1}X \subseteq u^{-1}L^1(R) = L_R^1((p_1, p_2, Q'))$ and $u^{-1}Y \subseteq u^{-1}L^2(R) = L_R^2((p_1, p_2, Q'))$ by determinism of R , and we can conclude that $u^{-1}X \subseteq L(B_{q'}^1)$ and $u^{-1}Y \subseteq L(B_{q'}^2)$.

Therefore, we have shown:

Proposition 155

Checking \mathcal{F}_{fin}^+ -refinement for two finite LTSs is decidable.

Next, we reduce $\mathcal{F}_{b,fin}^+$ -refinement of two labeled nets to \mathcal{F}_{fin}^+ -refinement on two finite LTS.

6.2.2 Reducing the decision of compositional b -conformance to \mathcal{F}_{fin}^+ -refinement

In this section, we shall prove decidability of $\mathcal{F}_{b,fin}^+$ -refinement (and therefore compositional b -conformance by Thm. 152) for two interface-equivalent open nets *Impl* and *Spec*.

Checking $\mathcal{F}_{b,fin}^+$ -refinement entails checking both items of Def. 144. The first item of Def. 144—that is, checking $bound_b$ -inclusion—is decidable because we can represent the language $bound_b(N)$ of an open net N as a finite LTS $BSD_b(N)$ by Cor. 100. Thus, we can check whether $bound_b(Impl) \subseteq bound_b(Spec)$ by checking $L_0(BSD_b(Impl)) \subseteq L_0(BSD_b(Spec))$, which is clearly decidable.

To decide refinement of the fintree failures in $\mathcal{F}_{b,fin}^+$ —that is, the second item of Def. 144—it suffices to reduce this to checking \mathcal{F}_{fin}^+ -refinement for two finite LTS, which is decidable by Prop. 155. The LTS $BSD_b(N)$ is not suitable for representing $\mathcal{F}_{b,fin}^+(N)$ of an open net N : Although $BSD_b(N)$ represents, among others, the language $L_b(N)$ by Cor. 100, we cannot distinguish the $bound_b$ -violators that can be performed (i.e., $bound_b$ -violators in $L_b(N)$) from those that have only been added as continuations. Thus, $BSD_b(N)$ cannot properly represent the refusal and fin-refusal sets of N . Therefore, we propose a finite-state representation of $\mathcal{F}_{b,fin}^+(Impl)$ and $\mathcal{F}_{b,fin}^+(Spec)$, respectively, on which checking \mathcal{F}_{fin}^+ -refinement coincides with checking $\mathcal{F}_{b,fin}^+$ -refinement for *Impl* and *Spec*. A finite-state representation of $\mathcal{F}_{b,fin}^+(N)$ of an open net N is the essential *behavior* of N .

Definition 156 [labeled transition system BEH_b]

Let N be a labeled net such that m_N is b -bounded. We define the labeled transition system $BEH_b(N) = (Q, \delta, m_N, \Sigma^{in}, \Sigma^{out}, \Omega)$ with

- $Q = \{\mathbb{U}_1, \mathbb{U}_2\} \uplus Q'$ where $Q' = \{m \in M_N \mid m \text{ is } b\text{-bounded in } N\}$,
- $\delta = \{(m, x, m') \in Q' \times (\Sigma \uplus \{\tau\}) \times Q' \mid$
 $\exists t \in T_N : m \xrightarrow{t} m' \wedge l(t) = x\}$
 $\uplus \{(m, x, \mathbb{U}_1), (m, x, \mathbb{U}_2) \in Q' \times (\Sigma \uplus \{\tau\}) \times \{\mathbb{U}_1, \mathbb{U}_2\} \mid$
 $\exists t \in T_N : \exists m' \in M_N \setminus Q' : m \xrightarrow{t} m' \wedge l(t) = x\}$
 $\uplus \{(\mathbb{U}_1, x, \mathbb{U}_1), (\mathbb{U}_1, x, \mathbb{U}_2) \mid x \in \Sigma\}, \text{ and}$
- $\Omega = \{m \in Q' \mid m \in \Omega_N\} \uplus \{\mathbb{U}_1\}$.

We restrict $BEH_b(N)$ to the states and transitions that are reachable from the initial state m_N . For an open net N , we define $BEH_b(N) = BEH_b(env(N))$.

For a labeled net N , $BEH_b(N)$ comprises the reachability graph $RG(N)$ but merges all markings of N that are reachable by a $bound_b$ -violation of N into the state \mathbb{U}_1 . That way, $BEH_b(N)$ is finite but, in contrast to $BSD_b(N)$ in Def. 99, not τ -free. The state \mathbb{U}_1 has a loop for every symbol of N 's alphabet; intuitively, \mathbb{U}_1 models the $bound_b$ -violators of N . The state \mathbb{U}_1 and its incoming transitions are then duplicated to the state \mathbb{U}_2 and its incoming transitions, except for \mathbb{U}_1 's self-loops. Therefore, the state \mathbb{U}_2 also models the set $bound_b(N)$. The set of final states of $BEH_b(N)$ consists of the set of (unmerged) final markings of N and the state \mathbb{U}_1 ; the state \mathbb{U}_2 is not a final state of $BEH_b(N)$. We can distinguish strict $bound_b$ -violators from $bound_b$ -violators of N in $BEH_b(N)$: Every trace of $BEH_b(N)$ that reaches \mathbb{U}_2 without passing through \mathbb{U}_1 is a strict $bound_b$ -violation of N , and all $bound_b$ -violators of N are prefixes of these strict $bound_b$ -violators. We can identify strict $bound_b$ -violators because \mathbb{U}_2 refuses any set of traces.

Example 157 Figure 85 sketches the construction of $BEH_1(D')$ from the open net D' in Fig. 82b. The set of final markings of D' is $\{[]\}$; thus, $BEH_1(D')$ has the final states $[]$ and \mathbb{U}_1 . We already detailed in Ex. 139 that $(s, \{f\}, \emptyset) \in \mathcal{F}_{\tau, fin}^+(D')$. The fintree failure $(s, \{f\}, \emptyset)$ is also reflected in $BEH_1(D')$: After the trace s , $BEH_1(D')$ can refuse f (because no transition in $BEH_1(D')$ is labeled with f) and fin-refuse \emptyset .

Not all fintree failures of the 1-bounded \mathcal{F}_{fin}^+ -semantics of D' are captured by $BEH_1(D')$: As an example, consider the fintree failure $(q, \{qf\}, \{q\}) \in \mathcal{F}_{b, fin}^+(D')$. Using trace q , $BEH_1(D')$ always reaches the state $[p_1, q^i]$. $BEH_1(D')$ can refuse the trace qf because no transition of $BEH_1(D')$ is labeled with f . However, $BEH_1(D')$ cannot fin-refuse q from $[p_1, q^i]$ because the final state \mathbb{U}_1 is reachable. Note that q is not a $bound_1$ -violation of D' . In fact, the reason why $(q, \{qf\}, \{q\})$ is not captured by $BEH_1(D')$ is that qq (i.e., a continuation of q with a trace from the fin-refusal set) is a strict $bound_1$ -violation of D' and, therefore, always reaches the final state \mathbb{U}_1 . \diamond

The next lemma gives four observations about $BEH_b(N)$ of a labeled net N . The first item states that the states \mathbb{U}_1 and \mathbb{U}_2 model $bound_b(N)$ —that is, an observation we already explained after Def. 156. The second item states that every $finbound_b$ -violation (w, X, Y) of N is a fintree failure of $BEH_b(N)$.

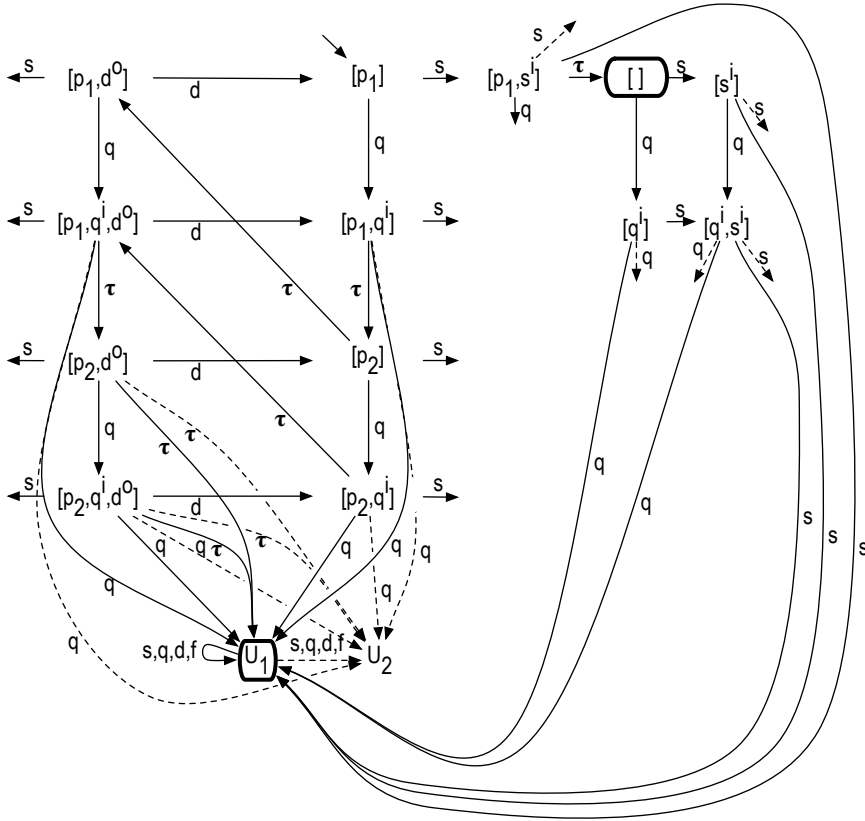


Figure 85: Sketch of the finite LTS $BEH_1(D')$ from the open net D' in Fig. 82b. The two final states \mathbb{U}_1 and $[]$ are depicted with a thick frame. In general, a transition without sink leads to a state not shown, with the exception that dashed transitions always lead to state \mathbb{U}_2 .

This directly follows from \mathbb{U}_2 modeling $bound_b(N)$, because we can reach \mathbb{U}_2 with w and \mathbb{U}_2 can refuse and fin-refuse everything. Note that it is not possible to model every $finbound_b$ -violator of N with \mathbb{U}_1 in $BEH_b(N)$ because \mathbb{U}_1 is a final state of $BEH_b(N)$. The third item states that every fintree failure of $BEH_b(N)$'s \mathcal{F}_{fin}^+ -semantics is also a fintree failure of N 's b -bounded \mathcal{F}_{fin}^+ -semantics. Finally, the fourth item states that the only fintree failures (w, X, Y) of N 's b -bounded \mathcal{F}_{fin}^+ -semantics that are not captured by $BEH_b(N)$ are “prefixes” of N 's $bound_b$ -violators. That way, we can conclude that (w, X, Y) is dominated by some fintree failure captured by $BEH_b(N)$ because of \mathbb{U}_2 modeling $bound_b(N)$ and refusing (fin-refusing) everything. An example for such fintree failures is $(q, \{qf\}, \{q\}) \in \mathcal{F}_{b,fin}^+(D')$ from Ex. 157.

Lemma 158

Let N be a labeled net such that m_N is b -bounded. Then the following facts hold for $BEH_b(N)$:

1. $w \in bound_b(N)$ iff $m_N \xRightarrow{w} \mathbb{U}_1$ iff $m_N \xRightarrow{w} \mathbb{U}_2$.
2. $finbound_b(N) \subseteq \mathcal{F}_{fin}^+(BEH_b(N))$.
3. $\mathcal{F}_{fin}^+(BEH_b(N)) \subseteq \mathcal{F}_{b,fin}^+(N)$.

4. Let $(w, X, Y) \in \mathcal{F}_{b, \text{fin}}^+(N) \setminus \mathcal{F}_{\text{fin}}^+(BEH_b(N))$. Then there exists an $u \in (\downarrow X \cup \downarrow Y) \setminus X$ such that $wu \in \text{bound}_b(N)$.

Proof. (1) follows immediately from the definition of $BEH_b(N)$, and (2) is an implication of (1) because \mathbb{U}_2 can refuse all $X \subseteq \Sigma^+$ and fin-refuse all $Y \subseteq \Sigma^*$.

(3) The sets agree on the fintree failures $(w, X, Y) \in \text{finbound}_b(N)$ by (2). So consider $w \notin \text{bound}_b(N)$. If $(w, X, Y) \in \mathcal{F}_{\text{fin}}^+(BEH_b(N))$ due to $m_N \xrightarrow{w} m$, then we also have $m_N \xrightarrow{w} m$ in N with the same underlying run. In $BEH_b(N)$, m could only have more traces (possibly to final states) due to runs using \mathbb{U}_1 , so it can only refuse and fin-refuse less. Thus, $(w, X, Y) \in \mathcal{F}_{\text{fin}}^+(N)$ and inclusion follows.

(4) If $(w, X, Y) \in \mathcal{F}_{b, \text{fin}}^+(N)$ due to m , but $(w, X, Y) \notin \mathcal{F}_{\text{fin}}^+(BEH_b(N))$, then this must be due to a transition sequence from m that passes through \mathbb{U}_1 ; assume this happens for the first time after $u \neq \varepsilon$; that is, we have $m_N \xrightarrow{w} m \xrightarrow{u} \mathbb{U}_1 \xrightarrow{u'} m'$ with $uu' \in X \cup Y$. Thus, $wu \in \text{bound}_b(N)$ by (1) and $u \in \downarrow X \cup \downarrow Y$. Because $m_N \xrightarrow{w} m \xrightarrow{u} m'$ also in N , we further have $u \notin X$. \square

With the next lemma, we show that deciding $\mathcal{F}_{b, \text{fin}}^+$ -refinement for two interface-equivalent open nets Impl and Spec reduces to checking $\mathcal{F}_{\text{fin}}^+$ -refinement of the LTSs $BEH_b(\text{Impl})$ and $BEH_b(\text{Spec})$. The proof idea incorporates that checking bound_b -inclusion is decidable using, for example, $\text{BSD}_b(\text{Impl})$ and $\text{BSD}_b(\text{Spec})$ from Chap. 5. Based on bound_b -inclusion, it is easy to see that every finbound_b -violation of Impl is also a finbound_b -violation of Spec . Then, we show that every “true” (i.e., not in $\text{finbound}_b(\text{Impl})$) fintree failure of Impl 's b -bounded $\mathcal{F}_{\text{fin}}^+$ -semantics is either captured by $BEH_b(\text{Impl})$ (and, therefore, dominated by a fintree failure in $BEH_b(\text{Spec})$) or not captured by $BEH_b(\text{Impl})$ (and, therefore, dominated by a finbound_b -violation of Spec). In either case, this implies a domineering fintree failure in Spec 's b -bounded $\mathcal{F}_{\text{fin}}^+$ -semantics.

Lemma 159

For two interface-equivalent open nets Impl and Spec with $\text{bound}_b(\text{Impl}) \subseteq \text{bound}_b(\text{Spec})$, we have

$$\text{Impl} \sqsubseteq_{\mathcal{F}_{b, \text{fin}}^+} \text{Spec} \text{ iff } BEH_b(\text{Impl}) \sqsubseteq_{\mathcal{F}_{\text{fin}}^+} BEH_b(\text{Spec}).$$

Proof. \Rightarrow : Let $(w, X, Y) \in \mathcal{F}_{\text{fin}}^+(BEH_b(\text{Impl}))$. We have $(w, X, Y) \in \mathcal{F}_{b, \text{fin}}^+(\text{Impl})$ by Lem. 158(3) and (w, X, Y) is dominated by a fintree failure of Spec by Def. 144: There exists $x \in \{\varepsilon\} \cup \downarrow X \cup \downarrow Y$ such that $(wx, x^{-1}X, x^{-1}Y) \in \mathcal{F}_{b, \text{fin}}^+(\text{Spec})$. If $(wx, x^{-1}X, x^{-1}Y) \in \mathcal{F}_{\text{fin}}^+(BEH_b(\text{Spec}))$, we are done. So assume otherwise and consider $u \in (\downarrow x^{-1}X \cup \downarrow x^{-1}Y) \setminus x^{-1}X$ with $wxu \in \text{bound}_b(\text{Spec})$ according to Lem. 158(4). Then $xu \in (\downarrow X \cup \downarrow Y) \setminus X$ by set theory. Therefore, $(wxu, (xu)^{-1}X, (xu)^{-1}Y) \in \mathcal{F}_{\text{fin}}^+(BEH_b(\text{Spec}))$ by Def. 63 and $\varepsilon \notin (xu)^{-1}X$. Hence, the fintree failure (w, X, Y) is dominated by $(wxu, (xu)^{-1}X, (xu)^{-1}Y)$ and, thus, $BEH_b(\text{Impl}) \sqsubseteq_{\mathcal{F}_{\text{fin}}^+} BEH_b(\text{Spec})$.

\Leftarrow : Let $(w, X, Y) \in \mathcal{F}_{b, \text{fin}}^+(\text{Impl})$. If $(w, X, Y) \in \mathcal{F}_{\text{fin}}^+(BEH_b(\text{Impl}))$, then (w, X, Y) is dominated by some $(wx, x^{-1}X, x^{-1}Y) \in \mathcal{F}_{\text{fin}}^+(BEH_b(\text{Spec}))$ with

$x \in \{\varepsilon\} \cup \downarrow X \cup \downarrow Y$ by assumption. We also have $(wx, x^{-1}X, x^{-1}Y) \in \mathcal{F}_{b,fin}^+(Spec)$ by Lem. 158(3), and therefore $Impl \sqsubseteq_{\mathcal{F}_{b,fin}^+} Spec$. So consider $(w, X, Y) \in \mathcal{F}_{b,fin}^+(Impl) \setminus \mathcal{F}_{fin}^+(BEH_b(Impl))$. By Lem. 158(4), there exists $u \in (\downarrow X \cup \downarrow Y) \setminus X$ such that $wu \in bound_b(Impl)$. Because of $bound_b$ -inclusion and $\varepsilon \notin u^{-1}X$, we have $u \in \{\varepsilon\} \cup \downarrow X \cup \downarrow Y$ and $(wu, u^{-1}X, u^{-1}Y) \in \mathcal{F}_{b,fin}^+(Spec)$ Def. 138. Thus, $Impl \sqsubseteq_{\mathcal{F}_{b,fin}^+} Spec$. \square

With Lem. 159, the fact that $bound_b$ -inclusion is decidable, and Prop. 155, we have shown the main result of this section: We can decide whether an open net $Impl$ b -refines an open net $Spec$, and, thus, decide whether $Impl$ compositionally b -conforms to $Spec$ by Thm. 152.

Theorem 160 [$\mathcal{F}_{b,fin}^+$ -refinement is decidable]

For two interface-equivalent open nets $Impl$ and $Spec$, checking whether $Impl \sqsubseteq_{\mathcal{F}_{b,fin}^+} Spec$ is decidable.

The construction of the labeled transition system $BEH_b(N)$ in Def. 156, the lemmata Lem. 158 and Lem. 159, and Thm. 160 can be generalized from a labeled net N with a given bound b to certain enhanced LTSs S : The state set of S is partitioned into $Q \uplus B(S)$ with a finite set Q of states (i.e., the set of b -bounded markings of N as in Def. 156) and a possibly infinite set $B(S)$ of *bad* states (i.e., the set of markings of N only reachable via bound b violations); the states in Q are reachable from the initial state q_S (i.e., m_N in N) without entering $B(S)$. For such an enhanced LTS S , we can define $\mathcal{F}_{fin}^+(S)$, \mathcal{F}_{fin}^+ -refinement, $bound_b(S)$, $finbound_b(S)$, $\mathcal{F}_{b,fin}^+(S)$ and $\mathcal{F}_{b,fin}^+$ -refinement as for labeled nets in Def. 63, Def. 69, Def. 84, Def. 138 and Def. 144. Still, Thm. 160 holds; that is, $\mathcal{F}_{b,fin}^+$ -refinement for two enhanced LTSs is decidable by reducing it to \mathcal{F}_{fin}^+ -refinement for two finite LTSs. In this context, the reachability graph $RG(N)$ of a labeled net N together with the set $B(N) \subseteq M_N$ of all markings that are reachable only via bound b violations of N —that is, the essence of Def. 156—is just a special case. Nevertheless, for readability reasons and because the generalization is straight-forward, we only presented the special case in this section.

6.3 CONCLUSIONS

In this chapter, we investigated the coarsest precongruence that is contained in the b -conformance relation—that is, compositional b -conformance. We characterized compositional b -conformance providing a failure-based semantics for open nets. To this end, we added information about $bound_b$ -violations to the coarsest precongruence that is contained in the conformance relation. Based on our characterization, we prove compositional b -conformance to be decidable: The problem could be reduced to deciding should testing [217], if we refine the proof in [217] by further details. The decision procedure presented in this chapter does not depend on Petri nets but is independent from the concrete model.

CONCLUSIONS AND RELATED WORK

IN this chapter, we summarize the results from Part II. We compare the compositional conformance and compositional b -conformance and classify both into the linear time - branching time spectrum of known preorders between systems in Sect. 7.2. Finally, we review related work in Sect. 7.3.

7.1 OVERVIEW OF THE RESULTS

We studied a conformance preorder describing whether an open system can safely be replaced by another open system, thereby guaranteeing responsiveness of the overall system. The latter guarantees the permanent possibility to either mutually communicate or mutually terminate. In Chap. 3, we showed that responsiveness can be seen as a minimal correctness criterion for open systems. It implies deadlock freedom but does not imply weakly termination. Besides responsiveness, we also investigated b -responsiveness. The latter requires responsiveness and additionally b -boundedness of the composition due to maintaining a previously known message bound b . The resulting conformance preorder for each variant of responsiveness is the conformance relation and the b -conformance relation, respectively.

Our goal was to analyze the conformance relation and the b -conformance relation for compositionality and decidability. To facilitate this analysis, we characterized both relations using certain denotational semantics for open nets. Figure 86 illustrates the general schema we employed: First, we provided a denotational semantics for open nets and a refinement relation upon this semantics. Then, we showed that this refinement relation coincides with the respective conformance relation. That way, we developed a characterization of the conformance relation.

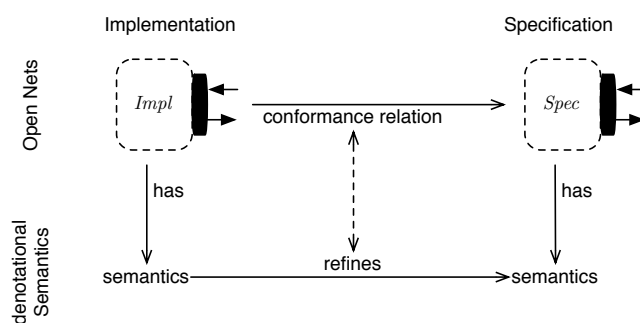


Figure 86: The general schema we employed to characterize a conformance relation using denotational semantics for open nets. A solid arc illustrates the relation described by the corresponding arc label. The dashed arc illustrates logical equivalence.

Table 8 recalls the structure of Part II. For each variant of the conformance preorder, we presented a characterization based on a denotational semantics of open nets. For conformance in Chap. 4, the semantics, called *stopdead*-semantics, consist of two sets collecting completed traces and unsuccessfully completed traces. For b -conformance in Chap. 5, we had to add the language and a set of uncoverable traces collecting catastrophic traces

that cannot be used reliably. The resulting semantics was the *b-coverable stopdead-semantics*.

relation	characterization	compositionality	decidability
conformance	Chap. 4	Chap. 4	Chap. 4
<i>b</i> -conformance	Chap. 5	Chap. 6	Chap. 5 & Chap. 6

Table 8: The structure of Part II without this chapter.

We showed that neither conformance nor *b*-conformance is a precongruence and characterized the coarsest precongruence that is contained in the respective preorder—that is, compositional conformance in Chap. 4 and compositional *b*-conformance in Chap. 6. In the unbounded setting, we showed in Chap. 4 that conformance and compositional conformance are undecidable. This motivates our focus on *b*-responsiveness instead of responsiveness. For the latter, we proved decidability of *b*-conformance in Chap. 5 and compositional *b*-conformance in Chap. 6. In addition, we elaborate a finite characterization of all *b*-conforming open nets for a given open net in Chap. 5.

7.2 CLASSIFYING COMPOSITIONAL CONFORMANCE AND COMPOSITIONAL *b*-CONFORMANCE

In this section, we study the relation between compositional conformance and compositional *b*-conformance. We already showed in Sect. 3.3.2 that conformance and *b*-conformance are incomparable. In the following, we use two examples to show that also compositional conformance and compositional *b*-conformance are incomparable. The first example is used to show that compositional conformance does not imply compositional *b*-conformance.

Example 161 Figure 87 shows the two open nets N_4 and N_5 from Sect. 3.3.2. Every fintree failure of the \mathcal{F}_{fin}^+ -semantics of N_4 is also a fintree failure of the \mathcal{F}_{fin}^+ -semantics of N_5 , because every transition sequence of $env(N_4)$ is also a transition sequence in $env(N_5)$, leading to the same markings of $env(N_4)$ and $env(N_5)$ except for the place p_0 . A token on p_0 , in turn, does not enable or hinder any further transition. In other words, we have $\mathcal{F}_{fin}^+(N_4) \subseteq \mathcal{F}_{fin}^+(N_5)$ and, thus, N_4 compositionally conforms to N_5 by Def. 69 and Thm. 75. In contrast, N_4 does not compositionally *b*-conform to N_5 : We have $\varepsilon \notin bound_b(N_5)$ but $\varepsilon \in bound_b(N_4)$ because the place p_0 is unbounded in the composition of N_4 with any open net. Thus, N_4 does not $\mathcal{F}_{b,fin}^+$ -refine N_5 by Def. 144, and Thm. 152 shows the statement. \diamond

With the second example, we show that compositional *b*-conformance does not imply compositional conformance.

Example 162 Consider the open net N_6 in Fig. 87c. As in Ex. 54, we define the open net N_7 as the open net N_6 in Fig. 87c but with $\Omega_{N_7} = \{m \in Bags(P_{N_7}) \mid \forall p \in P_{N_7} \setminus \{p_0, p_1\} : m(p) = 0\}$ as its set of final markings. The open net N_6 compositionally *b*-conforms to N_7 : We have $bound_b(N_6) = bound_b(N_7)$ and $finbound_b(N_6) = finbound_b(N_7)$ because N_6 and N_7 differ only in their set of final markings. Thus, ev-

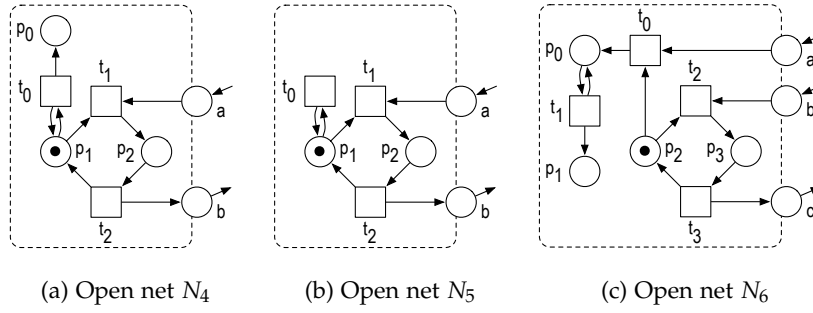


Figure 87: Three open nets from Fig. 38 proving that compositional conformance and compositional b -conformance are incomparable. In addition to the figures, we have $\Omega_{N_4} = \Omega_{N_5} = \Omega_{N_6} = \{\{\}\}$.

ery fintree failure (w, X, Y) of the b -bounded \mathcal{F}_{fin}^+ -semantics of N_6 that is not a fintree failure of the b -bounded \mathcal{F}_{fin}^+ -semantics of N_7 contains a trace $w \in Y$ that reaches a final marking of N_7 . By construction, a final marking of N_7 is only reachable by a $bound_b$ -violation of N_7 . Therefore, $(w, X, Y) \in \mathcal{F}_{b,fin}^+(N_6) \setminus \mathcal{F}_{b,fin}^+(N_7)$ implies $(w, X, Y) \in finbound_b(N_7)$ and, thus, $\mathcal{F}_{b,fin}^+(N_6) = \mathcal{F}_{b,fin}^+(N_7)$. Consequently, N_6 $\mathcal{F}_{b,fin}^+$ -refines N_7 and N_6 compositionally b -conforms to N_7 by Thm. 152.

However, N_6 does not compositionally conform to N_7 . As an example, consider the fintree failure $(a, \emptyset, \{\varepsilon\}) \in \mathcal{F}_{fin}^+(N_6)$. We have $(a, \emptyset, \{\varepsilon\}) \notin \mathcal{F}_{fin}^+(N_7)$ because we reach the final marking $[p_0]$ with trace a in $env(N_7)$. Therefore, N_6 does not \mathcal{F}_{fin}^+ -refine N_7 and does not compositionally conform to N_7 by Thm. 75. \diamond

With Ex. 161 and Ex. 162, we showed that compositional conformance and compositional b -conformance are incomparable. In the remainder of this section, we show how compositional conformance and compositional b -conformance relate to the known preorders from the linear time - branching time spectrum [104, 105].

Figure 88 depicts some of the known preorders from the linear time - branching time spectrum and the relations between them: bisimulation [202] (B), ready simulation [36] (RS), must testing [72] (MT), should (or fair) testing [196, 48, 217] (ST), completed trace (CT) and trace (T) preorder. An arrow (and a sequence of arrows) between two preorders denotes the inclusion relation; for example, the bisimulation preorder implies (is finer than) the ready simulation preorder. An absent arrow (or sequence of arrows) between two preorders indicates that the inclusion does not hold; for example, the should testing preorder is not finer than the must testing preorder.

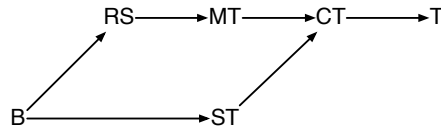


Figure 88: Some known preorders from the linear time - branching time spectrum.

Figure 89 depicts the classification of compositional conformance and compositional b -conformance into the linear time - branching time spectrum from Fig. 88. Compositional conformance is the should testing preorder [217] extended with traces that do not lead to a final marking. For

compositional b -conformance, we had to extend the should testing preorder by information about bound violations, which make compositional b -conformance incomparable to compositional conformance.

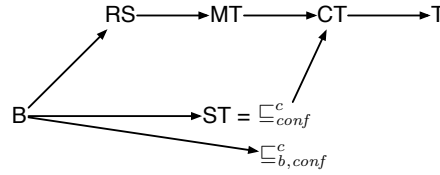


Figure 89: (Compositional) conformance and (compositional) b -conformance classified into the linear time - branching time spectrum.

If we extend bisimulation in Def. 5 to respect final states (i.e., two states q_1 and q_2 in a bisimulation relation have to satisfy: q_1 is a final state if and only if q_2 is a final state), then bisimulation implies compositional b -conformance. Compositional b -conformance does not imply trace-inclusion (i.e., the trace preorder): Consider the open nets N_8 and N_9 in Fig. 90. Every $bound_b$ -violation of N_8 is also a $bound_b$ -violation of N_9 , and every trace $w \in L(N_8) \setminus L(N_9)$ is a $bound_b$ -violation of N_9 . In addition, N_9 can refuse more traces than N_8 because of the missing transition t_2 . Therefore, N_8 compositionally b -conforms to N_9 , but we have, for example, $ab \in L(N_8)$ and $ab \notin L(N_9)$.

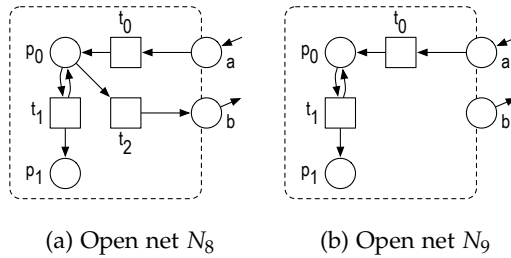


Figure 90: Two open nets proving that compositional b -conformance does not imply trace-inclusion. In addition to the figures, we have $\Omega_{N_8} = \Omega_{N_9} = \emptyset$.

7.3 RELATED WORK

In this section, we review work related to conformance checking, our denotational semantics for open nets, and the undecidability results.

The idea of assigning a formal semantics to a program for verification purposes was introduced by Floyd [96] and Hoare [118]. The intuition behind conformance checking derives from program refinement calculi [80, 184, 182, 24]. Other names for a conformance relation found in literature are refinement relation [256, 37], implementation relation [119, 72, 143, 238], conformation relation [81], preorder relation [68], accordance relation [226, 11], and subcontract relation [141, 44], for instance.

7.3.1 Work based on process algebra and declarative models

WORK OF RAMAJANI AND REHOF Rajamani and Rehof [213] define a conformance relation in a bisimulation-like style for the process algebra CCS [177]. Although they use a formal model different than ours, they also investigate asynchronous communication. Rajamani and Rehof [213] investi-

gate stuck-freeness—that is, the behavioral correctness property that guarantees that a message sent by a sender will not get stuck without some receiver ever receiving it, and that a receiver waiting for a message will not get stuck without some sender ever sending it. In contrast, we consider responsiveness and b -responsiveness, which are incomparable to stuck-freeness: On the one hand, a message may get stuck on a channel without being received for responsiveness, as long as the sender and the receiver continue communicating over other channels. On the other hand, stuck-freeness does not imply responsiveness because it allows to avoid getting stuck by repeatedly following internal transitions, which does not imply perpetual communication as needed for responsiveness.

WORK OF FOURNET ET AL. Fournet et al. [97] continue the work of Rajamani and Rehof [213] and present with stuck-free conformance a precongruence that excludes deadlocks. Their precongruence, like compositional conformance and compositional b -conformance, is based upon a variation of failures semantics rather than traces. However, stuckness is more discriminative than deadlock freedom by taking orphan messages into account. In contrast, responsiveness and b -responsiveness allow for orphaned messages if communication continues otherwise (i.e., over other channels).

WORK OF PADOVANI ET AL. Padovani et al. [141, 59] introduce the subcontract preorder for CCS-like [177] processes without τ -actions. In contrast, our model for open systems may contain internal transitions (i.e., τ -actions). Their subcontract preorder is equivalent to must testing [72] and therefore incomparable to compositional conformance and compositional b -conformance (see Fig. 89). As an additional difference, their subcontract preorder is an asymmetric notion; that is, it is focusing only on a successful termination of the test (i.e., the system’s environment), rather than on the system under test. In contrast, our notions of responsiveness and b -responsiveness are symmetric notions where both composed systems have to terminate successfully.

WORK OF BRAVETTI ET AL. Bravetti and Zavattaro [46] extend their previous work in [43, 44, 45] to asynchronously communicating processes and define the subcontract preorder which preserves weak termination. Our notions of conformance and b -conformance do not preserve weak termination—that is, our preorders are coarser. The model in [46] is a modified version of Milner’s CCS [177] with *one* unbounded but ordered message queue. In contrast, we use Petri nets with interface places (i.e., open nets) as a model, and *each* interface place models an unbounded unordered message queue.

WORK OF DILL Trace-based semantics like ours (in particular the b -bounded *stopdead*-semantics and the b -coverable *stopdead*-semantics in Chap. 5) where the language is flooded with error traces go back to the work of Dill [81]. Errors in [81] arise from communication mismatches and are similar to our $bound_b$ -violators. Dill’s semantics can be seen as a declarative model and Dill’s refinement relations, to which he refers as conformation, are trace inclusions like our characterizations of the conformance and b -conformance preorders. In contrast to our asynchronous (i.e., buffered) setting, Dill considers a synchronous (i.e., an unbuffered) setting. Similar decision procedures for preorders other than b -conformance have also been studied in [47].

7.3.2 *Work based on automata*

WORK OF DE ALFARO AND HENZINGER Interface automata, as defined by de Alfaro and Henzinger [18], take up the same ideas as Dill [81] but on an operational (i.e., automaton) model instead on a declarative one. In contrast to a refinement relation based on trace inclusions (like the refinement relations of Dill [81] and the refinement relation we used to characterize conformance and b -conformance in Chap. 4 and Chap. 5), refinement of interface automata is characterized by an alternating simulation relation similar to the refinement of modal transition systems [142]. Alternating simulation is conceptually more complex than refinement based on trace containment. Further, alternating simulation is overly strong in comparison to our refinements based on the *stopdead*-semantics and the b -coverable *stopdead*-semantics, which are the weakest preorders preserving responsiveness and b -responsiveness.

WORK OF CHILTON ET AL. Chilton et al. [62] (a preliminary version appeared as [61]) formulate a theory for components based on I/O automata [160, 129] augmented by an inconsistency predicate on states. I/O automata are conceptually similar to interface automata by de Alfaro and Henzinger [18] except that each state is required to be input-enabled. Like in our setting, system models in [62] can be specified operationally (in our case with open nets, in their case by means of I/O automata), or in a purely declarative manner by means of traces. They consider with quiescent traces a kind of stop traces and their refinement involves trace containment. The precongruence defined in [61] is based on traces but in a synchronous setting. In contrast, our precongruences are variants of the should testing preorder [217] in an asynchronous setting. Moreover, our notion of divergence (i.e., “infinite internal chatter”) is different from the one in [62]: If two open systems indefinitely interact with each other, then they are responsive and, hence, we do not treat such a trace as problematic, but Chen et al. [61] do. The reason is that, intuitively, we assume a stronger notion of fairness.

Common for the synchronous setting of Dill [81], de Alfaro and Henzinger [18], and Chen et al. [61] is that they all have to apply some kind of output pruning (whereas we have not): In the composition of two open systems, if a sequence of output transitions leads to an error state, these transitions and the states involved have to be removed. We avoid pruning by introducing the notion of b -uncoverable traces in Chap. 5.

WORK OF STAHL ET AL. Stahl et al. [226, 11] consider a conformance relation—called *accordance*—which preserves deadlock freedom on service automata (see Sect. 2.7). The notion of *accordance* has been first introduced in [10]. However, the decision procedure for *accordance* in [10] was limited to acyclic finite state services. *Accordance* for deadlock freedom is strictly weaker than our conformance relation for responsiveness, because responsiveness implies deadlock freedom (see Sect. 3.3 for a detailed comparison). Based on the *accordance* relation, Lohmann et al. [152] introduce a single service that encodes a set of services. This motivates the notion of a maximal b -partner in Sect. 5.3. With Thm. 129, we showed how the notion of a maximal b -partner can be used to decide b -conformance. The notion of a maximal partner is related to the notion of a canonical dual from [59]. Castagna et al. [59] propose a trivial construction method (based on mirror-

ing the message channels) for their restricted setting that does not apply to our setting.

WORK OF LOHMANN ET AL. Lohmann and Wolf [156] present a decision procedure for the responsiveness in [258], but on an automaton model and for a less general variant of responsiveness (see Sect. 3.4 for more detailed comparison to our variants of responsiveness). More generally, we deal with open nets that are responsive in some open net compositions but not in others. Responsiveness in [258, 156] is mainly motivated by algorithmic considerations for deciding the respective conformance preorder, but without characterizing the latter semantically or studying compositionality. Although we are more general, our decision procedure for b -conformance in Chap. 5 has the same worst case complexity as the decision procedure for responsiveness in [156]: Lohmann and Wolf [156] propose to compute the operating guideline (i.e., a finite representation of all b -partners) of both $Impl$ and $Spec$. Then, they check for a weak simulation relation of $OG(Impl)$ by $OG(Spec)$ that relates certain state labels of these OG's (i.e., bits representing sets of states in [156]). In contrast, we propose to compute $CSD_b(Impl)$ and $CSD_b(Spec)$ and check for a bisimulation between them that respects the state annotations. Computing $CSD_b(N)$ is at most as expensive as computing $OG(N)$ for any open net N .

WORK OF MOOIJ ET AL. Mooij et al. [180, 179] construct a finite maximal b -partner—called maximal controller—for the accordance relation of Stahl et al. [226, 11] (i.e., for deadlock freedom). In essence, their construction algorithm “unfolds” the operating guideline of Lohmann et al. [153] into a single service automaton, which results in an exponential blowup of the size of the maximal b -partner compared to the size of the operating guideline. Parnjai [205, 203] lifts this construction algorithm to an accordance relation based on a notion of responsiveness that is weaker than ours. In contrast, our construction of a maximal b -partner max_b from the LTS CSD_b in Sect. 5.3 at most doubles the size of max_b compared to the size of CSD_b . As the size of CSD_b is comparable to the size of an operating guideline (see the previous paragraph), our maximal b -partner is in general drastically smaller than the maximal controllers in [180, 179, 205, 203]. van Hee et al. [114] show how to compute maximal controllers for weak termination [181, 162, 44], but only for a subclass of open nets and without providing an implementation.

7.3.3 Work based on Petri nets

WORK OF VOGLER Vogler [246] presents a few tens of equivalences to support the modular construction of Petri nets. The setting of his work is more general than ours, as he studies asynchronously communicating (i.e., by fusing places) and synchronously communicating (i.e., by fusing transitions) Petri nets. As a difference, in the setting of Vogler [246], the interface is not separated into input and output places and interface places may be unbounded (like in the *stopdead*-semantics in Chap. 4). For open nets with an empty set of final markings, our definitions of responsiveness and conformance yields an equivalence, which is similar to P -deadlock equivalence in [246]. Vogler presents the notion of IR-equivalence for open nets that coincides with should (or fair) testing (called PF++-equivalence in [246]) and, thus, is essentially compositional conformance by Sect. 7.2. However, IR-

equivalence is, as the subcontract preorder of Laneve and Padovani [141], an asymmetric notion, whereas our notion of conformance is symmetric.

Our decidability result for compositional b -conformance builds upon the decidability of should testing by Rensink and Vogler [217]. We showed how to decide \mathcal{F}_{fin}^+ -refinement for two finite LTSs, generalizing the construction of Rensink and Vogler [217, Theorem 61] for deciding \mathcal{F}^+ -refinement. In the second step, we reduced $\mathcal{F}_{b,fin}^+$ -refinement under a precondition (i.e., $bound_b$ -inclusion) to \mathcal{F}_{fin}^+ -refinement for two finite LTSs. That way, we can conclude decidability of compositional b -conformance as it coincides with $\mathcal{F}_{b,fin}^+$ -refinement.

WORK OF VAN DER AALST AND BASTEN Van der Aalst and Basten [30, 7] introduce the notion of projection inheritance for a subclass of Petri nets—that is, workflow nets (WFNs) [1]. WFNs are subject to several syntactic restrictions and therefore less general than open nets. The notion of projection inheritance is based on branching bisimulation and relates two WFNs if they can be substituted. As a difference, the projection inheritance approach assumes a synchronous communication model (i.e., by fusing transition) and considers soundness as a correctness criterion. Soundness implies weak termination and, thus, our notions of conformance are strictly coarser than projection inheritance.

WORK OF MARTENS Martens [163, 164] presents a refinement notion for workflow modules—that is, for a Petri net formalism similar to WFNs. As WFNs, workflow modules are less general than open nets. To decide refinement of acyclic workflow modules, Martens introduces a data structure called communication graph and a simulation relation on these graphs. In essence, a communication graph represents the communication behavior of an open system and can be compared with a reduced version of our LTS CSD_b without any state labels. Due to the limitations of workflow modules and the loss of information in communication graphs compared to our LTS CSD_b , his simulation relation on communication graphs is only sufficient for the accordance notion of Stahl et al. [226, 11].

WORK OF BONCHI ET AL. Bonchi et al. [40] model the behavior of services using Consume-Produce-Read Nets; another variant of Petri nets. For their model, they present saturated bisimulation as a refinement relation. However, saturated bisimulation is too restrictive to allow reordering of sending messages and is, therefore, not suitable for a refinement relation based on asynchronous communication.

WORK OF STAHL AND VOGLER Our trace-based semantics—the *stopdead*-semantics of an open net—is an adaption of a trace-based semantics with the same name in [227, 228]. A stop except for inputs we introduced in Sect. 4.1 is, in essence, a silent marking [187] defined on the composition of two open nets instead on the environment of one open net. Compared to the work of Stahl and Vogler [228] for characterizing conformance with respect to deadlock freedom, finer trace sets are required to characterize the preorders based on responsiveness. While traces are adequate for the precongruence dealing with deadlock freedom [228], they do not suffice to characterize the coarsest precongruence for responsiveness, and we had to use some kind of failures instead. Standard failure semantics was introduced by Brookes et al. [50]. By characterizing compositional conformance and compositional

b -conformance, we use an extension of Vogler’s F^+ -semantics [246] (which Voorhoeve and Mauw [250] later introduced as impossible futures). The corresponding precongruence—that is, compositional conformance—is in essence the should testing preorder [196, 48, 217].

The construction of the LTS BSD_b in Sect. 5.2 is based on a construction with the same name in [228]. In contrast to Stahl and Vogler [228], our BSD_b represents five languages that partition the language Σ^* and are not included in each other. This is because the empty state Q_\emptyset also arises in the construction of BSD_b in [228], but is not distinguished (in terms of the state labels) from nonempty states.

7.3.4 Work related to the undecidability results

Our undecidability result in Sect. 4.3 is an extract of [193], where we showed undecidability for the unbounded preorders and precongruences with respect to deadlock freedom, responsiveness, and weak termination. The results in [193] suggest that the subcontract preorder of Bravetti and Zavattaro [46] is undecidable, although we have no formal proof for this conjecture.

Our proofs in Sect. 4.3 work by reduction from the halting problem of 2-counter machines using a variation of the “Jančar-Patterns” [126]. Counter machines and their halting problem were introduced by Minsky [178]. The halting problem for counter machines can be used very naturally to show the undecidability of other problems related to Petri nets, such as bisimilarity and language inclusion [126, 92].

The controllability problem for responsiveness [258, 156]—that is, the question whether a given open net has at least one partner—is decidable: There always exists a trivial partner with a loop in which messages are sent without waiting for an answer. As the corresponding preorder—that is, the conformance relation—is undecidable, we conclude that conformance is a more difficult problem than controllability.

Part III

THE LOG-MODEL SCENARIO

This chapter is based on results published in [190].

As explained in Chap. 1, we consider two scenarios in this thesis: the model-model scenario and the log-model scenario. In the model-model scenario, we assume that both the specification and the implementation of an open system are given as formal models. In the log-model scenario however, we assume the specification of an open system to be given as a formal model, but no formal model of the implementation is available. This is often a more realistic and practically relevant assumption than assuming the availability of a formal model of the implementation as we do in the model-model scenario. *In practice, often no formal model of the implementation is available* because the implementation is too complex to be formally modeled or people lack the skills to clearly state the precise behavior. Even if there exists a formal model of the implemented system, it can differ significantly from the actual implementation: the formal model may have been implemented incorrectly, or the implementation may have been changed over time. As the implementation itself is unknown (i.e., a black box), we cannot translate the implementation into a formal model. Instead, we are given some kind of observed behavior of the implementation, to which we refer as *event log*. In practice, an event log may be extracted from databases, message logs, or audit trails [8]. Figure 91 illustrates our assumptions for the log-model scenario.

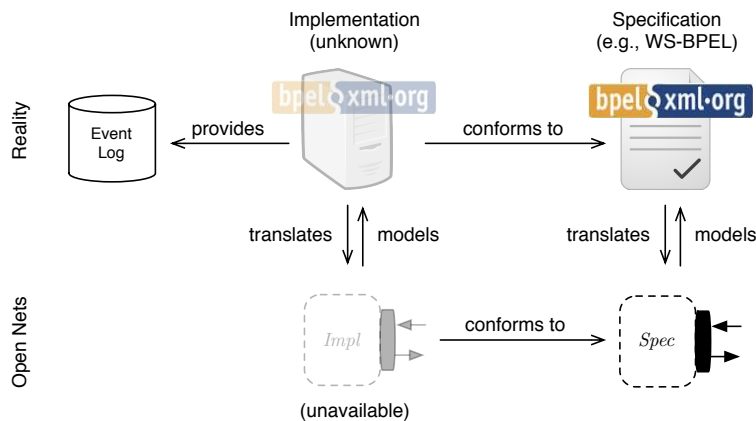


Figure 91: The log-model scenario

In Part III, we investigate how to use the event log to check conformance of the unknown implementation to the given specification. An event log is inherently *incomplete*: It is unlikely that every behavior of the unknown implementation was observed and recorded in the event log; it is even impossible, if the unknown implementation contains at least one loop that allows for an infinite set of traces. Therefore, an event log captures, in general, only parts of the behavior of an implementation. This incompleteness hinders the application of traditional verification techniques like conformance checking in the log-model scenario. Still, *testing for conformance* may be applicable,

which we investigate in this chapter. Another idea is to support the design of responsive open systems in the log-model scenario by *discovering a formal model* of the unknown implementation based on the given event log. We investigate this idea in Chap. 9. In the final chapter of Part III, Chap. 10, we summarize our results and review related work.

In the remainder of this chapter, we present a testing approach for conformance. Testing for conformance means that if there is some erroneous behavior captured by the given event log, we can conclude that the unknown implementation does not conform to the specification. However, if there is no erroneous behavior captured by the event log, we cannot make any statement whether the implementation conforms to the specification. Our notion of testing solely relies on the given specification and the observed behavior recorded by the event log; we have no control over the test case (i.e., the open system that communicates with the unknown implementation and from whose communication the provided event log originates). Our notion of testing is also called *monitoring* [219] or *passive testing* [239]. Passive testing is opposed to active testing, where a tester has active control over the test environment and especially a set of predefined tests that are executed [239, 49]. For our testing approach, we consider only the b -conformance relation because b -conformance is, in contrast to the conformance relation in Chap. 4, decidable.

Figure 92 illustrates our testing approach in the log-model scenario. A formal model of the implementation is unknown, but the implementation provides an event log instead. We assume that we are given a formalization Log of the provided event log. Again we focus on control flow, so Log is a multiset of traces that abstracts for example from captured resource or timing information. For our testing approach, we present a necessary condition for deciding b -conformance: We analyze whether there exists a b -conforming implementation which can “replay” Log —that is, whether there exists an implementation which may produce the behavior seen in Log without any mismatch. Thereby, we use the finite characterization of all b -conforming open nets that we developed in the model-model scenario. If there does not exist a b -conforming implementation that can replay Log , then the implementation, which provided the given event log, is certainly not b -conforming to the specification.

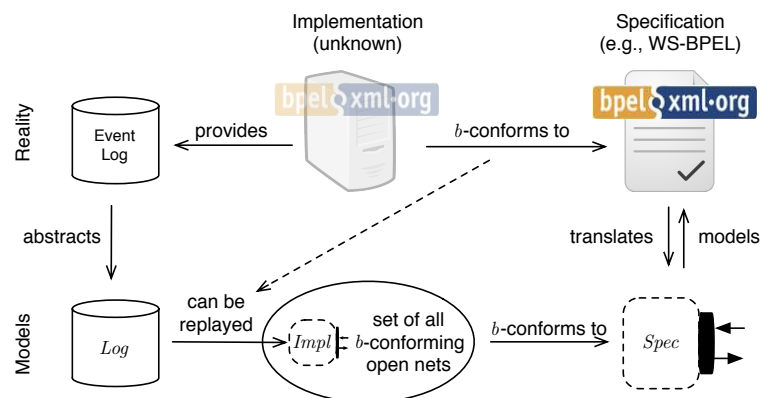


Figure 92: An illustration of conformance testing in the log-model scenario. A solid arc illustrates the relation described by the corresponding arc label. The dashed arc illustrates logical implication.

We formalize a given event log and show how to replay that formalization on an open net in Sect. 8.1. In Sect. 8.2, we develop an algorithm to test for b -conformance, and in Sect. 8.3, we implement that algorithm using the decision algorithm for b -conformance from Chap. 5. We evaluate our testing approach using industrial-sized specifications and event logs in Sect. 8.4 and finish this chapter with a conclusion in Sect. 8.5.

8.1 FORMALIZING OBSERVED BEHAVIOR

In this section, we start by formalizing the notion of an event log in Sect. 8.1.1. In Sect. 8.1.2, we show how an event log can be replayed on a labeled net, and in Sect. 8.1.3 we extend the replay approach to open nets. Figure 93 illustrates the focus of this section.

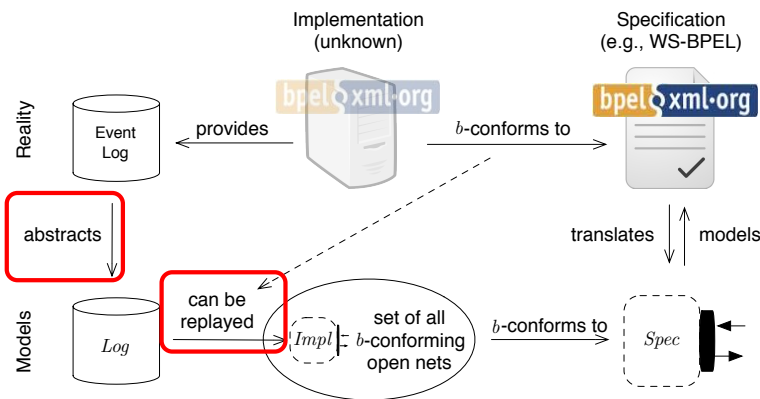


Figure 93: The focus of Sect. 8.1

8.1.1 Events, event traces, and event logs

In this section, we formalize observed behavior in terms of *event traces*. An event trace describes an observed communication sequence between two open systems in a particular case in terms of a sequence of *events* (i.e., sent and received messages). We describe an event as a label (i.e., a symbol) and abstract from extra information, such as the message content and resource or timing information of the message.

Technically, an event in an event trace and an action of a labeled Petri net coincide: Both are ordinary labels. However, we distinguish them linguistically because of their different origins. An event derives from an event trace and represents something that was observed—that is, the sending or receiving of a message—whereas an action derives from a labeled Petri net and describes the possibility to send or receive a message. In other words, an event is an observed action. As the internal, invisible action τ represents an action that cannot be observed in a labeled Petri net, τ is not an event.

Convention 8 For the remainder of this thesis, we assume a set \mathcal{E} of *events* that may have been captured in an event log. We assume $\tau \notin \mathcal{E}$. \diamond

In general, it is always possible to observe behavior that was already observed before. Therefore, we formalize an event log as a multiset of event traces instead of a set of event traces.

Definition 163 [event trace and event log]

An event trace $w \in \mathcal{E}^*$ is a sequence of events, and an event log $Log \in Bags(\mathcal{E}^*)$ is a multiset of event traces.

Example 164 As a running example for this chapter, consider the event log $DLog$ in Tab. 9. The event log $DLog$ contains information on 210 traces. There are three types of traces: qd , qqd , and sfd . Formally, $DLog$ is the multiset $[qd, \dots, qd, qqd, \dots, qqd, sfd, \dots, sfd]$ that contains the event trace qd 100 times, the event trace qqd 100 times, and the event trace sfd 10 times. \diamond

cardinality	event trace
100	qd
100	qqd
10	sfd

Table 9: The event log $DLog$ 8.1.2 *Replaying an event log on a labeled net*

We want to compare a (discovered) model (i.e., a labeled net N) with a given event log (i.e., a multiset Log of event traces). To this end, we use a replay technique [15, 5] from the area of process mining [2]. The idea is to relate each event trace $w \in Log$ to a run v of N that can be executed from N 's initial marking. We refer to this relation as an *alignment*, which, in turn, consists of a sequence of *moves*. There are three types of moves: *log moves*, *model moves*, and *synchronous moves*. Log moves and model moves formalize “mismatches” between w and v by relating a unique symbol \gg (“no move”) to an event of w or a transition of v , respectively. A synchronous move formalizes a “match” by relating an event of w to a transition of v .

Definition 165 [move]

Let Log be an event log, and let N be a labeled net. A *move* is a pair $(x, y) \in ((\mathcal{E} \uplus \{\gg\}) \times (T \uplus \{\gg\})) \setminus \{(\gg, \gg)\}$. A move (x, y) is

- a *log move* if $x \neq \gg$ and $y = \gg$;
- a *model move* if $x = \gg$ and $y \neq \gg$, and a *silent (model) move* if, additionally, $l(y) = \tau$;
- a *synchronous move* if $x \neq \gg$ and $y \neq \gg$.

Definition 166 [alignment]

Let Log be an event log, and let N be a labeled net. An *alignment* of an event trace $w \in Log$ to N is a sequence $\gamma = (x_1, y_1) \dots (x_k, y_k)$ of moves such that

1. Restricting γ 's first component to \mathcal{E} yields the event trace w , i.e., $(x_1 \dots x_k)|_{\mathcal{E}} = w$;
2. Restricting γ 's second component to T yields a run v of N from N 's initial marking m_N , i.e., $(y_1 \dots y_k)|_T = v$ such that $m_N \xrightarrow{v}$ in N ;

3. Events and transition labels coincide for all synchronous moves, i.e., for all $1 \leq i \leq k$, if $x_i \neq \gg$ and $y_i \neq \gg$, then $l(y_i) = x_i$.

We denote by $trace(\gamma) \in \Sigma^*$ the trace of N that is obtained from the run v , i.e., $trace(\gamma)$ is obtained from v by replacing each transition with its label and removing all τ labels.

Graphically, we denote an alignment γ of an event trace $w = w_1 \dots w_n$ to an open net N with transitions t_1, \dots, t_m and labeling function l as follows:

$$\gamma = \begin{array}{c|c|c|c|c} w_1 & w_2 & \gg & \dots & w_n \\ \hline l(t_1) & \gg & l(t_2) & \dots & l(t_m) \\ \hline t_1 & & t_2 & \dots & t_m \end{array}$$

The top row of γ corresponds to the event trace $w = w_1 \dots w_n$ in Log and the bottom two rows correspond to the labeled net N . There are two bottom rows because multiple transitions of N may have the same label; the upper bottom row consists of transition labels, and the lower bottom row consists of transitions. For a log move, the symbol \gg (“no move”) appears in the upper bottom row and, as no transition has been fired, the lower bottom row is empty. For a model move, the symbol \gg (“no move”) appears in the top row.

Example 167 Consider again the database D from Sect. 3.2. For convenience, we depict D and its environment $env(D)$ again in Fig. 94. For the trace sfd in $DLog$ from Tab. 9 and the labeled net $env(D)$, we have $sf \in L(env(D))$ but $sfd \notin L(env(D))$; that is, sfd deviates from sf by adding an additional event d . Thus, an alignment of sfd to $env(D)$ is $\gamma_1 = (s, s)(\gg, shutdown)(\gg, forward)(f, f)(d, \gg)$ or graphically

$$\gamma_1 = \begin{array}{c|c|c|c|c} s & \gg & \gg & f & d \\ \hline s & \tau & \tau & f & \gg \\ \hline s & shutdown & forward & f & \end{array}$$

The move (d, \gg) is a log move, because $env(D)$ cannot fire transition d without a token on the place d^0 . The move $(\gg, shutdown)$ is a model move. In addition, $(\gg, shutdown)$ is a silent move because the transition $shutdown$ is labeled with τ . All in all, the alignment γ_1 consists of two synchronous moves, two silent moves, and one log move.

In general, there exist many alignments of an event trace to a labeled net. For example, the alignment γ_1 is not the only alignment of sfd to $env(D)$. Another alignment of sfd to $env(D)$ is $\gamma_2 = (s, s)(\gg, shutdown)(\gg, forward)(f, \gg)(\gg, f)(d, \gg)$ or graphically

$$\gamma_2 = \begin{array}{c|c|c|c|c|c} s & \gg & \gg & f & \gg & d \\ \hline s & \tau & \tau & \gg & f & \gg \\ \hline s & shutdown & forward & & f & \end{array}$$

The alignment γ_2 contains a log move (f, \gg) and a model move (\gg, f) instead of a synchronous move (f, f) as in the alignment γ_1 . All in all, γ_2 consists of one synchronous move, two silent moves, two log moves, and one model move. \diamond

Intuitively, synchronous and silent moves in an alignment model that the event trace “fits” to the labeled net, whereas log and model moves in an

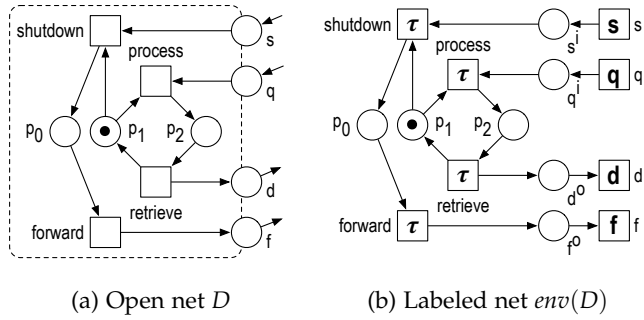


Figure 94: The open net D and the labeled net $env(D)$ from Sect. 3.2. In addition to the figures, we have $\Omega_D = \{[p_0]\}$ and $\Omega_{env(D)} = \{[p_0]\}$.

alignment model that the event trace deviates from (a trace of) the labeled net. The idea of the replay technique [15, 5] is to align every event trace w of a given event log to the labeled net N such that w fits “best” to N ; that is, an alignment of w to N has as many synchronous and silent moves as possible. To choose such an alignment, we use a cost function on moves to find an alignment with the least costs—that is, a *cost-minimal* alignment.

Definition 168 [cost function and cost-minimal alignment]
 Let $\gamma = (x_1, y_1) \dots (x_k, y_k)$ be an alignment of an event trace $w \in Log$ to a labeled net N . A *cost function* κ assigns to each move (x_i, y_i) (for $1 \leq i \leq k$) a cost $\kappa((x_i, y_i))$ such that every synchronous and silent move has cost 0, and all other types of moves have cost greater than 0. The *cost* of γ is $\kappa(\gamma) = \sum_{i=1}^k \kappa((x_i, y_i))$; γ is *cost-minimal* if, for all alignments γ' of w to N , $\kappa(\gamma) \leq \kappa(\gamma')$.

Convention 9 To simplify future notations that are based on the cost of an alignment, we assume in the remainder of this thesis that a cost function κ is given every time an event log is given. For the examples in the remainder of this thesis, we assume that κ assigns cost of 1 to each log move and to each non-silent model move, and cost of 0 to each synchronous move and to each silent model move. Van der Aalst et al. [5] also refer to this cost function as “standard distance function”. \diamond

Example 169 Consider again the alignments γ_1 and γ_2 from Ex. 167. Both alignments align the event trace sfd of $DLog$ in Tab. 9 to the labeled net $env(D)$ in Fig. 94b. We can distinguish γ_1 and γ_2 by their costs: γ_1 consists of two synchronous moves, two silent moves, and one log move. Thus, we have $\kappa(\gamma_1) = 2 \cdot 0 + 2 \cdot 0 + 1 \cdot 1 = 1$. In contrast, γ_2 consists of one synchronous move, two silent moves, two log moves, and one model move, yielding $\kappa(\gamma_2) = 1 \cdot 0 + 2 \cdot 0 + 2 \cdot 1 + 1 \cdot 1 = 3$. Thus, we prefer γ_1 over γ_2 . \diamond

For every event trace w of a given event log Log and every labeled net N , there exists at least one cost-minimal alignment γ of w to N . However, γ is in general not unique; that is, there may exist multiple cost-minimal alignments of w to N . We show this with the following example.

Example 170 Consider again the event log $DLog$ in Tab. 9 and the labeled net $env(D)$ in Fig. 94b. For the trace $qqd \in DLog$, an alignment of qqd to $env(D)$ is $\gamma_3 = (q, q)(q, q)(\gg, process)(\gg, retrieve)(d, d)$ or graphically

$$\gamma_3 = \begin{array}{c|c|c|c|c} q & q & \gg & \gg & d \\ q & q & \tau & \tau & d \\ q & q & process & retrieve & d \end{array}$$

The alignment γ_3 consists of three synchronous moves and two silent moves. Thus, we have $\kappa(\gamma_3) = 3 \cdot 0 + 2 \cdot 0 = 0$, and, therefore, γ_3 is clearly a cost-minimal alignment of qqd to $env(D)$. However, there exist other cost-minimal alignments of qqd to $env(D)$. As an example, consider the following alignment γ_4 of qqd and $env(D)$:

$$\gamma_4 = \begin{array}{c|c|c|c|c} q & \gg & \gg & q & d \\ q & \tau & \tau & q & d \\ q & process & retrieve & q & d \end{array}$$

The alignment γ_4 consists, as γ_3 , of three synchronous moves and two silent moves. Thus, $\kappa(\gamma_4) = \kappa(\gamma_3) = 0$ and γ_4 is also a cost-minimal alignment of qqd to $env(D)$. \diamond

Example 170 shows that there may exist more than one cost-minimal alignment of an event trace w to a labeled net N . As our goal is to align event traces in the event log to traces of the model such that they “fit best”, we select an arbitrary cost-minimal alignment. To this end, we use a deterministic “oracle” function which gives, for each event trace w of an event log Log and a labeled net N , a cost-minimal alignment of w to N . It is always possible to create such a function based on some predefined precedences—for example, by establishing a partial order over the moves of alignments.

Definition 171 [oracle function for cost-minimal alignments]

Let Log be an event log and let N be a labeled net. Then \mathcal{O}_N is an *oracle function* if for all $w \in Log$, $\mathcal{O}_N(w)$ is a cost-minimal alignment of w to N .

The alignments produced by the oracle function \mathcal{O}_N can be used to replay an event log on the labeled net N and to quantify (in terms of costs) the mismatch between them. By Conv. 9, we can also derive the following corollary.

Corollary 172 [language vs. cost-minimal alignments]

For any labeled net N and $w \in \mathcal{E}^*$, $w \in L(N)$ if and only if $\kappa(\mathcal{O}_N(w)) = 0$.

In the next section, we lift the replay approach to event logs and open nets.

8.1.3 Replaying an event log on an open net

In this section, we describe two viewpoints of an event log and how we can replay an event log on an open net depending on the viewpoint taken.

Assume an open net N that communicates with its environment—that is, other open nets—and an event log Log that captures the communication between N and its environment. The event log Log may take one out of two *viewpoints* with respect to N depending on what or when events are

recorded in Log [190]: If events are recorded when N consumes (produces) a message from (for) its environment, then Log takes the *viewpoint of N* . In contrast, if events are recorded when the environment consumes (produces) a message from (for) N , then Log takes the *viewpoint of N 's environment*. Figure 95 illustrates the two different viewpoints of Log .

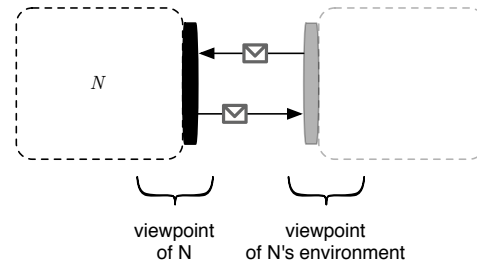


Figure 95: The viewpoint of an event log depends on what or when events are observed: An event log Log can take the viewpoint of an open net N (i.e., an event in Log represents N sending or receiving a message) or the viewpoint of N 's environment (i.e., an event in Log represents N 's environment sending or receiving a message).

For replaying an event log Log on an open net N , we have to consider Log 's viewpoint. If Log takes the viewpoint of N , we can use the inner net $inner(N)$ for replaying Log on N . This changes if Log takes the viewpoint of N 's environment. We cannot use $inner(N)$ because of the assumed asynchronous communication, and we cannot use a concrete model of N 's environment because such a model is, in general, not available. Therefore, we may be forced to use the labeled net $env(N)$ for replaying Log on N : The labeled net $env(N)$ “simulates” asynchronous communication from the viewpoint of the environment of N . Example 167 and Ex. 170 use the latter viewpoint. We formalize the usage of $inner(N)$ and $env(N)$ depending on which viewpoint Log takes by introducing the notion of a *replay environment* of Log and N .

Definition 173 [replay environment]

Let Log be an event log and let N be an open net. The *replay environment* $replay(Log, N)$ of Log and N is a labeled net defined by

$$replay(Log, N) = \begin{cases} env(N), & \text{if } Log \text{ takes the viewpoint of the} \\ & \text{environment of } N \\ inner(N), & \text{if } Log \text{ takes the viewpoint of } N. \end{cases}$$

Convention 10 To simplify the notation in Def. 173, we do not mention the event log Log as a parameter of the replay environment and write $replay(N)$ instead of $replay(Log, N)$; the concrete event log Log will be always clear from the context. \diamond

Example 174 Consider again the open net D in Fig. 94a. Figure 94b illustrates the environment $env(D)$ of D . If $DLog$ in Tab. 9 takes the viewpoint of D 's environment, we have to consider the actions of any open net that can be composed with D . Such an open net may send a message s or a message q to D at any time. Therefore, each reachable marking of $env(N)$ enables the s -labeled and the q -labeled transition. All internals of D , such

as receiving a message s (e.g., firing the transition *shutdown* at the marking $[p_1, s^i]$ yielding the marking $[p_0]$) or sending a message d (e.g., firing the transition *retrieve* at the marking $[p_2]$ yielding the marking $[p_1, d^o]$), are hidden by labeling the respective transitions with τ . If $DLog$ takes the viewpoint of D 's environment, then γ_1 from Ex. 167 is an alignment of the event trace sfd to $replay(D)$ and γ_3 from Ex. 170 is an alignment of qqd to $replay(D)$.

It may also be possible that $DLog$ takes the viewpoint of D instead of the environment of D . Then, we have $replay(D) = inner(D)$, which we depict in Fig. 96. The open net D cannot receive a message s or a message q at any time. For example, D cannot receive s after receiving q because D has to send a message d first. Therefore, after firing transition *process* in $inner(D)$, we first have to fire transition *retrieve* to enable transitions *shutdown* and *process* again. Consequently, we cannot align the event trace qqd to $replay(D)$ with a log move if $DLog$ takes the viewpoint of D , in contrast to the alignment γ_3 of qqd to $replay(D)$ if $DLog$ takes the viewpoint of D 's environment. An alignment of qqd to $replay(D)$ is, for example, the following alignment γ_5 :

$$\gamma_5 = \begin{array}{c|c|c} q & q & d \\ \hline q & \gg & d \\ process & & retrieve \end{array}$$

The alignment γ_5 of qqd to $replay(D)$ has costs of $\kappa(\gamma_5) = 1$ and is cost-minimal. In contrast, if $DLog$ takes the viewpoint of D 's environment, then γ_3 is a cost-minimal alignment of qqd to $replay(D)$ with costs of $\kappa(\gamma_3) = 0$. \diamond

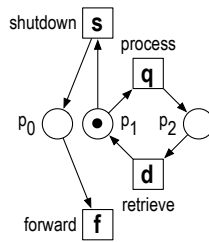


Figure 96: The labeled net $inner(D)$. We have $\Omega_{inner(D)} = \{[p_0]\}$.

8.2 THE TESTING PROCEDURE

In this section, we present a testing approach for b -conformance that is based on a simple necessary condition: If an event log Log contains observed behavior of the unknown implementation $Impl$, and $Impl$ b -conforms to the known specification $Spec$, then there exists at least one open net (viz., $Impl$) that b -conforms to $Spec$ and on which Log can be replayed without any mismatch. In other words, if we can show that Log cannot be replayed without any mismatch on any open net that b -conforms to $Spec$, then $Impl$ cannot b -conform to $Spec$: Log contains observed behavior of $Impl$ and, thus, can be certainly replayed on $Impl$ without any mismatch. Figure 97 illustrates the focus of this section; note that the dashed arc in Fig. 97 illustrates logical implication.

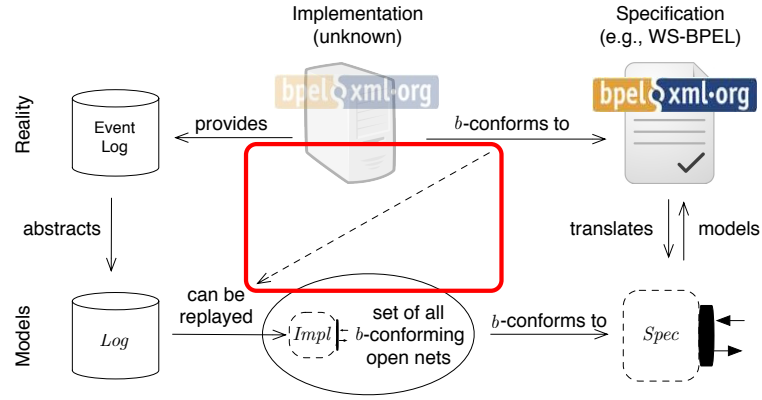


Figure 97: The focus of Sect. 8.2

For our testing approach, we have to decide whether there exists a b -conforming open net to $Spec$ which can replay Log without any mismatch. To this end, we construct the open net $mp_b(max_b(Spec))$ and show that $mp_b(max_b(Spec))$ represents the language of all open nets that b -conform to $Spec$. The existence of $mp_b(max_b(Spec))$ relies on the existence of two specific b -partners of any open net N : a maximal b -partner $max_b(N)$ from Def. 131 and an L_b -maximal b -partner $mp_b(N)$ from Def. 110. In this chapter, we refer to the open net $mp_b(N)$ as *most-permissive b -partner*. A maximal b -partner $max_b(N)$ is maximal with respect to the b -conformance relation; that is, every b -partner of $max_b(N)$ b -conforms to N by Thm. 129. A most-permissive b -partner $mp_b(N)$, in turn, is maximal with respect to its flooded language; that is, every b -partner of N has at most the traces and $bound_b$ -violators of $mp_b(N)$ by Lem. 130. For technical details of maximal and most-permissive b -partners, we refer to Sect. 5.2; here, we only employ the results of the theory presented in that section.

Because replaying an event log on an open net N only refers to the language of N , we employ a slight modification of Lem. 130: The open net $mp_b(N)$ is also maximal with respect to its language $L(mp_b(N))$. In other words, N can visit all markings in the composition with $mp_b(N)$ that can be visited in the composition with any b -partner of N .

Lemma 175 [mp_b is maximal with respect to its language]

Let N be an open net such that $MP_b(N)$ exists. Then for all b -partner C of N , $L(C) \subseteq L(mp_b(N))$.

Proof. As $MP_b(N)$ exists, the open net $mp_b(N)$ is a b -partner of N by Lem. 112. Let C be a b -partner of N . Then $L(C) \subseteq L_b(C) \subseteq co-uncov_b(N)$ by Def. 84 and Def. 93. By Cor. 116(7) and the construction of $MP_b(N)$ in Def. 104, we have $co-uncov_b(N) = L(CSD_b(N)) \setminus L_0(CSD_b(N)) = L(MP_b(N))$, thus $L(C) \subseteq L(MP_b(N))$. By the construction of $mp_b(N)$ from $MP_b(N)$ in Def. 110 and by Def. 15, we have $L(MP_b(N)) = L(inner(mp_b(N))) \subseteq L(mp_b(N))$, thus $L(C) \subseteq L(mp_b(N))$. \square

Given an open net $Spec$, the open net $max_b(Spec)$ is a b -partner of $Spec$ and the open net $mp_b(max_b(Spec))$ is a b -partner of $max_b(Spec)$. In addition, $mp_b(max_b(Spec))$ b -conforms to $Spec$. Figure 98 illustrates the relationship between the three open nets $Spec$, $max_b(Spec)$, and $mp_b(max_b(Spec))$. In the following, we show that $mp_b(max_b(Spec))$ is a canonical open net that represents the language of all open nets $Impl$ that b -conform to $Spec$. In other

words, a word w is a trace of $mp_b(max_b(Spec))$ if and only if there exists a b -conforming open net $Impl$ of $Spec$ such that w is also a trace of $Impl$.

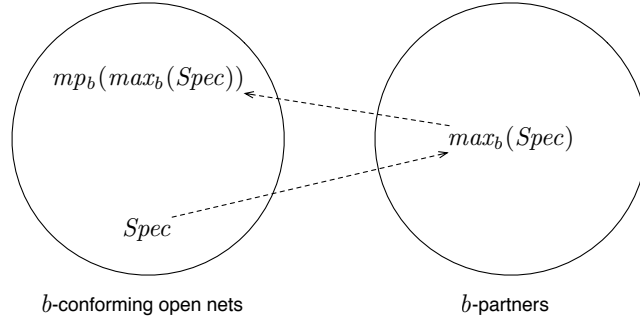


Figure 98: The relationship between the three open nets $Spec$, $max_b(Spec)$, and $mp_b(max_b(Spec))$. The two circles form an Euler diagram. The left circle illustrates the set of b -conforming open nets of $Spec$, and the right circle illustrates the set of b -partners of $Spec$. A dashed arc illustrates which open net we construct from which other open net and, in addition, the b -partner relation.

Lemma 176 [language of the open net $mp_b(max_b)$]

Let $Spec$ be an open net such that $MP_b(Spec)$ exists. For all words $w \in \mathcal{E}^*$, we have

$$w \in L(mp_b(max_b(Spec))) \quad \text{iff} \quad \text{there exists an open net } Impl \text{ such that} \\ Impl \sqsubseteq_{b, conf} Spec \text{ and } w \in L(Impl).$$

Proof. \Rightarrow : Because $MP_b(Spec)$ exists, the open net $max_b(Spec)$ exists by Def. 131. The open net $max_b(Spec)$ is a b -partner of the open net $Spec$ by Thm. 133. By Def. 44, $Spec$ is also a b -partner of $max_b(Spec)$ and, thus, the LTS $MP_b(max_b(Spec))$ exists by Def. 104 and Thm. 115. Therefore, the open net $mp_b(max_b(Spec))$ exists by Def. 110. The open net $mp_b(max_b(Spec))$ is a b -partner of $max_b(Spec)$ by Lem. 112. By Thm. 129, $mp_b(max_b(Spec))$ b -conforms to $Spec$ and, by assumption, we have $w \in L(mp_b(max_b(Spec)))$.

\Leftarrow : Assume there exists an open net $Impl$ such that $Impl \sqsubseteq_{b, conf} Spec$ and $w \in L(Impl)$. Because $MP_b(Spec)$ exists, the open net $max_b(Spec)$ exists by Def. 131. By Thm. 129 and Thm. 133, $Impl$ is a b -partner of $max_b(Spec)$. Then, the LTS $MP_b(max_b(Spec))$ exists by Def. 104 and Thm. 115, and, thus, the open net $mp_b(max_b(Spec))$ exists by Def. 110. By Lem. 175, we have $w \in L(Impl) \subseteq L(mp_b(max_b(Spec)))$. \square

Technically, we can show Lem. 176 also with $max_b(max_b(N))$ instead of $mp_b(max_b(N))$ for an open net N , because Lem. 175 holds for $max_b(N)$, too (see also Thm. 133 and Def. 128 for relating $mp_b(N)$ and $max_b(N)$). However, we use $mp_b(max_b(N))$ because $mp_b(N)$ has at most as many places and transitions as $max_b(N)$ by construction (see Def. 110 and Def. 131). Although the construction of $mp_b(N)$ and $max_b(N)$ is equally complex, the smaller size of $mp_b(N)$ becomes handy for implementing our testing approach.

Lemma 176 gives a necessary condition for the question whether the unknown implementation $Impl$ b -conforms to the given specification $Spec$: If an event log Log derives from $Impl$, then we must be able to replay Log on the labeled net $env(mp_b(max_b(Spec)))$ without any mismatch, because we have $L(mp_b(max_b(Spec))) = L(env(mp_b(max_b(Spec))))$ by Conv. 6.

Based on this necessary condition for b -conformance, we develop a testing approach as illustrated in Fig. 99 (ignoring the three groups at the moment): We compute the open net $mp_b(max_b(Spec))$ and replay the given event log Log on the labeled net $env(mp_b(max_b(Spec)))$. If Log cannot be replayed on the labeled net $env(mp_b(max_b(Spec)))$ without any mismatch, then Log cannot be replayed without any mismatch on any open net that b -conforms to $Spec$ by Lem. 176. Because Log contains observed behavior of $Impl$ and, thus, can be certainly replayed on $Impl$ without any mismatch, the unknown implementation $Impl$ cannot b -conform to $Spec$. However, if Log can be replayed on the labeled net $env(mp_b(max_b(Spec)))$ without any mismatch, we cannot make any statement whether the unknown implementation $Impl$ conforms to $Spec$: There exists an open net that b -conforms to $Spec$ by Lem. 176, but we do not know whether this open net coincides with $Impl$.

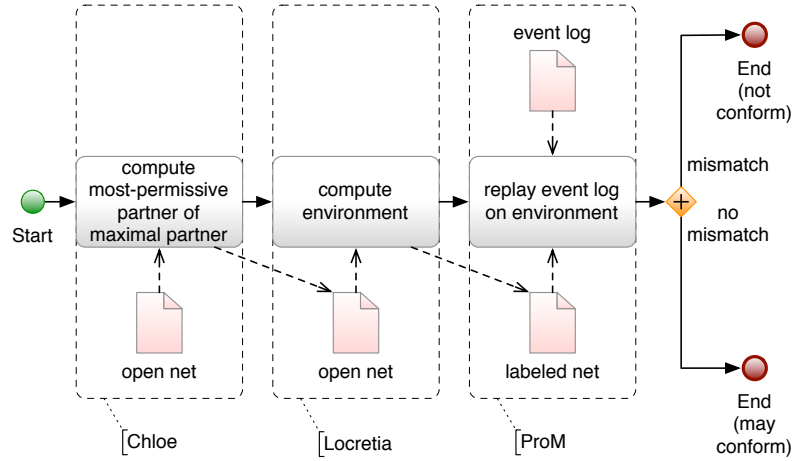


Figure 99: A BPMN diagram that illustrates the testing approach. The three groups indicate which tool implements which activity.

Theorem 177 [testing for b -conformance with $mp_b(max_b)$]

Let $Impl$ and $Spec$ be two interface-equivalent open nets such that $MP_b(Spec)$ exists. Let Log be an event log such that $Log \subseteq L(replay(Impl))$.

If there exists an event trace $w \in Log$ such that $\kappa(\mathbb{O}_{env(mp_b(max_b(Spec)))}(w)) > 0$, then $Impl$ does not b -conform to $Spec$.

Proof. Assume there exists $w \in Log$ such that $\kappa(\mathbb{O}_{env(mp_b(max_b(Spec)))}(w)) > 0$. If Log takes the viewpoint of the environment of $Impl$, we have $replay(Impl) = env(Impl)$ by Def. 173 and, therefore, $w \in L(Impl)$. If Log takes the viewpoint of $Impl$, we have $replay(Impl) = inner(Impl)$ by Def. 173 and, therefore, $w \in L(inner(Impl)) \subseteq L(Impl)$. In both cases, we have $w \in L(Impl)$.

By Cor. 172, the assumption $\kappa(\mathbb{O}_{env(mp_b(max_b(Spec)))}(w)) > 0$ implies $w \notin L(env(mp_b(max_b(Spec)))) = L(mp_b(max_b(Spec)))$. Therefore, $Impl$ does not b -conform to $Spec$ by Lem. 176. \square

The converse of Thm. 177 does not hold: If $Impl$ does not b -conform to $Spec$, then there may exist a trace $w \in L(replay(Impl))$ that cannot be replayed on $env(mp_b(max_b(Spec)))$ without any mismatch; still, that does not imply $w \in Log$. In other words, if there is no erroneous behavior captured by Log (e.g., the trace w), then we cannot make any statement whether the implementation $Impl$ b -conforms to $Spec$.

Note that neither the construction of the open net $mp_b(max_b(Spec))$ nor the replay of the given event log Log on $env(mp_b(max_b(Spec)))$ according to Thm. 177 depends on the viewpoint of Log : Our formalization of the language of an open net via its environment in Conv. 6 implicitly covers both viewpoints of Log by Def. 173.

Also note that Thm. 177 *does not make any assumption* about the relation between the given event log Log and the labeled net $env(mp_b(max_b(Spec)))$ on which we replay Log . In contrast, replay techniques from the area of process mining implicitly assume that Log only contains traces w of a labeled net N that lead to a final marking of N (cf. [219, 15, 5]). Replaying w on N without any mismatch but without reaching a final marking of N is considered erroneous in [219, 15, 5]. As we do not make this assumption, our approach even works with event logs of low quality [6], e.g., event logs with incomplete event traces.

In the next section, we describe the implementation of our testing approach.

8.3 IMPLEMENTATION

In the log-model scenario, we cannot check whether an implementation $Impl$ b -conforms to a specification $Spec$ as in Chap. 5, because the open net $Impl$ is simply not available. Instead, we use a testing approach based on observed behavior Log of $Impl$: We compute the open net $mp_b(max_b(Spec))$ from $Spec$, and then test whether we can replay Log on the environment of $mp_b(max_b(Spec))$ without any mismatch, as illustrated in Fig. 99.

For computing the open net $mp_b(max_b(Spec))$ (the first activity in Fig. 99), we rely on the theory about most-permissive and maximal b -partners in Chap. 5: In Sect. 5.4, we presented the tool Chloe [115], which can compute the LTS $CSD_b(N)$ for any open net N . Thereby, $CSD_b(N)$ represents the b -coverable *stopdead*-semantics of N . Both open nets $mp_b(N)$ and $max_b(N)$ can be constructed from $CSD_b(N)$ according to Def. 110 and Def. 131. Consequently, we reuse the existing implementation and extend Chloe to also construct $mp_b(N)$ and $max_b(N)$ for any open net N . In addition, we can compile an open net N into the labeled net $env(N)$ (the second activity in Fig. 99) using the tool Locretia [116].

For replaying an event log on a labeled net (the third activity in Fig. 99) according to Sect. 8.1.2, we use the existing package “PNetReplayer” of the tool ProM [212]. ProM is an extensible framework that supports a wide variety of process mining techniques. The package “PNetReplayer” implements the A^* -algorithm [15] and is part of the current ProM release version 6.3.

The tools Chloe and Locretia are free and open source software licensed under the GNU Affero General Public License; the tool ProM is free and open source software licensed under the GNU Public License. Therefore, we can test for b -conformance by completely relying on free and open source software.

In the next section, we show that $mp_b(max_b(N))$ can actually be computed for open nets N of industrial size. Based on that computation, we show that our testing approach is applicable for industrial-sized event logs.

8.4 EVALUATION AND EXPERIMENTAL RESULTS

Testing for b -conformance, as illustrated in Fig. 99, can be done using the three tools Chloe [115], Locretia [116], and ProM [212]. In this section, we

evaluate these tools with real-life models and artificial event logs. We describe our evaluation process and prepare the real-life models in Sect. 8.4.1. In Sect. 8.4.2, we evaluate our testing approach using artificial event logs of unknown implementations that 1-conform to their respective specification; in Sect. 8.4.3, we use artificial event logs of non 1-conforming implementations instead.

8.4.1 Preparing the evaluation process

Figure 100 illustrates our evaluation process in detail. First, we compute nine open nets as specifications from nine industrial (service) models. Then, for each computed open net N , we artificially create two event logs: (1) an event log $Succeed(N)$ of an unknown implementation that 1-conforms to N , and (2) an event log $Fail(N)$ of an unknown implementation that does not 1-conform to N . Finally, we test both unknown implementations for 1-conformance to N using our testing approach: Replaying $Succeed(N)$ on $env(mp_1(max_1(N)))$ should yield costs of 0 (i.e., there is no mismatch and the test succeeds), and replaying $Fail(N)$ on $env(mp_1(max_1(N)))$ should yield costs that are greater than 0 (i.e., there is at least one mismatch and the test fails).

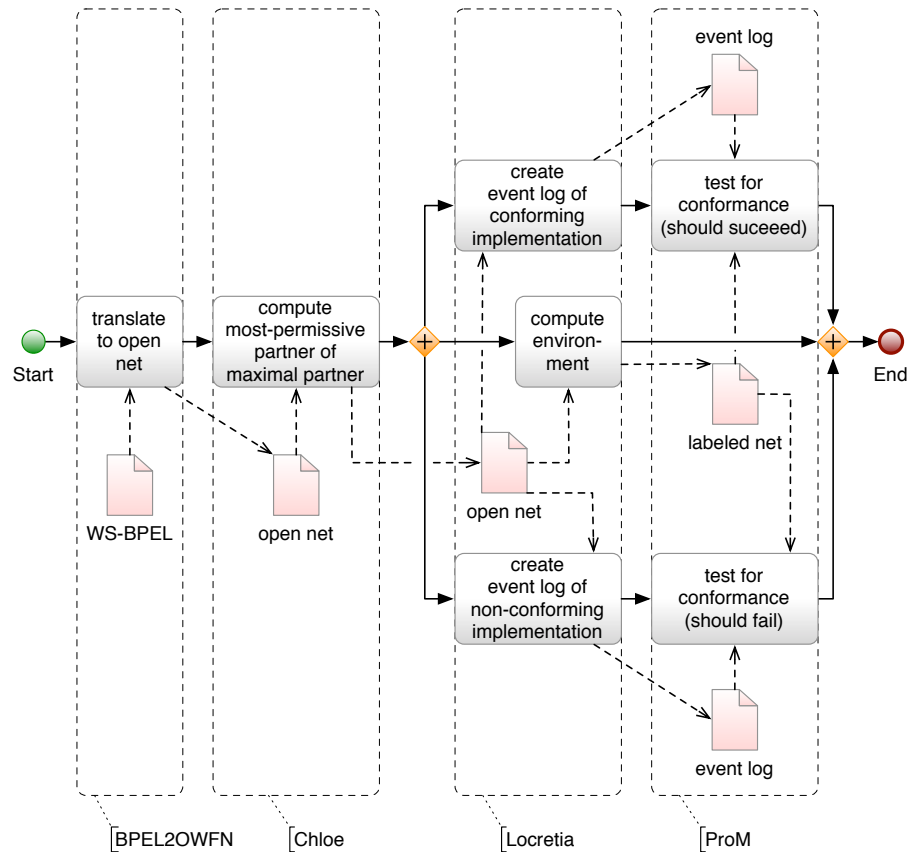


Figure 100: A BPMN diagram that illustrates the evaluation process. The four groups indicate which tool we use for which activity.

We use our running examples D , D' , U , U' , and the five industrial open systems CN , LA , PO , RS , and TR from Sect. 5.4 as specifications. Because CN , LA , PO , RS , and TR are services [201] that are specified in WS-BPEL [130],

we translate them into open nets using the compiler BPEL2OWFN [149]. Table 10 lists the characteristics of the resulting open nets from Sect. 5.4. As in Sect. 5.4, we conduct all computations in this section on a MacBook Air model A1466 [21] with one Intel Core i5 1.3 GHz CPU with 2 independent processor cores and 8 GiB of memory to demonstrate the feasibility of our implementation on today’s average (personal) computers.

open net (abbreviation)	$ P $	$ I $	$ O $	$ T $	$ F $
Database (D)	3	2	2	4	11
Patched Database (D')	2	2	2	3	8
First User (U)	2	2	2	2	6
Second User (U')	2	2	2	3	7
Contract Negotiation (CN)	76	4	7	98	294
Loan Approval (LA)	34	3	3	17	60
Purchase Order (PO)	74	4	6	96	290
Reservations (RS)	38	2	8	33	83
Ticket Reservation (TR)	90	3	6	123	363

Table 10: The size of the derived open nets.

We compute $max_1(N)$ and $mp_1(max_1(N))$ for each of the nine open nets N in Tab. 10 using the tool Chloe. Thereby, we compute $max_1(N)$ from the LTS $CSD_1(N)$ by Def. 131 and $mp_1(max_1(N))$ from the LTS $CSD_1(max_1(N))$ by Def. 110; $CSD_1(N)$ and $CSD_1(max_1(N))$ represent the 1-coverable *stopdead*-semantics of N and $max_1(N)$, respectively. Table 11 shows the size of $CSD_1(N)$ and $CSD_1(max_1(N))$. In three cases (namely CN , LA , and PO), $CSD_1(N)$ and $CSD_1(max_1(N))$ are of equal size. In the remaining six cases, $CSD_1(max_1(N))$ is up to three times larger than $CSD_1(N)$ (1,171 states versus 371 states). However, computing $CSD_1(max_1(N))$ is feasible for all examples on an average computer with a runtime up to 747 seconds (approx. 12.5 minutes) and 616,776 KiB of used memory (which is only approx. $\frac{1}{16}$ of the available memory).

Table 12 shows the size of the open nets $max_1(N)$ and $mp_1(max_1(N))$ that we generate from $CSD_1(N)$ and $CSD_1(max_1(N))$, respectively. In two of nine cases, $max_1(N)$ is smaller than $mp_1(max_1(N))$, but in seven of nine cases, $max_1(N)$ is larger than $mp_1(max_1(N))$. Therefore, it is not clear how the size of $max_1(N)$ and $mp_1(max_1(N))$ compare to each other in general.

In the next section, we create an artificial event log $Succeed(N)$ of an unknown implementation that 1-conforms to N , for each open net N in Tab. 10. Based on $Succeed(N)$, we then test whether the unknown implementation 1-conforms to N using our testing approach.

8.4.2 Testing 1-conforming implementations

For each open net N in Tab. 10, we create an artificial event log $Succeed(N)$ of an unknown implementation that 1-conforms to N using the tool *Locretia* [116] and the open net $mp_1(max_1(N))$ (because $mp_1(max_1(N))$ represents all traces of a 1-conforming implementation according to Lem. 176). Each such event log $Succeed(N)$ takes the viewpoint of a 1-partner of N (i.e., N ’s environment), is free of noise, and consists of 400 event traces with about 3,239–3,724 events; see Tab. 13 for an overview over the size of the generated event logs. The size of our generated event logs coincides with the size of event logs that were successfully applied to evaluate process min-

LTS	$ Q $	$ \delta $	$ \Sigma^{in} $	$ \Sigma^{out} $	time (s)	memory (KiB)
$CSD_1(D)$	4	16	2	2	0	1,424
$CSD_1(D')$	5	20	2	2	0	1,412
$CSD_1(U)$	8	32	2	2	0	1,412
$CSD_1(U')$	6	24	2	2	0	1,420
$CSD_1(CN)$	578	6,358	4	7	13	106,784
$CSD_1(LA)$	22	132	3	3	0	2,028
$CSD_1(PO)$	170	1,700	4	6	4	33,028
$CSD_1(RS)$	371	3,710	2	8	0	4,352
$CSD_1(TR)$	112	1,008	3	6	2	24,256
$CSD_1(max_1(D))$	5	20	2	2	0	1,448
$CSD_1(max_1(D'))$	6	24	2	2	0	1,464
$CSD_1(max_1(U))$	10	40	2	2	0	1,528
$CSD_1(max_1(U'))$	8	32	2	2	0	1,468
$CSD_1(max_1(CN))$	578	6,358	7	4	747	616,776
$CSD_1(max_1(LA))$	22	132	3	3	0	2,492
$CSD_1(max_1(PO))$	170	1,700	6	4	23	103,084
$CSD_1(max_1(RS))$	1,171	11,710	8	2	138	288,388
$CSD_1(max_1(TR))$	144	1,296	6	3	5	40,672

Table 11: The size of the LTSs $CSD_1(N)$ and $CSD_1(max_1(N))$ generated with the tool Chloe, including the used memory and time. $CSD_1(N)$ and $CSD_1(max_1(N))$ represent the 1-coverable *stopdead*-semantics of N and $max_1(N)$, respectively.

open net	$ P $	$ I $	$ O $	$ T $	$ F $
$max_1(D)$	4	2	2	12	35
$max_1(D')$	5	2	2	15	44
$max_1(U)$	10	2	2	30	87
$max_1(U')$	6	2	2	16	47
$max_1(CN)$	1,011	7	4	8,331	24,559
$max_1(LA)$	27	3	3	108	318
$max_1(PO)$	259	6	4	1,812	5,346
$max_1(RS)$	489	8	2	4,154	12,343
$max_1(TR)$	150	6	3	1,004	2,973
$mp_1(max_1(D))$	4	2	2	11	33
$mp_1(max_1(D'))$	5	2	2	13	39
$mp_1(max_1(U))$	9	2	2	24	72
$mp_1(max_1(U'))$	7	2	2	20	60
$mp_1(max_1(CN))$	577	4	7	3,899	11,697
$mp_1(max_1(LA))$	21	3	3	84	252
$mp_1(max_1(PO))$	169	4	6	1,066	3,198
$mp_1(max_1(RS))$	1,170	2	8	6,092	18,276
$mp_1(max_1(TR))$	143	3	6	730	2,190

Table 12: The size of $max_1(N)$ and $mp_1(max_1(N))$ generated with the tool Chloe. We do not including the used memory and time because every open net could be generated instantly from the given LTS $CSD_1(N)$ and $CSD_1(max_1(N))$, respectively.

ing techniques: For example, Buijs et al. [53] evaluate workflow discovery

algorithms with real-life event logs that were extracted from information systems of municipalities participating in the CoSeLoG project [4]. The extracted event logs in [53] contain 100–444 event traces and 590–3,269 events.

event log	event traces	events	events per event trace
$Succeed(D)$	400	3,239	8.10
$Succeed(D')$	400	3,481	8.70
$Succeed(U)$	400	3,724	9.31
$Succeed(U')$	400	3,421	8.55
$Succeed(CN)$	400	3,414	8.54
$Succeed(LA)$	400	3,415	8.54
$Succeed(PO)$	400	3,310	8.28
$Succeed(RS)$	400	3,409	8.52
$Succeed(TR)$	400	3,418	8.55

Table 13: The size of the event logs that we generated using the tool Locretia. Each event log $Succeed(N)$ takes the viewpoint of N 's environment and contains observed behavior from an unknown implementation that 1-conforms to N .

Example 178 Consider again our running example of this chapter, the open net D from Fig. 94a. Using Locretia, we generate an event log $Succeed(D)$ from a 1-conforming implementation of D . The generated event log $Succeed(D)$ contains 400 event traces and 3,239 events by Tab. 13. Figure 101 shows a screenshot of ProM visualizing the event log $Succeed(D)$. In ProM, a “case” refers to an event trace, which results in the previously mentioned 400 event traces and 3,239 events (visualized in the top left box of Fig. 101 under “key data”). The length of the event traces of $Succeed(D)$ is equally distributed between 1 and 16. \diamond

For each of the nine open nets $mp_1(max_1(N))$ in Tab. 12, we compute the labeled net $env(mp_1(max_1(N)))$ using the tool Locretia [116]. We test for 1-conformance of an unknown implementation (i.e., an open net that may produce the event log $Succeed(N)$) to N by replaying $Succeed(N)$ on $env(mp_1(max_1(N)))$. For replaying an event log on a labeled net, we use the package “PNetReplayer” of the tool ProM. All settings of “PNetReplayer” were left to the standard settings except for the cost function: As already detailed in Conv. 9, we use a cost function that assigns cost of 1 to each log move and to each non-silent model move, and cost of 0 to all other moves.

Table 14 shows the results of these tests: In each case, the cost for replaying the event log $Succeed(N)$ on the labeled net $env(mp_1(max_1(N)))$ are 0 (i.e., the test succeeds). Therefore, by Thm. 177, we cannot make any statement whether the unknown implementation 1-conforms to the specification N , which is exactly the result that we expected. The runtime of replaying $Succeed(N)$ on $env(mp_1(max_1(N)))$ using the A^* -algorithm [15] is between 0.452 and 3,874.521 seconds (approx. 1 hour). As testing for b -conformance is not time-critical, we conclude that our testing approach is feasible on today’s average computers.

8.4.3 Testing non 1-conforming implementations

In this section, we slightly alter the procedure from Sect. 8.4.2: We create an artificial event log $Fail(N)$ of an unknown implementation that *does not* 1-



Figure 101: The “dashboard” view in ProM visualizes the event log $Succceed(D)$ of an unknown implementation that 1-conforms to D .

event log	labeled net	replay costs	time (s)
$Succceed(D)$	$env(mp_1(max_1(D)))$	0	1.260
$Succceed(D')$	$env(mp_1(max_1(D'))) $	0	0.452
$Succceed(U)$	$env(mp_1(max_1(U)))$	0	0.616
$Succceed(U')$	$env(mp_1(max_1(U'))) $	0	0.616
$Succceed(CN)$	$env(mp_1(max_1(CN)))$	0	687.388
$Succceed(LA)$	$env(mp_1(max_1(LA)))$	0	1.175
$Succceed(PO)$	$env(mp_1(max_1(PO)))$	0	65.072
$Succceed(RS)$	$env(mp_1(max_1(RS)))$	0	3,874.521
$Succceed(TR)$	$env(mp_1(max_1(TR)))$	0	53.081

Table 14: The time and cost ProM reported for replaying the event log $Succceed(N)$ on the labeled net $env(mp_1(max_1(N)))$. We used the standard settings of the package “PNetReplayer” and the cost function from Conv. 9.

conform to N , for each open net N in Tab. 10, and subsequently test for 1-conformance using our testing approach.

For each open net N in Tab. 10, we create the event log $Fail(N)$ by modifying the already created event log $Succceed(N)$: We manually add an average-length trace that models the consecutively sending of two identical messages from N to its environment (i.e., N puts two tokens on an output place). As a consequence, any implementation exhibiting the observed behavior in $Fail(N)$ certainly violates the bound 1. In other words, the event log $Fail(N)$ clearly derives from an unknown implementation that does not 1-conform to N , and replaying $Fail(N)$ on $env(mp_1(max_1(N)))$ should always result in costs higher than 0. Table 15 gives an overview over the size of the resulting event logs.

event log	event traces	events	events per event trace
<i>Fail(D)</i>	401	3,247	8.10
<i>Fail(D')</i>	401	3,489	8.70
<i>Fail(U)</i>	401	3,733	9.31
<i>Fail(U')</i>	401	3,429	8.55
<i>Fail(CN)</i>	401	3,422	8.53
<i>Fail(LA)</i>	401	3,423	8.54
<i>Fail(PO)</i>	401	3,318	8.27
<i>Fail(RS)</i>	401	3,417	8.52
<i>Fail(TR)</i>	401	3,426	8.54

Table 15: The size of the event logs that we created by modifying the event logs from Tab. 13. Each event log *Fail(N)* takes the viewpoint of *N*'s environment and contains observed behavior from an unknown implementation that does not 1-conform to *N*.

Example 179 Consider again the open net *D* from Fig. 94a. In Sect. 8.4.2, we generated the event log *Succeed(D)* that contains 400 event traces and 3,239 events of an unknown implementation that 1-conforms to *D*; the average length of an event trace in *Succeed(D)* is 8. We manually add the trace *qdddqddd* of length 8 as an event trace to *Succeed(D)*, yielding the event log *Fail(D)* with 401 event traces and 3,247 events. The event trace *qdddqddd* “breaks” the bound 1 of any open net that 1-conforms to *D*, as already argued in Ex. 94. Therefore, *Fail(D)* certainly contains observed behavior from an implementation that does not 1-conform to *D*. \diamond

For each of the nine open nets *N* in Tab. 10, we test whether an unknown implementation—that is, an open net that may exhibit behavior captured in the event log *Fail(N)*—1-conforms to *N* by replaying *Fail(N)* on the labeled net $env(mp_1(max_1(N)))$. As in Sect. 8.4.2, we use the package “PNetReplayer” of the tool ProM with standard settings and the cost function from Conv. 9. Table 16 shows the results of these tests: In each case, the cost for replaying the event log *Fail(N)* on the labeled net $env(mp_1(max_1(N)))$ is greater than 0. Therefore, by Thm. 177, we can conclude that the tested implementation does not 1-conform to the specification in each case, which is exactly the result we expected. The runtime of replaying *Fail(N)* on $env(mp_1(max_1(N)))$ is, except for one case, slightly higher than the runtime of replaying the event log *Succeed(N)* on $env(mp_1(max_1(N)))$ in Sect. 8.4.2.

Example 180 Figure 102 depicts the open net $mp_1(max_1(D))$ and its environment, the labeled net $env(mp_1(max_1(D)))$. In Ex. 179, we added the event trace *qdddqddd* to the event log *Fail(D)*. An alignment of *qdddqddd* to $env(mp_1(max_1(D)))$ is, for example, the following alignment γ_6 :

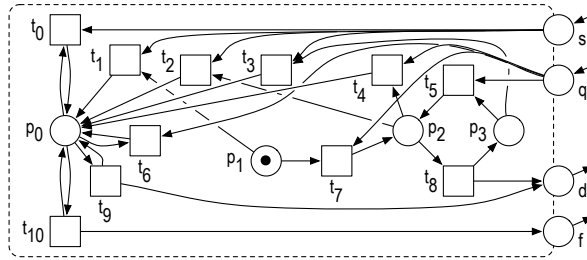
$$\gamma_6 = \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c} \gg & \gg & q & \gg & d & \gg & d & \gg & d & \gg & q & d & \dots \\ s & \tau & q & \tau & d & \tau & d & \tau & d & \tau & q & d & \dots \\ s & t_1 & q & t_9 & d & t_9 & d & t_9 & d & t_9 & q & d & \dots \end{array}$$

The alignment γ_6 of *qdddqddd* to $env(mp_1(max_1(D)))$ has costs of $\kappa(\gamma_6) = 1$ and is cost-minimal. Because the costs of $\kappa(\gamma_6)$ are greater than 0, we conclude by Thm. 177 that any open net which may produce *Fail(D)* does not 1-conform to the open net *D*.

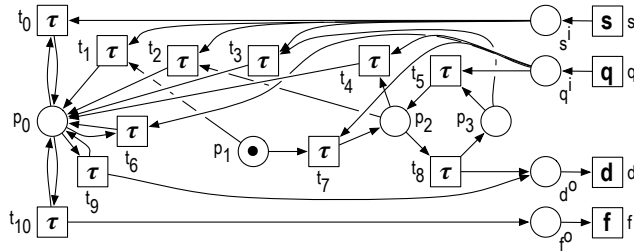
event log	labeled net	replay costs	time (s)
$Fail(D)$	$env(mp_1(max_1(D)))$	0.0024	1.288
$Fail(D')$	$env(mp_1(max_1(D')))$	0.0199	1.181
$Fail(U)$	$env(mp_1(max_1(U)))$	0.0224	1.432
$Fail(U')$	$env(mp_1(max_1(U')))$	0.0199	0.894
$Fail(CN)$	$env(mp_1(max_1(CN)))$	0.0199	688.702
$Fail(LA)$	$env(mp_1(max_1(LA)))$	3.1172	1.256
$Fail(PO)$	$env(mp_1(max_1(PO)))$	0.0199	70.266
$Fail(RS)$	$env(mp_1(max_1(RS)))$	0.0579	3,701.107
$Fail(TR)$	$env(mp_1(max_1(TR)))$	0.0200	55.380

Table 16: The time and cost ProM reported for replaying the event log $Fail(N)$ on the labeled net $env(mp_1(max_1(N)))$. We used the standard settings of the package “PNetReplayer” and the cost function from Conv. 9.

Figure 103 shows a screenshot of ProM visualizing the alignments of $Fail(D)$ to $env(mp_1(max_1(D)))$. The alignment γ_6 of $qdddqddd$ to $env(mp_1(max_1(D)))$ is depicted in the middle of the screen: All non-silent model moves are shown in violet, and all synchronous moves are shown in green. We can even recognize the first move of γ_6 : a non-silent model move using the s -labeled transition of $env(mp_1(max_1(D)))$. \diamond



(a) Open net $mp_1(max_1(D))$



(b) Labeled net $env(mp_1(max_1(D)))$

Figure 102: The open net $mp_1(max_1(D))$ that we obtained from the open net D in Fig. 94a using the tool Chloe, and its environment. In addition to the figures, we have $\Omega_{mp_1(max_1(D))} = \Omega_{env(mp_1(max_1(D)))} = \emptyset$.

Like traditional model checking [67], our testing approach also presents a counterexample if a test fails: Any event trace w whose replay costs on $env(mp_b(max_b(N)))$ are greater than 0 may be analyzed for log moves or non-silent model moves (i.e., any move with costs greater than 0). The



Figure 103: Visualizing the alignments of $Fail(D)$ to $env(mp_1(max_1(D)))$ in ProM. The alignment γ_6 from Ex. 180 is depicted in the middle of the screen: All non-silent model moves are shown in violet, and all synchronous moves are shown in green.

problem can be diagnosed by highlighting the prefix of w which leads to a trace that is not in the language of any open net that b -conforms to N .

Example 181 For our running example, consider again the alignment γ_6 of $qdddqddd$ to $env(mp_1(max_1(D)))$ from Ex. 180: γ_6 maps $qdddqddd$ to a trace of $env(mp_1(max_1(D)))$ that breaks the bound 1 by producing more than 1 token in the place q^i . From this, we can conclude that any open net that may produce $qdddqddd$ does not 1-conform to D because of a possible bound-violation. \diamond

8.5 CONCLUSIONS

Given a formal model of the specification of an open system and observed behavior of its running but unavailable implementation in the form of an event log, testing for conformance can show that the implementation does not conform to the specification if the event log contains some erroneous behavior. In this chapter, we presented a testing approach for the b -conformance relation from Chap. 5. To this end, we elaborated a necessary condition for b -conformance of the implementation $Impl$ to the specification $Spec$ based on the open net mp_b of the maximal b -partner of $Spec$: If the event log cannot be replayed on $env(mp_b(max_b(Spec)))$, then $Impl$ does not b -conform to $Spec$. We showed the existence of the open net $mp_b(max_b(Spec))$ and demonstrated that it can be automatically constructed, thereby using the theory and tools of Chap. 5. Our tool chain completely relied on free and open source software (i.e., the tools Chloe [115], Locretia [116], and ProM [212]) and proved to be feasible on a normal (personal) computer.

This chapter is based on results published in [191].

IN the previous chapter, we presented our first contribution to the log-model scenario: We elaborated a testing approach for b -conformance; that is, if there is some erroneous behavior captured by the given event log, we can conclude that the unknown implementation does not b -conform to the given specification and provide meaningful diagnostics. In this chapter, we further support the design of responsive open systems in the log-model scenario by discovering a formal model of the unknown implementation based on the given event log. The discovered formal model may help to explain and understand the running implementation, to further study its interaction with other open systems, to make predictions about its behavior [102], and to analyze its performance by simulation [220] or Markov-chain analysis [131, 82], for example.

Figure 104 illustrates our approach to discover a formal model $Impl$ of the unknown implementation from the given event log (i.e., its formalization Log), assuming that the unknown implementation b -conforms to its specification. As in Chap. 8, we consider only the b -conformance relation because b -conformance is, in contrast to the conformance relation in Chap. 4, decidable. To judge the discovered model we consider two aspects:

1. b -conformance (i.e., $Impl$ b -conforms to the model $Spec$ of the given specification), and
2. model quality (i.e., the ability of $Impl$ to describe the observed behavior in Log well according to different quality dimensions).

In other words, we search for a high-quality model in the set of all b -conforming open nets to the given specification. Thereby, our search space—the set of all b -conforming open nets—is in general infinite, and measuring the quality of a model with respect to an event log is a highly complex task.

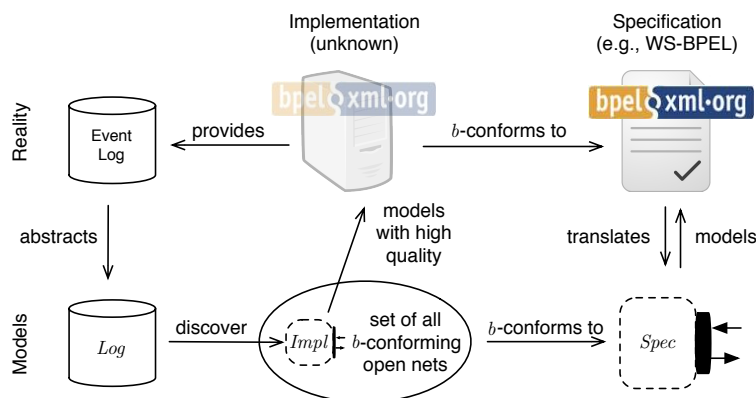


Figure 104: Discovering a high-quality model of the implementation in the log-model scenario.

We employ the finite characterization of all b -conforming open nets that we developed in Chap. 5 to elaborate a discovery procedure in Sect. 9.1.

Thereby, we also formalize and measure model quality with respect to an event log. In Sect. 9.2, based on this finite characterization, we additionally provide a suitable abstraction technique to improve the discovery procedure. We use the implemented decision algorithm for b -conformance from Chap. 5 to develop a genetic algorithm to discover a high-quality model of the implementation in Sect. 9.3. We evaluate the implemented algorithm with industrial-sized specifications and event logs in Sect. 9.4 and finish this chapter with a conclusion in Sect. 9.5.

9.1 THE DISCOVERY PROCEDURE

Given an open net $Spec$ and an event log Log , discovery aims to produce an open net $Impl$ that (1) b -conforms to $Spec$ and (2) adequately captures what was observed in Log . We address both requirements in the following.

9.1.1 Discovering a b -conforming open net

In this section, we show how to discover a b -conforming open net $Impl$ to a given open net $Spec$. Figure 105 illustrates the focus of this section.

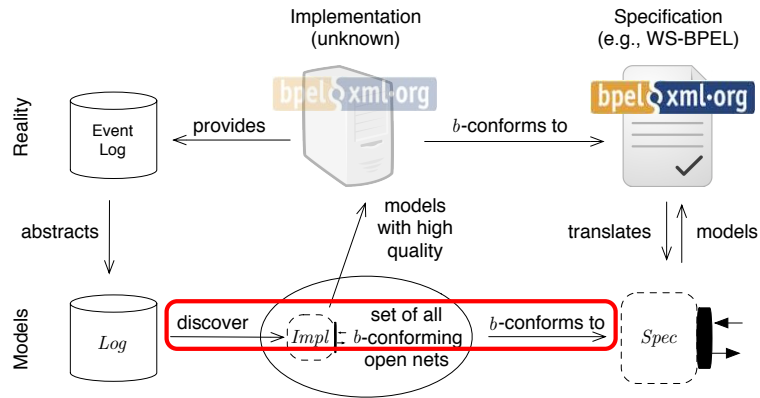


Figure 105: The focus of Sect. 9.1.1.

Given an open net $Spec$, checking whether some interface-equivalent open net $Impl$ b -conforms to $Spec$ reduces to checking whether $Impl$ matches with $MP_b(max_b(Spec))$ by Prop. 135. We can compute $MP_b(max_b(Spec))$ and check for b -conformance as illustrated in Fig. 79. As a consequence, the search space for our discovery procedure reduces to all b -conforming open nets to $Spec$ rather than any interface-equivalent open net of $Spec$. However, the search space is still infinite due to internal, unobservable actions: If an open net N has at least one b -conforming open net, then there exist infinitely many open nets that b -conform to N —a fact, which we already discussed in Chap. 3.

9.1.2 Discovering a high-quality open net

In the previous section, we restricted the search space for our discovery procedure to the open nets that b -conform to $Spec$. Next, we are interested in a b -conforming open net of highest quality. In this section, we formalize measures for the quality of an open net with respect to a given event log. Figure 106 illustrates the focus of this section.

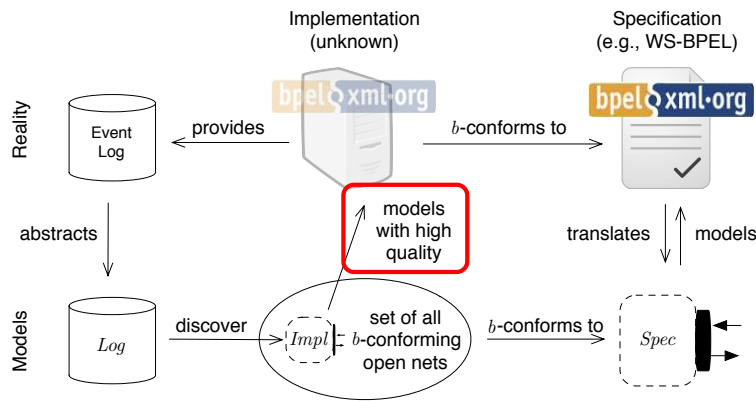


Figure 106: The focus of Sect. 9.1.2.

We adapt the idea of quantifying quality by measuring *quality dimensions*: Van der Aalst [2] describes four quality dimensions for general process models with respect to an event log in the area of process mining:

1. *fitness* (i.e., the discovered model should allow the behavior seen in the event log),
2. *simplicity* (i.e., the discovered model should be as simple as possible),
3. *generalization* (i.e., the discovered model should avoid overfitting by generalizing the example behavior seen in the event log), and
4. *precision* (i.e., the discovered model should avoid underfitting by not allowing behavior completely unrelated to what was seen in the event log).

These quality dimensions may compete with each other, as visualized in Fig. 10. For example, to improve the fitness of a model one may end up with a substantially more complex—that is, less simple—model. In addition, a more general model usually means a less precise model.

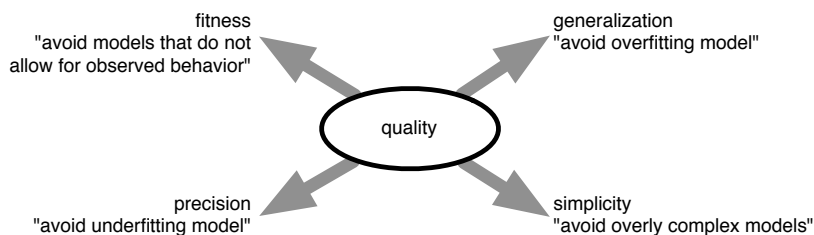


Figure 107: The different quality dimensions for model discovery.

In the area of process mining, numerous measures for the four quality dimensions have been developed [219, 5, 16]. In the following, we lift measures for fitness, simplicity, generalization, and precision from process models to models of open systems (i.e., open nets), and briefly compare them with the state-of-the-art in process mining. Thereby, whenever we measure a quality dimension between an event log and a labeled net, we automatically lift this definition to any open net via the open net's replay environment.

Convention 11 Throughout the remainder of this thesis, each quality measure between an event log Log and a labeled net is implicitly extended to any open net N via $replay(N)$. \diamond

9.1.2.1 Measuring fitness

Fitness indicates how much of the behavior in the event log is captured by the model. A labeled net N with good fitness allows for most of the behavior seen in the event log Log . We redefine the cost-based fitness measure from [5] for labeled nets: We quantify fitness as the total costs of aligning Log to N compared to the worst costs of aligning Log to N . Thereby, we compute the costs of aligning Log to N using the optimal alignment provided by the oracle function \mathbb{O}_N from Def. 171. The worst costs of aligning Log to N are bounded by the costs of the worst alignments—that is, just moves in the log and no moves in the model, for all event traces in Log . For the moves in the log only, we consider the “least expensive path” because an optimal alignment will always try to minimize costs [5], as formalized in Def. 168.

Definition 182 [fitness]

We define the *fitness* of an event log Log and a labeled net N as

$$fit(Log, N) = 1 - \frac{cost(Log, N)}{move(Log)}, \text{ where}$$

- $cost(Log, N) = \sum_{w \in Log} (Log(w) \cdot \kappa(\mathbb{O}_N(w)))$ are the total costs of aligning Log to N , and
- $move(Log) = \sum_{w \in Log} (Log(w) \cdot \sum_{v \in \Sigma^* \wedge x \in \Sigma \wedge vx \sqsubseteq w} \kappa((x, \gg)))$ are the total costs of moving through Log without ever moving in N .

We illustrate the fitness measure from Def. 182 and all following measures for the other three quality dimensions by measuring the quality of the database D and the patched database D' from Sect. 3.2. For convenience, we depict D and D' again in Fig. 108.

Example 183 As a running example for this chapter, consider again the event log $DLog$ from Chap. 8; for convenience, we depict it again in Tab. 17. The event log $DLog$ contains information on 210 traces. There are three types of traces: qd (100 times), qqd (100 times), and sfd (10 times). For all examples in this chapter, we assume that $DLog$ takes the viewpoint of D and D' , thus $replay(D) = inner(D)$ and $replay(D') = inner(D')$. Figure 108 also depicts the replay environments of D and D' .

There exist the following three cost-minimal alignments of the event traces qd , qqd , and sfd in $DLog$ to $replay(D)$:

$$\gamma_1 = \begin{array}{c|c} q & d \\ \hline q & d \\ \hline process & retrieve \end{array}$$

$$\gamma_2 = \begin{array}{c|c|c} q & q & d \\ \hline q & \gg & d \\ \hline process & & retrieve \end{array}$$

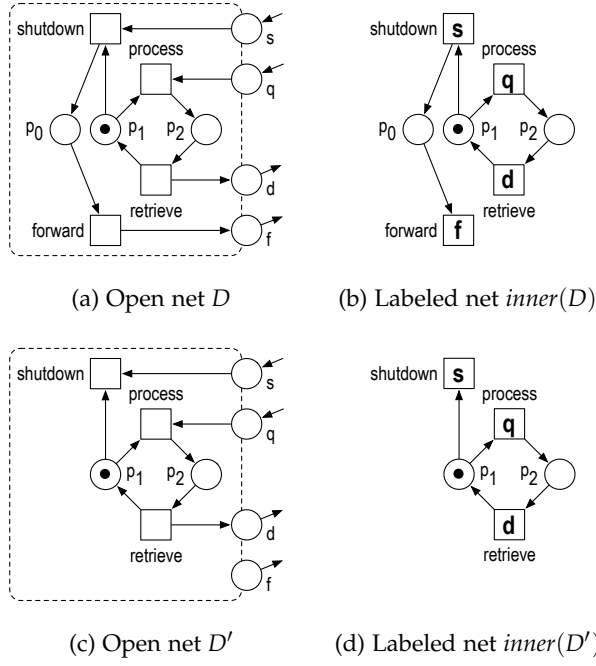


Figure 108: The open nets D and D' and the labeled nets $inner(D)$ and $inner(D')$ from Sect. 3.2. In addition to the figures, we have $\Omega_D = \Omega_{inner(D)} = \{[p_0]\}$ and $\Omega_{D'} = \Omega_{inner(D')} = \{[]\}$.

$$\gamma_3 = \begin{array}{c|c|c} s & f & d \\ \hline s & f & \gg \\ \hline shutdown & forward & \end{array}$$

We have $\kappa(\gamma_1) = 0$, $\kappa(\gamma_2) = 1$, and $\kappa(\gamma_3) = 1$. Therefore, the total costs of aligning $DLog$ to $replay(D)$ are $cost(DLog, replay(D)) = 100 \cdot 0 + 100 \cdot 1 + 10 \cdot 1 = 110$. The total costs of moving through $DLog$ without ever moving in $replay(D)$ are $move(DLog) = 100 \cdot 2 + 100 \cdot 3 + 10 \cdot 3 = 530$ (worst-case scenario). Thus, the fitness of $DLog$ and $replay(D)$ is $fit(DLog, replay(D)) = 1 - \frac{110}{530} \approx 0.7925$. \diamond

cardinality	event trace
100	qd
100	qqd
10	sfd

Table 17: The event log $DLog$.

Example 184 For the patched database D' , there exist the following three cost-minimal alignments of the event traces qd , qqd , and sfd in $DLog$ to $replay(D')$:

$$\gamma_4 = \begin{array}{c|c} q & d \\ \hline q & d \\ \hline process & retrieve \end{array}$$

$$\gamma_5 = \frac{q}{q} \mid \frac{q}{\gg} \mid \frac{d}{d}$$

process retrieve

$$\gamma_6 = \frac{s}{s} \mid \frac{f}{\gg} \mid \frac{d}{\gg}$$

shutdown

We have $\kappa(\gamma_4) = 0$, $\kappa(\gamma_5) = 1$, and $\kappa(\gamma_6) = 2$. Therefore, the total costs of aligning $DLog$ to $replay(D')$ are $cost(DLog, replay(D')) = 100 \cdot 0 + 100 \cdot 1 + 10 \cdot 2 = 120$. The total costs of moving through $DLog$ without ever moving in $replay(D')$ are $move(DLog) = 100 \cdot 2 + 100 \cdot 3 + 10 \cdot 3 = 530$ (worst-case scenario). Thus, the fitness of $DLog$ and $replay(D')$ is $fit(DLog, replay(D')) = 1 - \frac{120}{530} \approx 0.7736 < fit(DLog, replay(D))$. In other words, $DLog$ fits better to $replay(D)$ than to $replay(D')$ because of the trace sfd . \diamond

9.1.2.2 Measuring simplicity

Simplicity refers to open nets minimal in structure, which clearly reflect the log's behavior. This dimension is related to Occam's Razor [9], which states that "one should not increase, beyond what is necessary, the number of entities required to explain anything." There exist various techniques to quantify model complexity and, therefore, simplicity; see [173] for an overview. We define the simplicity of an open net by the size of its state-space, i.e., the number of states and transitions of the reachability graph in the underlying inner net. Remember that we always discover an open net $Impl$ that b -conforms to an open net $Spec$. Thus, we measure the difference in size between $Impl$ and a minimal open net C with the same behavior as $Impl$ (i.e., C also b -conforms to $Spec$). A minimal open net C with the same behavior as $Impl$ also matches with $MP_b(max_b(Spec))$; the size of the reachability graph of C 's inner net is the smallest subsystem G of $MP_b(max_b(Spec))$ such that $RG(inner(Impl))$ is weakly simulated by G .

Definition 185 [simplicity]

We define the *simplicity* of an open net $Impl$ that b -conforms to an open net $Spec$ as

$$sim(Impl, Spec) = \begin{cases} \frac{|Q_G| + |\delta_G|}{|P_{inner(Impl)}| + |T_{inner(Impl)}|}, & \text{if } \frac{|Q_G| + |\delta_G|}{|P_{inner(Impl)}| + |T_{inner(Impl)}|} \leq 1 \\ 1, & \text{otherwise,} \end{cases}$$

where G is the smallest LTS such that $G \subseteq MP_b(max_b(Spec))$ and $RG(inner(Impl))$ is weakly simulated by G .

By comparing the size of the subsystem G in Def. 185 with the size of $inner(Impl)$ (i.e., a labeled net) instead of comparing the size of G with the size of $RG(inner(Impl))$ (i.e., an LTS), we implicitly incorporate the following idea: The complexity of an open net arises not only from the size of the reachability graph of its inner net, but also from its own size in terms of places and transitions. Two different open nets N_1 and N_2 with identical reachability graph of their inner net, respectively, can be compared and are,

in general, not equally simple. The difference between N_1 and N_2 may arise because of τ -labeled and/or duplicated transitions, for example.

Example 186 Remember that D' b -conforms to D , but D does not b -conform to D' . For our running example, we consider D as the specification, and aim to discover an open net that 1-conforms to D while simultaneously exhibiting high quality with respect to $DLog$. Figure 109 depicts again the LTS $MP_1(max_1(D))$ from Sect. 5.3. For the open net D , we depict $RG(inner(D))$ in Fig. 110a and highlight the smallest LTS G such that $G \subseteq MP_1(max_1(D))$ and $RG(inner(D))$ is weakly simulated by G in Fig. 110b. We have $|Q_G| + |\delta_G| = 4 + 6 = 10$ and $|P_{inner(D)}| + |T_{inner(D)}| = 3 + 4 = 7$. Thus, $sim(D, D) = 1$. For the open net D' , we depict $RG(inner(D'))$ in Fig. 110c and highlight the smallest LTS G such that $G \subseteq MP_1(max_1(D))$ and $RG(inner(D'))$ is weakly simulated by G in Fig. 110d. We have $|Q_G| + |\delta_G| = 4 + 5 = 9$ and $|P_{inner(D')}| + |T_{inner(D')}| = 2 + 3 = 5$. Thus, $sim(D', D) = 1$. In other words, both labeled nets $inner(D)$ and $inner(D')$ are simpler than a (state-machine) labeled net with equal reachability graph that we can construct from a subsystem of $MP_1(max_1(D))$. \diamond

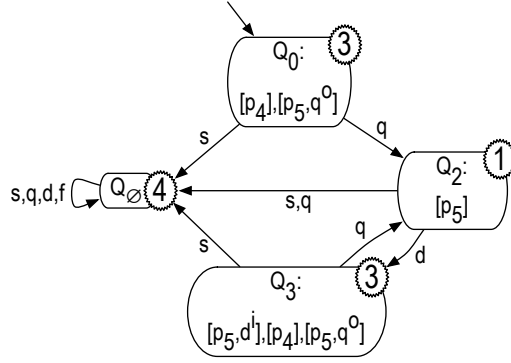


Figure 109: The LTS $MP_1(max_1(D))$ from Sect. 5.3. We depict the label of each state as an encircled number in the upper right corner of that state.

9.1.2.3 Measuring precision

Precision indicates whether a labeled net is not underfitting, i.e., by allowing for behavior unrelated to the behavior observed. To avoid underfitting, we prefer labeled nets with minimal behavior to represent the behavior observed in the event log as closely as possible. We redefine the alignment-based precision measures from [16] for labeled nets. This measure relies on building a tree-like LTS $AA(Log, N)$ that captures traces of a labeled net N that were used to align an event log Log to N . A state of $AA(Log, N)$ encodes a sequence of (labels of) transitions of N . For $AA(Log, N)$, the state labeling function λ serves as a *weight function*: We define the weight $\lambda(q)$ of each state q as the number of times a trace of Log was aligned to q . We shall use $AA(Log, N)$ also for measuring the generalization dimension later on.

Definition 187 [labeled transition system AA]

Let Log be an event log, and let N be a labeled net. We define the LTS $AA(Log, N) = (Q, \delta, q_{AA(N)}, \Sigma^{in}, \Sigma^{out}, \lambda)$ with

- $Q = \downarrow \{trace(\mathbb{O}_N(w)) \mid w \in Log\}$,

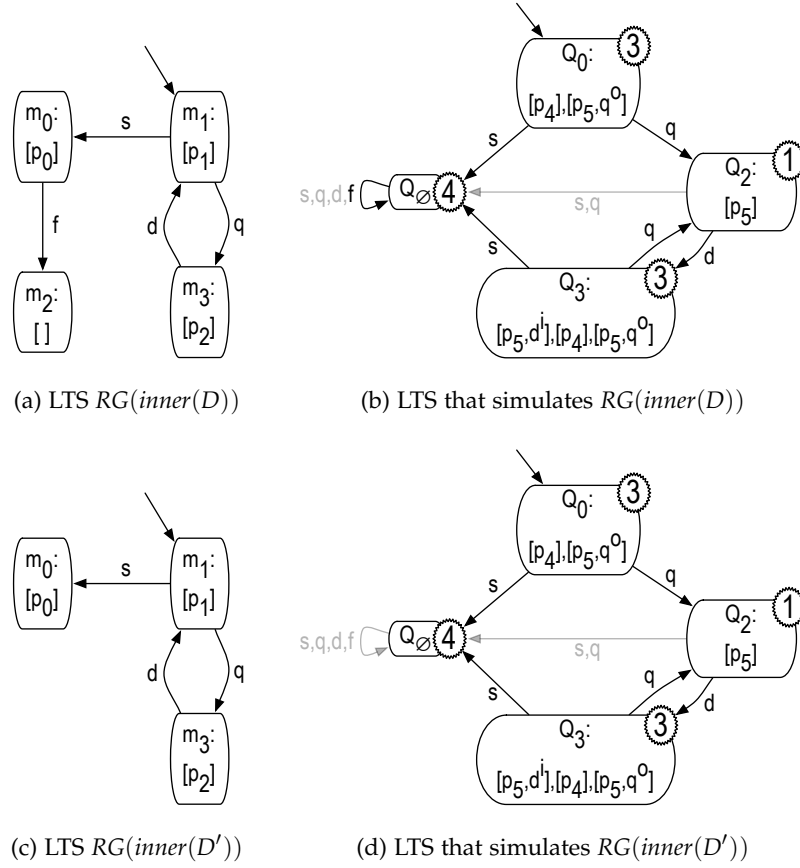


Figure 110: The reachability graphs of D and D' , and the smallest subsystem G of the LTS $MP_1(\text{max}_1(D))$ from Fig. 109 such that G weakly simulates $RG(\text{inner}(D))$ and $RG(\text{inner}(D'))$, respectively. The two LTSs on the right-hand side differ only in the f -labeled transition from the state Q_\emptyset to the state Q_\emptyset .

- $\delta = \{(q, x, qx) \in Q \times \Sigma \times Q \mid \exists w \in \text{Log} : qx \sqsubseteq \text{trace}(\mathcal{O}_N(w))\}$,
- $q_{AA(N)} = \varepsilon$, and
- $\lambda(q) = \sum_{w \in \text{Log} \wedge q \sqsubseteq \text{trace}(\mathcal{O}_N(w))} \text{Log}(w)$ for all $q \in Q$.

Example 188 The event log $D\text{Log}$ aligns to $\text{replay}(D)$ with the alignments γ_1 to γ_3 from Ex. 183. Thus, Fig. 111a depicts the LTS $AA(D\text{Log}, \text{replay}(D))$. In contrast, $D\text{Log}$ aligns to $\text{replay}(D')$ with the alignments γ_4 to γ_6 from Ex. 184. Thus, Fig. 111b depicts the LTS $AA(D\text{Log}, \text{replay}(D'))$. The LTS $AA(D\text{Log}, \text{replay}(D'))$ differs from the LTS $AA(D\text{Log}, \text{replay}(D))$ by the state sf because $sf \in L(\text{replay}(D))$ but $sf \notin L(\text{replay}(D'))$, and sf is used in alignment γ_3 . \diamond

For measuring precision, we relate executed and available actions after an aligned trace q of the event log with the help of $AA(\text{Log}, N)$. Thereby, we measure the executed actions after q as the number of unique actions that were observed at leaving state q —that is, the number of outgoing transitions of the state q of $AA(\text{Log}, N)$. We measure the available actions after q as the

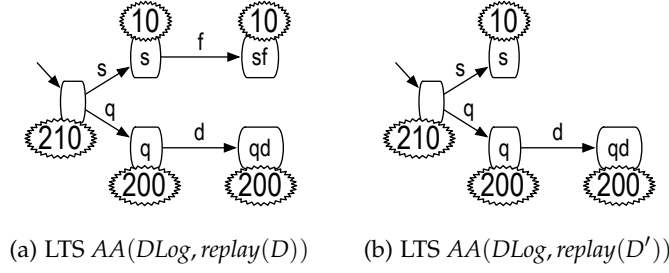


Figure 111: The LTSs $AA(DLog, replay(D))$ and $AA(DLog, replay(D'))$ for the event log $DLog$ from Tab. 17 and the labeled nets $replay(D)$ and $replay(D')$ from Fig. 108. We depict the label of each state as an encircled number over or under that state.

number of continuations (with one letter $x \in \Sigma$) of q that are available as traces of N .

Definition 189 [precision]

Let Log be an event log and let N be a labeled net. Let $AA(Log, N) = (Q, \delta, q_{AA(N)}, \Sigma^{in}, \Sigma^{out}, \lambda)$. We define the *precision* of Log and N as

$$pre(Log, N) = \frac{\sum_{q \in Q} (\lambda(q) \cdot |\{qx \in L(AA(Log, N)) \mid x \in \Sigma\}|)}{\sum_{q \in Q} (\lambda(q) \cdot |\{qx \in L(N) \mid x \in \Sigma\}|)}.$$

Example 190 Consider again the LTSs $AA(DLog, replay(D))$ and $AA(DLog, replay(D'))$ in Fig. 111. For the open net D in Fig. 108a, we have $pre(DLog, replay(D)) = \frac{210 \cdot 2 + 10 \cdot 1 + 10 \cdot 0 + 200 \cdot 1 + 200 \cdot 0}{210 \cdot 2 + 10 \cdot 1 + 10 \cdot 0 + 200 \cdot 1 + 200 \cdot 2} = \frac{630}{1030} \approx 0.6117$. The low precision of $DLog$ and $replay(D)$ results from the loop in $replay(D)$: Transitions *process* and *retrieve* can fire infinitely often in $replay(D)$.

For the open net D' in Fig. 108c, we have $pre(DLog, replay(D')) = \frac{210 \cdot 2 + 10 \cdot 0 + 200 \cdot 1 + 200 \cdot 0}{210 \cdot 2 + 10 \cdot 0 + 200 \cdot 1 + 200 \cdot 2} = \frac{620}{1020} \approx 0.6078$. As for $pre(DLog, replay(D))$, the low precision of $DLog$ and $replay(D')$ results from the loop in $replay(D')$ over the transitions *process* and *retrieve*. \diamond

9.1.2.4 *Measuring generalization*

Generalization penalizes overly precise open nets which overfit the given log. In general, a labeled net should not restrict behavior to just the behavior observed in the event log. Often only a fraction of the possible behavior has been observed. For the generalization dimension, we developed a new measure for an event log Log and a labeled net N : We combine the generalization measure from [5] with the LTS $AA(Log, N)$. The idea is to use the estimated probability $\pi(x, y)$ that a next visit to a state q of $AA(Log, N)$ will reveal a new trace that was not observed before: x is the number of unique actions observed at leaving state q (defined over the outgoing transitions of q in $AA(Log, N)$ as in Def. 189), and $y = \lambda(q)$ is the number of times q was visited by the event log. We employ an estimator for $\pi(x, y)$, which is inspired by [38].

Definition 191 [generalization]

Let Log be an event log and let N be a labeled net. Let $AA(Log, N) = (Q, \delta, q_{AA(N)}, \Sigma^{in}, \Sigma^{out}, \lambda)$. We define the *generalization* of Log and N as

$$gen(Log, N) = 1 - \left(\frac{1}{|Q|} \sum_{q \in Q} \pi(|\{qx \in L(AA(Log, N)) \mid x \in \Sigma\}|, \lambda(q)) \right),$$

where $\pi(x, y)$ can be approximated [5] by

$$\pi(x, y) = \begin{cases} \frac{x(x+1)}{y(y-1)}, & \text{if } y \geq x + 2 \\ 1, & \text{otherwise.} \end{cases}$$

Example 192 Consider again the LTSs $AA(DLog, replay(D))$ and $AA(DLog, replay(D'))$ in Fig. 111. For the open net D in Fig. 108a, we have $gen(DLog, replay(D)) = 1 - \frac{1}{5} \cdot (\pi(2, 210) + \pi(1, 10) + \pi(0, 10) + \pi(1, 200) + \pi(0, 200)) = 1 - \frac{1}{5} \cdot (0.0001 + 0.00005 + 0 + 0.00005 + 0) \approx 1$.

For the open net D' in Fig. 108c, we have $gen(DLog, replay(D')) = 1 - \frac{1}{5} \cdot (\pi(2, 210) + \pi(0, 10) + \pi(1, 200) + \pi(0, 200)) = 1 - \frac{1}{5} \cdot (0.0001 + 0 + 0.00005 + 0) \approx 1$. Thus, both open nets D and D' generalize the observed behavior in the event log $DLog$ by allowing for a large amount of unobserved behavior: For example, $replay(D)$ and $replay(D')$ additionally contain the traces $qdqd$ and $qdqdqd$ by generalizing the event trace qd from $DLog$ to a loop. \diamond

9.1.2.5 Valuing quality

To balance the four conflicting quality dimension, we also assume for each of the four quality dimensions a weight ω_{fit} , ω_{sim} , ω_{pre} , and ω_{gen} to be specified by a user. With these four weights, we can actually search for a b -conforming open net that has high or even highest quality. In general, there does not exist a process model that has the highest value for every dimension.

Definition 193 [quality]

Let Log be an event log and let $Impl$ and $Spec$ be two interface-equivalent open nets. Let $\omega_{all} = \omega_{fit} + \omega_{sim} + \omega_{pre} + \omega_{gen}$. The *quality* of Log , $Impl$, and $Spec$ is defined by

$$\begin{aligned} Q(Log, Impl, Spec) = & \frac{\omega_{fit}}{\omega_{all}} fit(Log, replay(Impl)) \\ & + \frac{\omega_{sim}}{\omega_{all}} sim(Impl, Spec) \\ & + \frac{\omega_{pre}}{\omega_{all}} pre(Log, replay(Impl)) \\ & + \frac{\omega_{gen}}{\omega_{all}} gen(Log, replay(Impl)). \end{aligned}$$

Example 194 For our running examples, we assume weights of 1 for each quality dimension. The open net D in Fig. 108a always serves as the specification. For the open net D as implementation, we have $Q(DLog, D, D) = \frac{1}{4} \cdot 0.7925 + \frac{1}{4} \cdot 1 + \frac{1}{4} \cdot 0.6117 + \frac{1}{4} \cdot 1 \approx 0.8511$.

For the open net D' in Fig. 108c as implementation, we have $Q(DLog, D', D) = \frac{1}{4} \cdot 0.7736 + \frac{1}{4} \cdot 1 + \frac{1}{4} \cdot 0.6078 + \frac{1}{4} \cdot 1 \approx 0.8454$. Hence, we conclude that the quality of $DLog$ and D is slightly better than the quality of $DLog$ and D' . In other words, the open net D explains better than the open net D' what we have seen in the event log $DLog$. \diamond

In the following section, we elaborate an abstraction technique for improving the discovery process.

9.2 IMPROVING THE DISCOVERY PROCEDURE WITH b -SUBNETS

Given an open net $Spec$, the search space for system discovery is infinite: Every open net $Impl$ that b -conforms to $Spec$ is a possible solution. For further improving the discovery procedure, we can restrict the search space to a *finite* number of open nets. To this end, we consider the LTS $MP_b(max_b(Spec))$ and restrict ourselves to open nets that match with $MP_b(max_b(Spec))$ and, in addition, whose inner net's reachability graph is an *initialized subsystem* of $MP_b(max_b(Spec))$. We refer to such an open net as a *b -subnet* of $Spec$.

Definition 195 [b -subnet]

Let $Impl$ and $Spec$ be two interface-equivalent open nets such that $MP_b(Spec)$ exists. Then $Impl$ is a *b -subnet* of $Spec$ if

1. $Impl$ matches with $MP_b(max_b(Spec))$,
2. $RG(inner(Impl))$ is an initialized subsystem of $MP_b(max_b(Spec))$, and
3. $Impl$ is structurally minimal, i.e., if we remove a place or a transition from $Impl$, then $RG(inner(Impl))$ changes.

Because of item (3) in Def. 195, there exists at most one b -subnet $Impl$ of $Spec$ for each initialized subsystem G of $MP_b(max_b(Spec))$; that is, $inner(Impl)$ is the state-machine labeled net of G . As the LTS $MP_b(max_b(Spec))$ is finite, the number of initialized subsystems of $MP_b(max_b(Spec))$ is finite and, therefore, the number of b -subnets of $Spec$ is finite, too. So instead of investigating any open net that b -conforms to $Spec$, we only consider b -subnets of $Spec$ as possible solutions.

Intuitively, a b -subnet $Impl$ of $Spec$ represents an equivalence class of open nets: For every open net N that b -conforms to $Spec$, there exists a smallest initialized subsystem G of $MP_b(max_b(Spec))$ such that $RG(inner(N))$ is weakly simulated by G ; a fixed initialized subsystem G of $MP_b(max_b(Spec))$ (and, thus, $Impl$ with $RG(inner(Impl)) = G$) represents all such N .

Subsequently, we discuss the implications of this restriction to b -subnets.

By Prop. 135, this directly implies that a b -subnet $Impl$ b -conforms to $Spec$.

Proposition 196 [b -subnet]

Any b -subnet $Impl$ of an open net $Spec$ b -conforms to $Spec$.

As a second implication, a b -subnet $Impl$ of an open net $Spec$ is internally bounded, i.e., $RG(inner(Impl))$ is always finite because $MP_b(max_b(Spec))$ is always finite.

In the following, we restricted the search space for system discovery to the finite set of b -subnets of $Spec$. This finite abstraction comes at a price: We may have excluded open nets that b -conform to $Spec$ and have

a higher quality than any b -subnet of $Spec$. Basically, we exclude open nets whose inner net's reachability graph contains an unfolding of a cycle of $MP_b(max_b(Spec))$.

Example 197 Consider again the open nets D and D' in Fig. 108. We already showed, for example in Ex. 98, that D' 1-conforms to D . However, D' is not a 1-subnet of D : The reachability graph $RG(inner(D'))$ is not an initialized subsystem of $MP_1(max_1(D))$ in Fig. 109. In other words, by restricting the search space for system discovery to 1-subnets of D , we cannot discover D' . Now consider the open net N_1 in Fig. 112a: N_1 1-conforms to D because every 1-partner of D perpetually communicates by sending a message q and subsequently receiving a message d (as we already detailed for D' and D in Ex. 49). In contrast to D' , the reachability graph $RG(inner(N_1))$ in Fig. 112c is an initialized subsystem of $MP_1(max_1(D))$: The states m_1 and Q_0 , m_1 and Q_2 , and m_2 and Q_3 coincide, respectively.

The difference between D' and N_1 illustrates how restricting the search space to b -subnets effects the structure of the discovered open nets in general: Rather than having a “furled” loop over some states (e.g., the states m_1 and m_3 in $RG(inner(D'))$), the reachability graph of the inner of a b -subnet may contain an “unrolling” of this loop instead (e.g., the states m_0 , m_1 , and m_2 in $RG(inner(N_1))$). Such an unrolling is, in essence, a sequence of states reaching the same (or an even greater) loop as the original one. \diamond

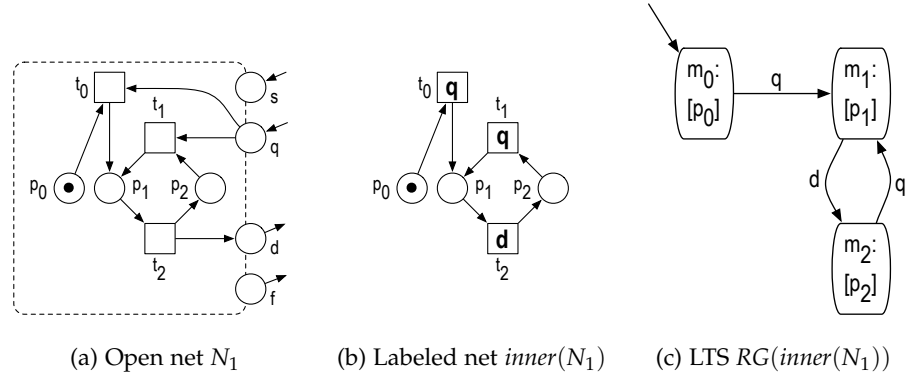


Figure 112: The open net N_1 , the labeled net $inner(N_1)$, and its reachability graph $RG(inner(N_1))$. In addition to the figures, we have $\Omega_{N_1} = \Omega_{inner(N_1)} = \{\{\}\}$.

In what follows, we discuss the impact of these “unfolded” b -conforming open nets on the quality of the discovered open nets with respect to a given event log Log . We illustrate the discussions with a series of technical examples like the open net N_1 in Fig. 112a.

9.2.1 Impact on the fitness dimension

From Def. 182, we conclude that fitness is preserved by the restriction to b -subnets: For every open net $Impl$ that b -conforms to an open net $Spec$, we can construct a b -subnet $Impl'$ of $Spec$ such that $Impl'$ has at least the fitness of $Impl$ with respect to any event log.

Lemma 198 [impact of b -subnets on fitness]

Let Log be an event log and let $Impl$ and $Spec$ be two interface-equivalent open nets such that $Impl$ b -conforms to $Spec$. Then there exists a b -subnet $Impl'$ of $Spec$ such that $fit(Log, replay(Impl)) \leq fit(Log, replay(Impl'))$.

Proof. If $Impl$ is a b -subnet of $Spec$, then the statement follows trivially, thus we assume $Impl$ is not a b -subnet of $Spec$. Let G denote the smallest subsystem of $MP_b(max_b(Spec))$ such that $RG(inner(Impl))$ is weakly simulated by G . The LTS G exists because $Impl$ matches with $MP_b(max_b(Spec))$ by Prop. 135 (i.e., there exists a weak simulation relation of $RG(inner(Impl))$ by $MP_b(max_b(Spec))$ according to Def. 123), and G is uniquely defined because $MP_b(max_b(Spec))$ is deterministic. We can construct an open net $Impl'$ from G in the sense of Def. 110, i.e., we have $RG(inner(Impl')) = G$, and in the same sense of Lem. 112, we can show that $Impl'$ is a b -partner of $max_b(Spec)$. Thus, $Impl'$ is a b -subnet of $Spec$. Because $RG(inner(Impl))$ is weakly simulated by $G = RG(inner(Impl'))$, every trace of $inner(Impl)$ is also a trace of $inner(Impl')$. Therefore, we have $fit(Log, replay(Impl)) \leq fit(Log, replay(Impl'))$ by Def. 182. \square

In other words, it suffices to consider only b -subnets of an open net $Spec$ for discovering a highly fitting open net that b -conforms to $Spec$ by Lem. 198.

Example 199 Consider again the open net D' in Fig. 108c, which is not a 1-subnet of the open net D in Fig. 108a. The reachability graph $RG(inner(D'))$ of its inner net $inner(D')$ is weakly simulated by the LTS G in Fig. 110d. Like in the proof of Lem. 198, we can construct an open net N_2 such that N_2 is a 1-subnet of D and $RG(inner(N_2)) = G$. We depict the resulting open net N_2 together with its inner net $inner(N_2)$ in Fig. 113. Clearly, N_2 1-conforms to D because no 1-partner of D sends a message s , as we already detailed in Ex. 49. By comparing their inner nets $inner(D')$ and $inner(N_2)$, we see that $L(inner(D')) \subseteq L(inner(N_2))$. Thus, $fit(Log, replay(D')) \leq fit(Log, replay(N_2))$ for every event log Log . \diamond

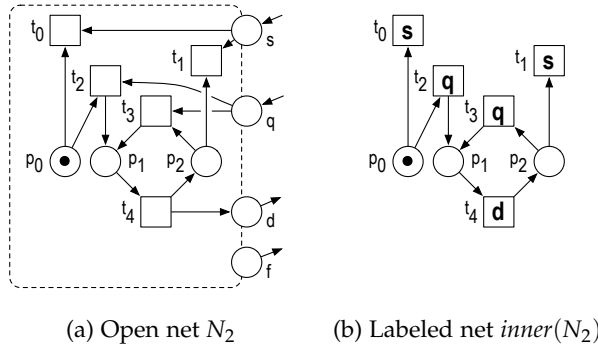


Figure 113: The open net N_2 and the labeled net $inner(N_2)$. In addition to the figures, we have $\Omega_{N_2} = \Omega_{inner(N_2)} = \{\{\}\}$.

9.2.2 Impact on the simplicity dimension

Technically, simplicity from Def. 185 is also preserved by the restriction to b -subnets.

Lemma 200 [impact of b -subnets on simplicity]

Let $Impl$ and $Spec$ be two interface-equivalent open nets such that $Impl$ b -conforms to $Spec$. Then there exists a b -subnet $Impl'$ of $Spec$ such that $sim(Impl, Spec) \leq sim(Impl', Spec)$.

Proof. The open net $Impl'$ that we constructed in Lem. 198 is a b -subnet of $Spec$ and the reachability graph of its inner net $inner(Impl')$ is identical to the smallest subsystem G of $MP_b(max_b(Spec))$ such that $RG(inner(Impl))$ is weakly simulated by G . Thus, we always have $sim(Impl', Spec) = 1$ by Def. 185. \square

By Lem. 200, it suffices to consider only b -subnets of an open net $Spec$ for discovering a simple open net that b -conforms to $Spec$. However, this is a mere sleight of hand: In essence, simplicity in Def. 185 compares the size of the open net $Impl$ with the size of a constructed b -subnet $Impl'$ of $Spec$ in terms of the reachability graph of their respective inner net. Thereby, the inner net of $Impl'$ is a subsystem of $MP_b(max_b(Spec))$ and seen as the reference point, i.e., the minimal size of any open net that b -conforms to $Spec$ with the same language as $Impl$. However, there may exist already “unrolled” loops in $MP_b(max_b(Spec))$ that limit the size of $Impl'$. In other words, we exclude b -conforming open nets from the search space which are obviously simpler, i.e., smaller in size, although the simplicity metric in Def. 185 does not reflect this. It is future work to modify the simplicity metric such that this discrepancy is taken into account.

Example 201 Consider again the open nets D' and N_2 in Fig. 108c and Fig. 113a. Both D' and N_2 are 1-conforming to the open net D in Fig. 108a. However, N_2 is a 1-subnet of D , whereas D' is not. In addition, $RG(inner(N_2))$ is the smallest subsystem G of $MP_1(max_1(Spec))$ that weakly simulates $RG(inner(D'))$ (cf. Fig. 110d). Obviously, D' is much smaller in size than N_2 . However, they both have a simplicity value of 1: For measuring the simplicity of D' , we have to compare the size of $RG(inner(D'))$ with the size of G according to Def. 185. As $\frac{|Q_G| + |\delta_G|}{P_{inner(D')} + T_{inner(D')}} = \frac{4+5}{2+3} > 1$, we have $sim(D', D) = 1$. On the other hand, as G is the reachability graph of $inner(N_2)$ and simulates itself, we have $sim(N_2, D) = 1$. \diamond

9.2.3 Impact on the precision dimension

In contrast to fitness and simplicity, precision from Def. 189 is not preserved by the restriction to b -subnets. We illustrate this with the following technical example:

Example 202 Consider the open net D in Fig. 108a, which of course 1-conforms to itself. However, D is not a 1-subnet of D , as $RG(inner(D))$ in Fig. 110a is not a subsystem of $MP_1(max_1(D))$ Fig. 109; rather, $RG(inner(D))$ is weakly simulated by a subsystem G of $MP_1(max_1(D))$ as illustrated in Fig. 110b. The open net N_3 in Fig. 114a represents G : the reachability graph of $inner(N_3)$ in Fig. 114b coincides with G . In other words, N_3 is the 1-subnet of D that comes closest to D 's behavior.

The open nets D and N_3 do not have the same LTS AA , and D has a higher precision than N_3 : $RG(inner(D))$ has an unfolded cycle of

$RG(inner(N_3))$ and thus $RG(inner(N_3))$ has more transitions enabled at the state after an event trace sf (i.e., the f -labeled transition may fire). \diamond

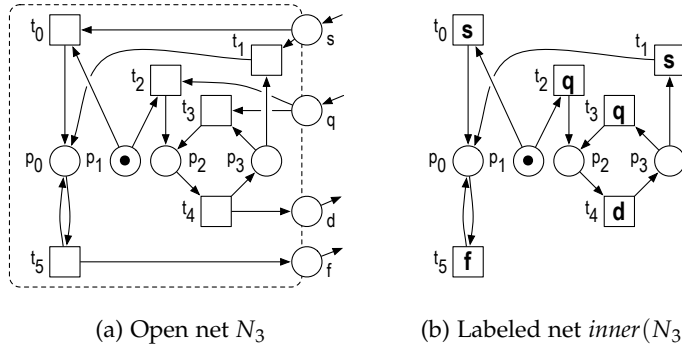


Figure 114: The open net N_3 and the labeled net $inner(N_3)$. In addition to the figures, we have $\Omega_{N_3} = \Omega_{inner(N_3)} = \{[]\}$.

Consequently, we may have excluded highly precise open nets that b -conform to an open net $Spec$ by restricting our search space for discovery to b -subnets of $Spec$ only.

9.2.4 Impact on the generalization dimension

Like precision, also generalization (see Def. 191) is not preserved by the restriction to b -subnets: The LTS AA used for measuring generalization is not the same for D and N_3 for some event log, which is the sole basis for our generalization metric.

We summarize the findings from Sect. 9.2.1 to Sect. 9.2.4 with the following corollary:

Corollary 203 [abstracting to b -subnets]

Restricting the search space of our discovery process to b -subnets preserves only fitness and simplicity.

Example 204 As a final example for the impact of b -subnets on the quality, consider the open net N_4 and its inner net $inner(N_4)$ in Fig. 115; N_4 is a 1-subnet of the open net D in Fig. 108a because N_4 matches with $MP_1(max_1(D))$ in Fig. 109 and $RG(inner(N_4))$ is an initialized subsystem of $MP_1(max_1(D))$. In the following, the event log $DLog$ in Tab. 17 contains the observed behavior and D serves as the specification: We then compare the quality of N_4 as implementation with the quality of the implementations D and D' from Ex. 194.

There exist the following three cost-minimal alignments of the event traces qd, qqd , and sfd from $DLog$ to $replay(N_4) = inner(N_4)$:

$$\gamma_7 = \begin{array}{c|c} q & d \\ \hline q & d \\ t_2 & t_4 \end{array}$$

$$\gamma_8 = \begin{array}{c|c|c} q & q & d \\ \hline q & q & d \\ t_2 & t_1 & t_5 \end{array}$$

$$\gamma_9 = \begin{array}{c|c|c} s & f & d \\ \hline s & \gg & d \\ \hline t_0 & & t_5 \end{array}$$

We have $\kappa(\gamma_7) = 0$, $\kappa(\gamma_8) = 0$, and $\kappa(\gamma_9) = 1$. Therefore, the total costs of aligning $DLog$ to $replay(N_4)$ are $cost(DLog, replay(N_4)) = 100 \cdot 0 + 100 \cdot 0 + 10 \cdot 1 = 10$. The total costs of moving through $DLog$ without ever moving in $replay(N_4)$ are $move(DLog) = 100 \cdot 2 + 100 \cdot 3 + 10 \cdot 3 = 530$ (worst-case scenario). Thus, the fitness of $DLog$ and $replay(N_4)$ is $fit(DLog, replay(N_4)) = 1 - \frac{10}{530} \approx 0.9811$, which is higher than the fitness of $fit(DLog, replay(D)) \approx 0.7925$ in Ex. 183 and $fit(DLog, replay(D')) \approx 0.7736$ in Ex. 184. This is because N_4 contains more observed behavior from $DLog$ (i.e., more event traces) than D and D' .

As N_4 is a 1-subnet of D , we have $sim(N_4, D) = 1$ by Def. 185; this coincides with the simplicity $sim(D, D) = sim(D', D) = 1$ in Ex. 186.

Figure 116 depicts the LTS $AA(DLog, replay(N_4))$. We have $pre(DLog, replay(N_4)) = \frac{210 \cdot 2 + 10 \cdot 1 + 10 \cdot 0 + 200 \cdot 2 + 100 \cdot 1 + 100 \cdot 0 + 100 \cdot 0}{210 \cdot 2 + 10 \cdot 1 + 10 \cdot 1 + 200 \cdot 2 + 100 \cdot 1 + 100 \cdot 1 + 100 \cdot 1} = \frac{930}{1140} \approx 0.8158$. Therefore, N_4 has a higher precision than D and D' in Ex. 190, whose precision is $pre(DLog, replay(D)) \approx 0.6117$ and $pre(DLog, replay(D')) \approx 0.6078$, respectively. Despite this difference in precision, N_4 still has a high generalization: We have $gen(DLog, replay(N_4)) = 1 - \frac{1}{7} \cdot (\pi(2, 210) + \pi(1, 10) + \pi(0, 10) + \pi(2, 200) + \pi(1, 100) + \pi(0, 100) + \pi(0, 100)) \approx 0.9968$, which is nearly as high as the generalization of D and D' with $gen(DLog, replay(D)) \approx 1$ and $gen(DLog, replay(D')) \approx 1$ in Ex. 192, respectively.

Using a weight of 1 for each quality dimension as in Ex. 194, we have $Q(DLog, N_4, D) = \frac{1}{4} \cdot 0.9811 + \frac{1}{4} \cdot 1 + \frac{1}{4} \cdot 0.8158 + \frac{1}{4} \cdot 0.9968 \approx 0.9484$ compared to $Q(DLog, D, D) \approx 0.8511$ and $Q(DLog, D', D) \approx 0.8454$ in Ex. 194. In other words, the open net N_4 , which is a 1-subnet of D , explains better than the open nets D and D' what we have seen in the event log $DLog$. \diamond

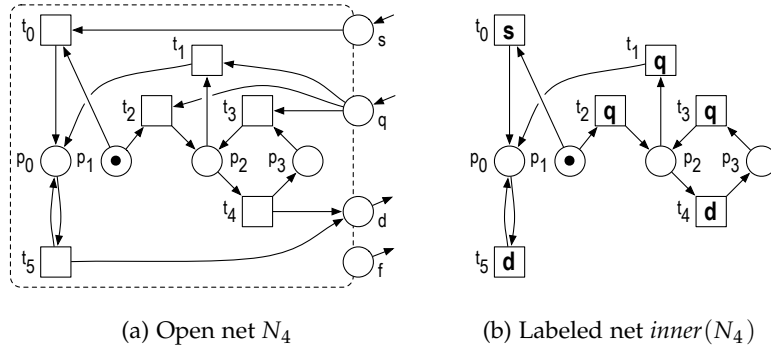


Figure 115: The open net N_4 and the labeled net $inner(N_4)$. In addition to the figures, we have $\Omega_{N_4} = \Omega_{inner(N_4)} = \{\{\}\}$.

9.3 IMPLEMENTATION

In this section, we describe the implementation of our discovery approach.

Given an open net $Spec$ and an event log Log recording behavior between $Spec$ and its environment, we aim to discover an open net $Impl$ that b -conforms to $Spec$ and has high quality with respect to $Spec$ and Log . As moti-

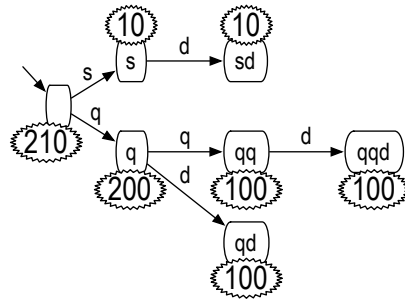


Figure 116: The LTS $AA(DLog, replay(N_4))$ for the event log $DLog$ from Tab. 17 and the labeled net $replay(N_4)$ from Fig. 115b. We depict the label of each state as an encircled number over or under that state.

vated at the beginning of this chapter, the search space (i.e., the number of open nets that b -conform to $Spec$) is infinite. Even if we further improve the discovery process by restricting the search space to b -subnets (see Sect. 9.2), it may still be too large to search for an optimal candidate exhaustively. We illustrate the enormous size of the search space restricted to b -subnets with the following example: Consider the industrial-sized open net LA with 34 places (3 input places, 3 output places) and 17 transitions that we derived from the WS-BPEL process “Loan Approval” in Sect. 5.4 and Sect. 8.4. Counting the number of 1-subnets of LA can be reduced to a propositional model counting problem by encoding the structure of $MP_1(LA)$ and Def. 195 into a propositional formula. Propositional model counting, also known as #SAT, is the problem of counting the number of models (i.e., satisfying truth assignments) of a given propositional formula [242]. Using the sharpSAT solver [237], we find that there exist 119, 803, 403, 352, 641, 974, 351 1-subnets of LA . In other words, if we unrealistically assume that we can evaluate the quality of one billion 1-subnets of LA per second (with respect to some event log Log), we still need over 3,798 years to exhaustively search for the 1-subnet of LA that has the highest quality with respect to $Spec$ and Log . Note that the open net LA is the smallest of the five industrial-sized examples used in Sect. 5.4 and Sect. 8.4.

Because of the huge search space, we are using a search heuristic in form of a *genetic algorithm* [89] to discover an open net $Impl$ that has a high but possibly not the maximal quality with respect to Log . Genetic algorithms have been successfully applied to discover process models [171, 52, 53].

A genetic algorithm evolves a population of candidate solutions (i.e., the *individuals*) step-wise (i.e., in *generations*) toward better solutions of an optimization problem. In our setting, an individual is either an open net that matches with $MP_b(max_b(Spec))$ (in the discovery process without the improvement from Sect. 9.2) or a b -subnet of $Spec$ (in the discovery process with the improvement from Sect. 9.2). In both cases, the quality of a candidate solution is determined by the quality (see Def. 193) with respect to Log and $Spec$, as every b -subnet of $Spec$ is an open net as well.

Our algorithm employs the general procedure of genetic algorithms [89], which we depict in Fig. 117:

1. Choose the initial population (i.e., the first generation) of individuals. These are randomly generated individuals (either open nets that match with $MP_b(max_b(Spec))$, or b -subnets of $Spec$). The size of the initial population is part of the input parameters of the algorithm.

2. The algorithm repeats the following steps until a termination criterion is satisfied:
 - a) Compute the quality of each individual in this generation, using Def. 193.
 - b) *Elitism*: Directly shift a proportion of the individuals with the highest quality into the next generation.
 - c) Select all individuals of the current generation for breeding: Create new individuals (called *children*) through crossover, mutation, and replacement operations. The crossover operation randomly exchanges parts between two given individuals. The mutation operation randomly adds or removes a transition or a final state from a given individual. The replacement operation replaces a randomly chosen individual by a new, randomly generated individual. The probabilities for each operation are part of the input parameters of the algorithm.
 - d) Evaluate the quality of each newly breed individuals.
 - e) Replace the individuals with the least quality in the current generation with high-quality newly breed individuals. They, together with the initially shifted elite individuals, form the new generation.
3. If at least one termination criterion is satisfied, return the individual with the highest quality of the latest generation.

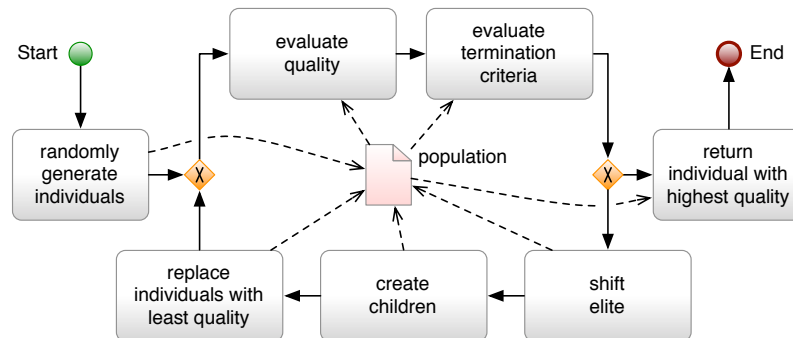


Figure 117: A BPMN diagram that illustrates the genetic algorithm.

We employ a combination of four different termination criteria to determine when to terminate evolution:

1. A time limit stops the evolution after a certain amount of time, regardless how far the individuals have been evolved.
2. A generation count stops the evolution after a certain number of generations.
3. A stagnation count stops the evolution after a certain number of generations without finding better candidate models than the best one found thus far.
4. A sufficient quality criterion stops the evolution if the quality of the current generation's highest-quality individual exceeds a specified threshold.

We have implemented the genetic algorithm, both with and without the improvement from Sect. 9.2, in Java as a ProM plug-in [188]. ProM [212] is an extensible framework that supports a wide variety of process mining techniques; we already used the PNetReplayer plug-in in ProM for our testing approach in Sect. 8.3. Our implementation uses the Watchmaker framework [87], which is a free and open source framework for implementing platform-independent genetic algorithms in Java. Thus, our implementation completely relies on free and open source software.

For merely technical reasons—that is, to avoid storing the set of markings of each state of $MP_b(max_b(Spec))$ —the implementation stores the necessary information for matching an open net $Impl$ with $MP_b(max_b(Spec))$ into Boolean formulae with whom we annotate every state of $MP_b(max_b(Spec))$. The resulting state-annotated LTS technically resembles the operating guidelines of Lohmann et al. [153, 156].

Definition 205 [MP_b with Boolean annotation]

Let N be an open net such that $MP_b(N)$ exists. The *Boolean annotation* of $MP_b(N)$ is a function ϕ that assigns to each state Q of $MP_b(N)$ a Boolean formula over the propositions $\Sigma^{out} \uplus \{final\}$ with:

$$\phi(Q) = \bigwedge_{m \in Q} \left(\bigvee_{x \in \Sigma^{out}: m \xrightarrow{x} \text{ in } env(N)} x \quad \bigvee_{m' \in \Omega: m \xrightarrow{\varepsilon} m' \text{ in } env(N)} final \right).$$

For an open net C , a marking m of $inner(C)$ *models* $\phi(Q)$ if $\phi(Q)$ evaluates to true with the following assignment β to the propositions:

- Let $\beta(final)$ be true iff $\exists m' \in \Omega_{inner(C)} : m \xrightarrow{\varepsilon} m'$ in $inner(C)$.
- For other propositions $x \in \Sigma^{out}$, let $\beta(x)$ be true iff $m \xrightarrow{x}$ in $inner(C)$.

Definition 206 [matching with Boolean annotation]

Let N be an open net such that $MP_b(N)$ exists. An open net C *matches* with $MP_b(N)$ and its Boolean annotation ϕ if

1. $I_C = \Sigma^{out}$ and $O_C = \Sigma^{in}$, and
2. $RG(inner(C))$ is weakly simulated by $MP_b(N)$ with relation ϱ such that for all $(m, Q) \in \varrho$:
 - a) If m is not b -bounded in $inner(C)$, then $Q = Q_\emptyset$.
 - b) If m is a stop except for inputs in $inner(C)$, then m models $\phi(Q)$.

Intuitively, the Boolean formula $\phi(Q)$ of a state Q of $MP_b(N)$ represents items (2b) and (2c) of Def. 123: Let $m_Q \in Q$ and let m be a marking of $inner(C)$. Then $\beta(final)$ is false if m is dead except for inputs in $inner(C)$, and there must exist an $x \in O_N = \Sigma^{out}$ such that $m \xrightarrow{x}$ in $inner(C)$ (i.e., $\beta(x)$ is true) and $m_Q \xrightarrow{x}$ in $env(N)$ (i.e., β models $\phi(Q)$) in order to fulfill item (2c) of Def. 123. Fulfilling item (2b) of Def. 123 works accordingly.

Therefore, we directly conclude from Thm. 126.

Corollary 207 [matching with Boolean annotation]

Let N be an open net such that $MP_b(N)$ exists. Then an open net C matches with $MP_b(N)$ and its Boolean annotation ϕ iff C is a b -partner of N .

Computing the Boolean annotation ϕ of a given LTS $MP_b(N)$ is also implemented in the tool Chloe [115].

Summing up, our implementation for discovering a high-quality open net that b -conforms to a given open net $Spec$ takes the following inputs:

- the LTS $MP_b(max_b(Spec))$ with its Boolean annotation ϕ ,
- an event log Log of an open net that b -conforms to $Spec$,
- the weights for the quality dimensions, and
- the parameters and termination criteria for the genetic algorithm.

The output of our implementation is an open net $Impl$ that b -conforms to $Spec$ and has high quality with respect Log and $Spec$.

Example 208 Figure 118 shows the first screen after providing the event log $DLog$ in Tab. 17 and the LTS $MP_1(max_1(D))$ in Fig. 109 with its Boolean annotation as inputs to the implemented ProM plug-in. Here, the user has to provide the parameters of the genetic algorithm and the termination criteria; the predefined standard parameters are an initial population of 100 individuals with 30 elite individuals, 1 crossover point, and a mutation/crossover/replacement probability of 0.1. The standard termination criteria are a maximal runtime of 300 seconds, maximal 1,000 generations with at most 750 generations of stagnation, and a sufficient quality of 0.999. In a following, second screen, the user has to provide weights for the four quality dimensions and may chose to use the abstraction to b -subnets; the standard setting is a weight of 1 for each quality dimension and the abstraction to b -subnets enabled.

Figure 119 shows a screenshot of the running ProM plug-in. As an example, we use the plug-in to discover a high-quality open net that 1-conforms to the open net D in Fig. 108a using $DLog$, $MP_1(max_1(D))$ with its Boolean annotation, and all standard settings. The discovered open net is the open net N_4 from Ex. 204 with 4 inner places, 6 transitions, and a quality of $Q(DLog, N_4, D) \approx 0.9484$. We already showed in Ex. 204 that N_4 has a higher quality than, for example, the open nets D and D' in Fig. 108.◊

9.4 EVALUATION AND EXPERIMENTAL RESULTS

In this section, we evaluate our implementation from Sect. 9.3 with real-life models and artificial event logs. We describe our evaluation process and the necessary preparations in Sect. 9.4.1. In Sect. 9.4.2, we perform the evaluation process and statistically interpret the results.

9.4.1 Preparing the evaluation process

Figure 120 illustrates our evaluation process. To this end, we compute nine open nets as specifications of nine industrial open systems. For each open net $Spec$, we artificially create an event log $Log(Spec)$ that captures observed communication behavior of an open net that 1-conforms to $Spec$. Finally, we discover an open net $Impl$ that 1-conforms to $Spec$ and has high quality with respect to $Log(Spec)$ and $Spec$.

We evaluate our implementation using the running examples D , D' , U , U' , and the five industrial open systems CN , LA , PO , RS , and TR from Sect. 8.4



Figure 118: The first screen of the ProM plug-in asks for the parameters of the genetic algorithm and the termination criteria. Visualized are the standard parameters: An initial population of 100 individuals with 30 elite individuals, 1 crossover point, and a mutation/crossover/replacement probability of 0.1. In addition, the user has to provide the termination criteria; the standard termination criteria are a maximal runtime of 300 seconds, maximal 1,000 generations with at most 750 generations of stagnation, and a sufficient quality of 0.999.

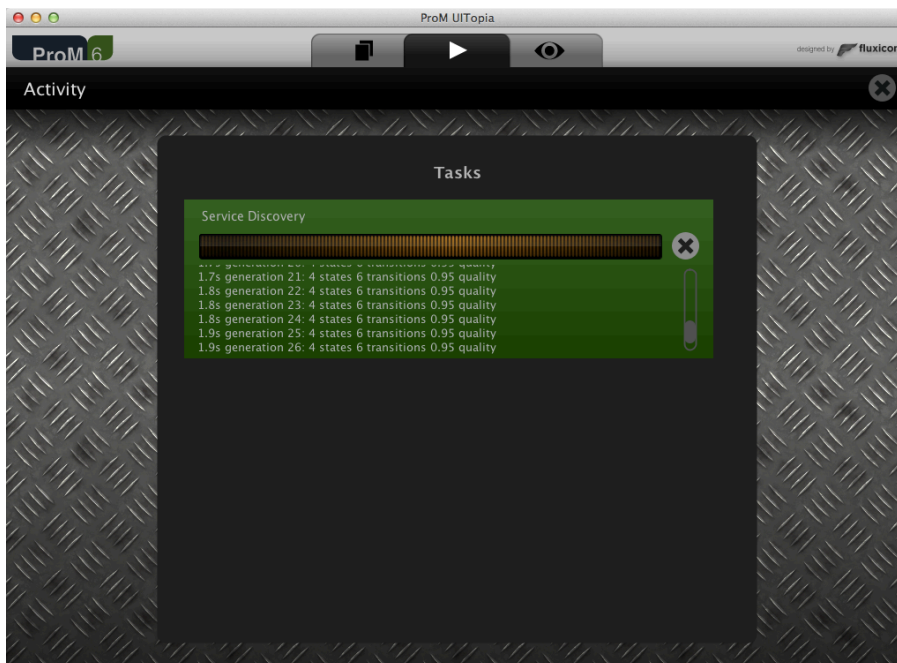


Figure 119: The running ProM plug-in discovers the open net N_4 in Fig. 115a as a high-quality open net that 1-conforms to the open net D in Fig. 108a.

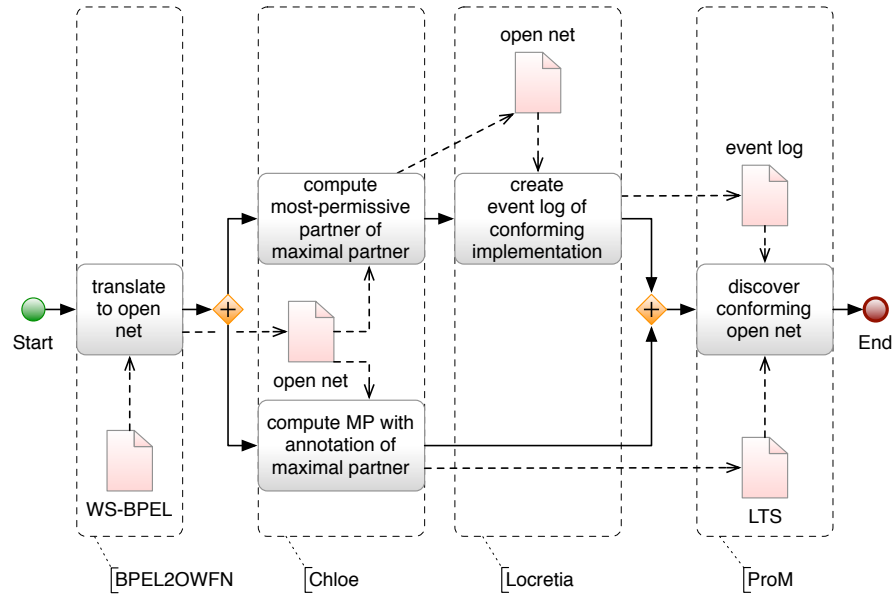


Figure 120: A BPMN diagram that illustrates the evaluation process. The four groups indicate which tool we use for which activity.

on a MacBook Air model A1466 [21]. For an overview over the characteristics of the nine open nets N and the LTSs $MP_1(max_1(N))$, we refer back to Tab. 10 and Tab. 11. For each open net $Spec$ in Tab. 10, we use the reachability graph of $mp_1(max_1(Spec))$ to generate a random event log $Log(Spec)$. That way, we guarantee that there exists at least one open net that 1-conforms to $Spec$ and exhibits the observed behavior in $Log(Spec)$ while simultaneously leaving a maximal degree of freedom in generating $Log(Spec)$, because $mp_1(max_1(Spec))$ is L_1 -maximal by Lem. 130. We generate $Log(Spec)$ with the viewpoint of $Spec$ (i.e., assuming $replay(Spec) = inner(Spec)$ by Def. 173) using the tool Locretia [116]. Each such event log $Log(Spec)$ is noise-free and consists of 400 event traces with about 3,209–3,585 events; see Tab. 18 for the characteristics of the generated event logs. As in Sect. 8.4, the size of our generated event logs coincides with the size of event logs that were successfully applied to evaluate process mining techniques, e.g., in [53].

event log	event traces	events	events per event trace
$Log(D)$	400	3,469	8.67
$Log(D')$	400	3,445	8.61
$Log(U)$	400	3,510	8.77
$Log(U')$	400	3,496	8.74
$Log(CN)$	400	3,585	8.96
$Log(LA)$	400	3,366	8.41
$Log(PO)$	400	3,488	8.72
$Log(RS)$	400	3,252	8.13
$Log(TR)$	400	3,209	8.02

Table 18: The size of the event logs of a 1-conforming implementation that we generated with the tool Locretia.

We use our implementation from Sect. 9.3 to discover an open net $Impl$ that 1-conforms to $Spec$ and has high quality with respect to $Log(Spec)$. As

parameters for the genetic algorithm, we use an initial population of 100 individuals, a mutation/crossover/replacement probability of 0.3 with at most 1 crossover point, and elitism of 0.3, i.e., the 30 individuals with the highest quality are directly shifted to the next generation. The computation of a new generation stops after 1,000 generations, if the highest quality stagnates for 750 generations, if a quality of 0.999 is reached, or if the algorithm ran for 60 minutes. To take into account that a discovered open net can be smaller than the LTS to be compared with (see Sect. 9.1), we chose a weight of 1 for simplicity and a weight of 2 for all other quality dimensions.

To the best of our knowledge, there does not exist any other discovery implementation with which we can compare our algorithm. Therefore, we perform three different experiments on the open nets *Spec* in Tab. 10. In the first experiment, we randomly generate open nets that 1-conform to the given open net *Spec*. Note that because of the restriction to 1,000 generations, our genetic algorithm generates at most 70,100 different individuals: 100 individuals for the initial population, and $100 - 30 = 70$ individuals (because of elitism) for each generation. Thus, for comparability, we randomly generate 71,100 1-conforming open nets in the first experiment, or stop the random generation after 60 minutes (whatever appears first). The generated open net with the highest quality is the experiment result. In the second experiment, we discover open nets that match with $MP_1(\max_1(\textit{Spec}))$ using our genetic algorithm without the abstraction technique. In the third experiment, we discover 1-subnets of *Spec* using our genetic algorithm with the abstraction technique, as explained in Sect. 9.2.

9.4.2 Discovering 1-conforming open nets

In this section, we perform multiple runs of each of the three experiments that we described in Sect. 9.4.1. We statistically interpret the results of these runs to evaluate our discovery algorithm, because a genetic algorithm inherently involves randomness due to mutation, crossover, and replacement operations.

We start by formulating three hypotheses about our discovery algorithm based on one run of each of the three experiments. Table 19 shows the result of one run of Experiment 1. For each open net *Spec* in Tab. 10, Tab. 19 gives the size of the discovered 1-conforming open net (columns 2, 3, and 4), the values of its quality and of the individual quality dimensions (columns 5–9), and the time to discover this open net (last column). The quality of the randomly discovered open nets range from 0.52 to 0.80 with a mean quality of 0.65. The discovery process terminates in all cases because it exceeds the time limit of one hour; in some cases, the discovery process took even significantly longer than one hour. The reason for this is that we check whether a termination criterion (e.g., the runtime of the algorithm) is met *after* each successful generation of a random open net that 1-conforms to *Spec*. Consequently, if we randomly generate a very large open net right before we would reach the time limit, the total runtime of the discovery algorithm may significantly exceed the time limit.

Table 20 shows the results for our discovery algorithm without the abstraction technique from Sect. 9.2—that is, one run of Experiment 2. The quality of the discovered open nets range from 0.55 to 0.80 with a mean quality of 0.71. In all cases except for U' , the quality of the discovered open nets in Experiment 2 is higher than the quality of the discovered open nets in Experiment 1, while simultaneously their size (measured in terms of places,

<i>Spec</i>	discovered <i>Impl</i>			quality					time in s
	$ P $	$ T $	$ F $	Q	<i>fit</i>	<i>sim</i>	<i>pre</i>	<i>gen</i>	
<i>D</i>	17	20	60	0.71	0.40	0.41	1	0.87	3,653.8
<i>D'</i>	90	94	282	0.72	0.51	0.10	1	0.96	3,631.7
<i>U</i>	65	82	246	0.69	0.50	0.22	0.99	0.80	3,603.8
<i>U'</i>	6	9	27	0.80	0.31	1	1	1	3,600.2
<i>CN</i>	526	1,313	3,939	0.52	0.46	0.75	0.82	0.17	3,975.3
<i>LA</i>	21	52	156	0.77	0.46	1	0.98	0.75	3,795.9
<i>PO</i>	414	818	2,454	0.52	0.47	0.50	0.85	0.23	3,928.5
<i>RS</i>	122	208	624	0.63	0.33	0.85	0.96	0.50	3,784.4
<i>TR</i>	457	653	1,959	0.52	0.37	0.19	0.93	0.42	3,822.4
mean				0.65					
sum									33,796.0

Table 19: Experiment 1: Randomly discover an open net *Impl* that 1-conforms to the given open net *Spec*, and measure the quality of *Impl* w.r.t. *Spec* and $\text{Log}(\text{Spec})$.

transitions, and arcs) is smaller (except for the open net that we discovered from the specification *U'*). However, the runtime of Experiment 2 is nearly as high as the runtime of Experiment 1. This can be explained by the general procedure of genetic algorithms [89]: Genetic algorithms like ours evaluate all given termination criteria (e.g., the runtime of the algorithm) after creating a new generation out of the old generation; creating a new generation out of the old one may take a significant amount of time. In our case, creating a new generation also involves a replacement operation that replaces an individual with low quality by a randomly created individual. The replacement operation may be invoked several times for each generation, and each invocation may take—similar to the random creation in Experiment 1—a significant amount of time.

<i>Spec</i>	discovered <i>Impl</i>			quality					time in s
	$ P $	$ T $	$ F $	Q	<i>fit</i>	<i>sim</i>	<i>pre</i>	<i>gen</i>	
<i>D</i>	4	6	18	0.79	0.34	1	1	0.94	3,668.1
<i>D'</i>	9	11	33	0.80	0.42	0.80	1	1	3,682.6
<i>U</i>	9	20	60	0.73	0.45	0.97	0.99	0.66	3,746.7
<i>U'</i>	9	10	30	0.80	0.38	0.89	1	1	3,603.2
<i>CN</i>	173	554	1,662	0.55	0.37	0.99	0.85	0.21	3,817.4
<i>LA</i>	23	45	135	0.80	0.43	0.91	1	0.92	3,668.9
<i>PO</i>	102	293	879	0.59	0.46	1	0.87	0.22	4,002.0
<i>RS</i>	18	20	60	0.71	0.21	0.84	1	0.84	3,621.6
<i>TR</i>	68	176	528	0.62	0.40	0.97	0.92	0.38	3,800.1
mean				0.71					
sum									33,610.6

Table 20: Experiment 2: Discover an open net *Impl* that 1-conforms to the given open net *Spec* using the genetic algorithm, and measure the quality of *Impl* w.r.t. *Spec* and $\text{Log}(\text{Spec})$.

Table 21 shows the results for our discovery algorithm with the abstraction technique from Sect. 9.2—that is, one run of Experiment 3. The results

in Tab. 20 show that discovered open nets in Experiment 2 are more complex than the ones in Experiment 3 because 1-subnets are obviously smaller than arbitrary open nets that 1-conform to the given open net. This explains the higher computation time in Experiment 2 by a factor of 1–11 compared to Experiment 3: Smaller candidates enable the genetic algorithm to compute more generations in less time. For the same reason, Experiment 3 produced, in general, open nets with higher fitness. The simplicity values are by Def. 185 higher for Experiment 3. In all examples, the discovered open nets in Experiment 3 have slightly higher precision values than the discovered open nets in Experiment 2. Only one out of nine examples has a slightly lower generalization value. Restricting the search space to 1-subnets is an abstraction, which neither preserves precision nor generalization by Cor. 203. Therefore, we expected lower precision and generalization values for the open nets discovered in Experiment 3, although our experiment shows that the precision and generalization values actually are higher. Despite the (theoretical) loss of preservation of the abstraction, the overall quality of the respective open net discovered in Experiment 3 is in all examples better. The quality of the discovered open nets range from 0.59 to 0.84 with a mean quality of 0.77, compared to a mean quality of 0.65 in Experiment 1 and a mean quality of 0.71 in Experiment 2.

<i>Spec</i>	discovered <i>Impl</i>			quality					time in s
	$ P $	$ T $	$ F $	Q	<i>fit</i>	<i>sim</i>	<i>pre</i>	<i>gen</i>	
<i>D</i>	4	5	15	0.81	0.39	1	1	0.94	337.0
<i>D'</i>	5	7	21	0.81	0.45	1	1	0.89	394.9
<i>U</i>	9	15	45	0.80	0.47	1	0.99	0.83	831.2
<i>U'</i>	7	10	30	0.82	0.40	1	1	0.98	803.7
<i>CN</i>	68	189	567	0.59	0.28	1	0.90	0.39	3,641.8
<i>LA</i>	19	48	144	0.84	0.46	1	1	0.99	2,573.9
<i>PO</i>	56	122	366	0.74	0.40	1	0.96	0.72	3,624.0
<i>RS</i>	12	13	39	0.79	0.27	1	1	0.99	3,601.3
<i>TR</i>	35	68	204	0.77	0.38	1	0.98	0.82	3,601.1
mean				0.77					
sum									19,408.9

Table 21: Experiment 3: Discover an open net *Impl* that 1-conforms to the given open net *Spec* using the genetic algorithm with the abstraction technique to 1-subnets, and measure the quality of *Impl* w.r.t. *Spec* and $\text{Log}(\text{Spec})$.

Based on Tab. 19, Tab. 20, and Tab. 21, we formulate three scientific hypotheses:

1. Our approach of using a search heuristic in form of a genetic algorithm is better than guessing: The quality of an open net discovered by our discovery algorithm without the abstraction technique is better than the quality of a randomly discovered open net.
2. Using the abstraction technique from Sect. 9.2, we can discover open nets with a better quality than without the abstraction technique.
3. Using the abstraction technique from Sect. 9.2, we can discover high-quality open nets in less time than without the abstraction technique.

We show these three hypotheses and, thus, evaluate our discovery algorithm using two-sample unpooled t-tests [113]. Thereby, a sample is a set of

runs of an experiment (called observations) that we described in Sect. 9.4.1. The two-sample unpooled t-test (using Welch’s test statistic) is used to determine if two sample means are equal; a common application is to test if a new procedure is superior to a current procedure [113]. We can apply a two-sample unpooled t-test if the following three assumptions are met: (1) the two samples derive from normal populations or their combined sample size is greater than 40, (2) the observations in each sample are independent, and (3) the standard deviations of the two samples are unequal or unknown. In our setting, the second assumption holds because two runs of an experiment do not influence each other. The third assumption holds, because we have no information about the standard deviations of the discovered open nets’ quality or the time needed to discover an open net. Because of the first assumption, we use samples that consist of 30 runs of an experiment each; as a two-sample unpooled t-test compares two samples (that is, two experiments), the combined sample size is $30 + 30 = 60$ for each t-test, which is greater than 40.

As specifications for the experiments, we only use the five industrial open systems *CN*, *LA*, *PO*, *RS*, and *TR* from Sect. 8.4. The running examples *D*, *D'*, *U*, and *U'* are all smaller than the industrial open systems and, therefore, we do not expect significant differences between their runs. With three experiments, five specifications, and 30 runs per sample, we perform $3 \cdot 5 \cdot 30 = 450$ runs in total.

Figure 121 to Fig. 125 show the quality of the discovered 1-conforming open net (on the left-hand side) and the time needed to discover that open net (on the right-hand side), for each of the Experiments 1 to 3, for each of the 30 runs, and for each of the specifications *CN*, *LA*, *PO*, *RS*, and *TR*. In the majority of runs, the quality of the discovered open net in Experiment 2 is better than the quality of the discovered open net in Experiment 1: that is the case in 28 runs for *CN*, 27 runs for *LA*, 25 runs for *PO*, 30 runs for *RS*, and 29 runs for *TR*, which sums up to 139 out of 150 runs. In nearly all runs, the quality of the discovered open net in Experiment 3 is better than the quality of the discovered open net in Experiment 2: that is the case in 28 runs for *CN*, and all 30 runs for *LA*, *PO*, *RS*, and *TR*. Simultaneously, discovering an open net in Experiment 3 takes less time than in Experiment 2: that is the case in 25 runs for *CN*, and all 30 runs for *LA*, *PO*, *RS*, and *TR*.

We proceed by statistically testing our three hypotheses using two-sample unpooled t-tests. Table 22 shows the results of five tests comparing the quality of the discovered 1-conforming open net in Experiment 1 (the first sample) with the quality of the discovered 1-conforming open net in Experiment 2 (the second sample), for each of the specifications *CN*, *LA*, *PO*, *RS*, and *TR*. Using Welch’s test statistic (column 6) and the computed degrees of freedom (column 7) for the two samples, we compute a p-value (column 8) for each t-test [113]. The p-value is the probability of obtaining a test statistic at least as extreme as the one that was actually observed, assuming that the null hypothesis (that is, the means of the two samples are equal) is true. In all five tests, we can reject the null hypothesis even under the revised standards for statistical evidence [128], which mandate the conduct of tests at the 0.005 (“significant”) or 0.001 (“very significant”) level of significance. In other words, as the p-value is smaller than 0.001 in each t-test, we conclude that the quality of an open net discovered by our genetic discovery algorithm is very significantly better than the quality of a randomly discovered open net. This confirms our first hypothesis.

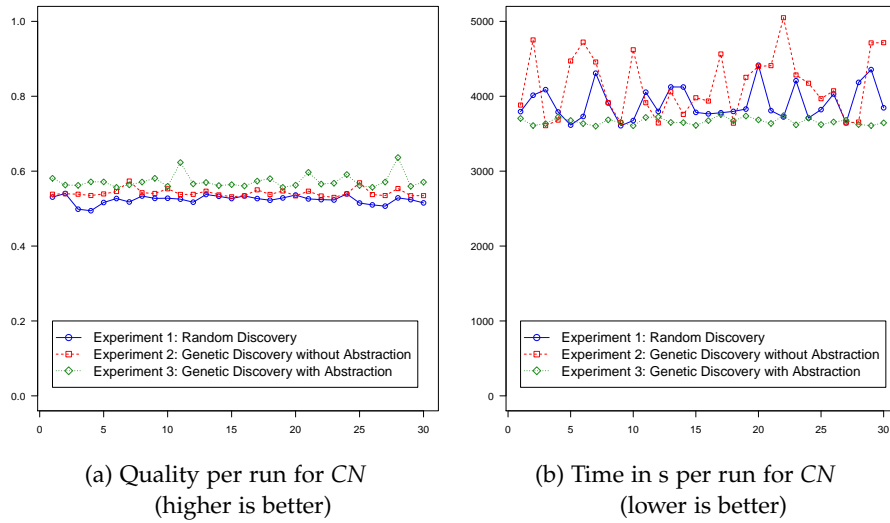


Figure 121: The quality of a discovered open net that 1-conforms to CN , and the time needed to discover it. We repeated the Experiments 1 to 3 thirty times.

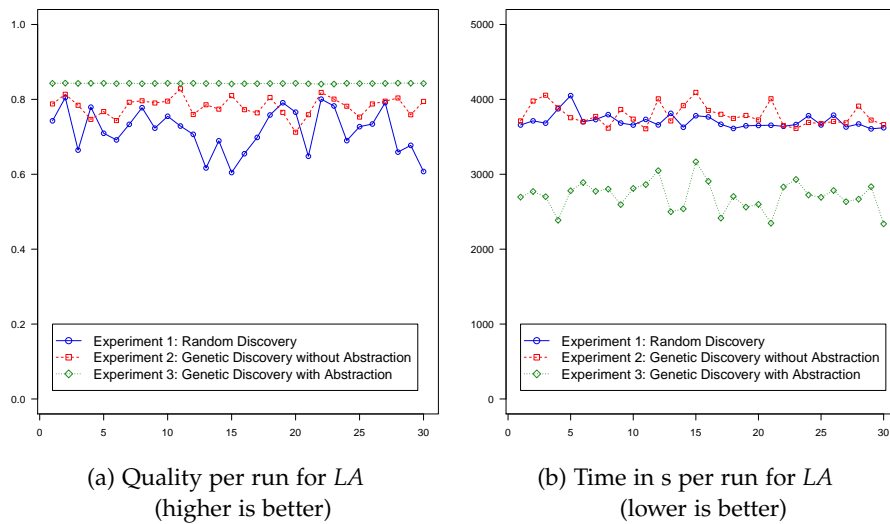


Figure 122: The quality of a discovered open net that 1-conforms to LA , and the time needed to discover it. We repeated the Experiments 1 to 3 thirty times.

Table 23 shows the results of five tests comparing the quality of the discovered open net in Experiment 2 (the first sample) with the quality of the discovered open net in Experiment 3 (the second sample), for each of the specifications CN , LA , PO , RS , and TR . As in Tab. 22, the p-value is smaller than 0.001 in each t-test. Thus, we reject the null hypothesis and conclude that the quality of an open net discovered by our genetic discovery algorithm with the abstraction technique from Sect. 9.2 is very significantly better than the quality of an open net that we discovered without that abstraction technique. This confirms our second hypothesis.

Finally, Tab. 24 shows the results of five tests comparing the time needed to discover a 1-conforming open net in Experiment 2 (the first sample) with the time needed to discover a 1-conforming open net in Experiment 3 (the second sample), for each of the specifications CN , LA , PO , RS , and TR . Again, the p-value is smaller than 0.001 in each t-test; therefore, we reject the null hypothesis and conclude that our genetic algorithm with the

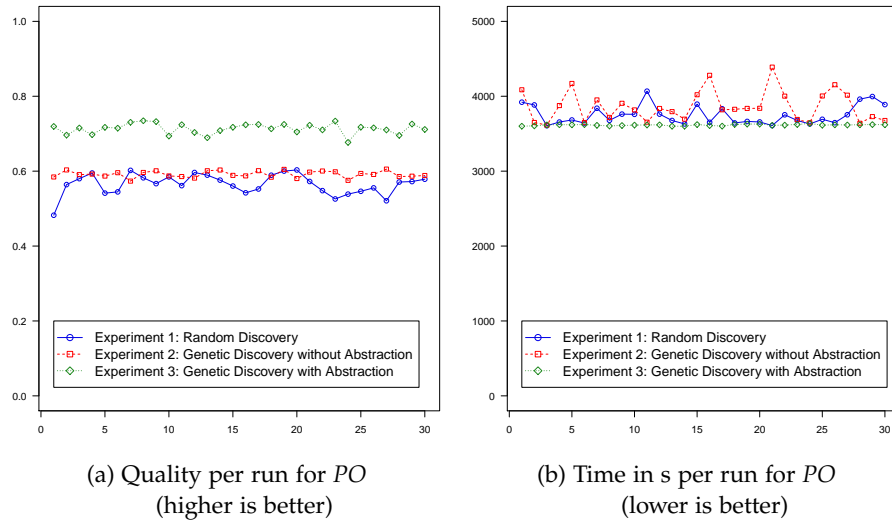


Figure 123: The quality of a discovered open net that 1-conforms to PO , and the time needed to discover it. We repeated the Experiments 1 to 3 thirty times.

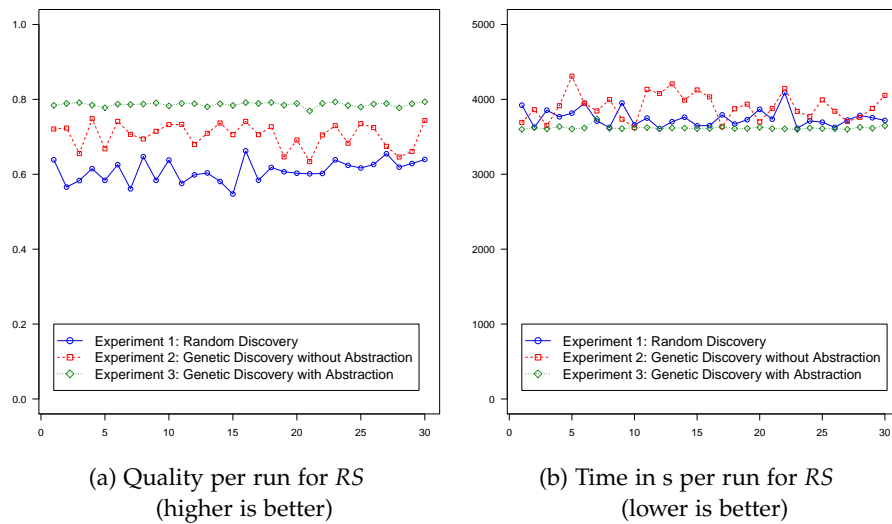


Figure 124: The quality of a discovered open net that 1-conforms to RS , and the time needed to discover it. We repeated the Experiments 1 to 3 thirty times.

abstraction technique from Sect. 9.2 is very significantly faster than our genetic discovery algorithm without that abstraction technique. This confirms our third and last hypothesis.

Summing up, our experimental results validate that, in general, the genetic discovery algorithm produces significantly better results on a finite abstraction of the search space than on the complete search space, while taking significantly less time. Although the abstraction technique only preserves fitness and simplicity, the values of the precision and the generalization dimensions as well as the quality are high and, in general, higher than without that abstraction technique.

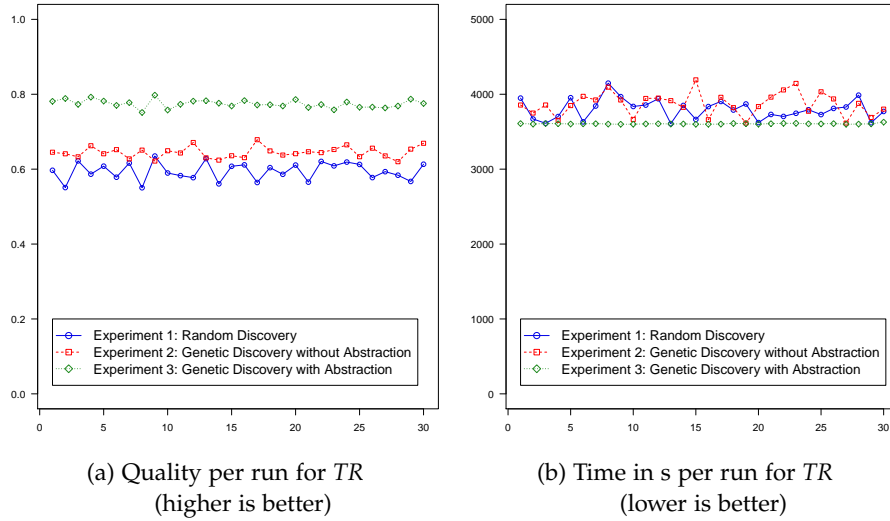


Figure 125: The quality of a discovered open net that 1-conforms to TR , and the time needed to discover it. We repeated the Experiments 1 to 3 thirty times.

<i>Spec</i>	Experiment 1		Experiment 2		t	d.f.	p-value
	size	mean	size	mean			
<i>CN</i>	30	0.5237	30	0.5417	-6.5842	57.395	$7.604 \cdot 10^{-9}$
<i>LA</i>	30	0.7171	30	0.7815	-5.5609	39.595	$1.008 \cdot 10^{-6}$
<i>PO</i>	30	0.5647	30	0.5917	-5.1132	34.805	$5.775 \cdot 10^{-6}$
<i>RS</i>	30	0.6090	30	0.7039	-11.8367	56.951	$< 2.2 \cdot 10^{-16}$
<i>TR</i>	30	0.5943	30	0.6446	-9.9478	49.028	$1.19 \cdot 10^{-13}$

Table 22: Comparing the quality of the discovered 1-conforming open nets in the Experiments 1 and 2 using two-sample unpooled t-tests for equal means with null hypothesis “equal means” and alternative hypothesis “true difference in means is less than 0” (i.e., the open nets discovered in Experiment 2 are of better quality than the open nets discovered in Experiment 1, which corresponds to our first hypothesis).

<i>Spec</i>	Experiment 2		Experiment 3		t	d.f.	p-value
	size	mean	size	mean			
<i>CN</i>	30	0.5417	30	0.5726	-8.0556	44.872	$1.46 \cdot 10^{-10}$
<i>LA</i>	30	0.7815	30	0.8425	-13.2035	29.042	$4.193 \cdot 10^{-14}$
<i>PO</i>	30	0.5917	30	0.7135	-40.3245	48.544	$< 2.2 \cdot 10^{-16}$
<i>RS</i>	30	0.7039	30	0.7863	-13.4608	30.508	$1.131 \cdot 10^{-14}$
<i>TR</i>	30	0.6446	30	0.7746	-39.0412	52.514	$< 2.2 \cdot 10^{-16}$

Table 23: Comparing the quality of the discovered 1-conforming open nets in the Experiments 2 and 3 using two-sample unpooled t-tests for equal means with null hypothesis “equal means” and alternative hypothesis “true difference in means is less than 0” (i.e., the open nets discovered in Experiment 3 are of better quality than the open nets discovered in Experiment 2, which corresponds to our second hypothesis).

9.5 CONCLUSIONS

In this chapter, we presented a technique to discover a system model of an open system *Impl* from a given system model *Spec* and observed behavior

<i>Spec</i>	Experiment 2		Experiment 3		t	d.f.	p-value
	size	mean	size	mean			
CN	30	4,153	30	3,665	6.3216	29.678	$2.987 \cdot 10^{-7}$
LA	30	3,788	30	2,710	24.5849	51.849	$< 2.2 \cdot 10^{-16}$
PO	30	3,866	30	3,615	6.6908	29.114	$1.196 \cdot 10^{-7}$
RS	30	3,905	30	3,620	8.6104	30.127	$6.375 \cdot 10^{-10}$
TR	30	3,871	30	3,605	9.5494	29.077	$9.051 \cdot 10^{-11}$

Table 24: Comparing the time (in seconds) needed to discover a 1-conforming open net in the Experiments 2 and 3 using two-sample unpooled t-tests for equal means with null hypothesis “equal means” and alternative hypothesis “true difference in means is greater than 0” (i.e., it takes less time to discover an open net in Experiment 3 than in Experiment 2, which corresponds to our third hypothesis).

of *Impl* interacting with its environment. Our technique produces a system model for *Impl* that b -conforms to *Spec* and, in addition, balances the four conflicting quality dimensions (i.e., fitness, simplicity, precision, and generalization). As an additional improvement, we proposed an abstraction technique to reduce the infinite search space to a finite one. As an exhaustive search to find an optimal solution may still be intractable, we implemented our technique as a genetic algorithm. In a prototypical implementation, we experimented with several system models of industrial size. Our results showed that the algorithm finds (nearly) optimal solutions in acceptable time on a computer with average computing power.

It is worth mentioning that although we evaluated our approach using service models, the approach is not restricted to service models but can discover arbitrary open systems. In addition, we can also apply our approach to discover a b -partner C of an open net N such that C has, among the set of all b -partners of N , high or even the highest quality with respect to a given event \log *Log*: The set of all b -partners of N is equally well represented by the LTS $MP_b(N)$ as the set of all b -conforming open nets is represented by the LTS $MP_b(\max_b(N))$. Thus, we can also use the technique presented in this chapter to discover a b -partner.

CONCLUSIONS AND RELATED WORK

IN this chapter, we summarize the results from Part III. In addition, we review work that is related to conformance testing in Sect. 10.2 and work that is related to open system discovery in Sect. 10.3.

10.1 OVERVIEW OF THE RESULTS

We studied a conformance relation between two open systems in the log-model scenario. In the log-model scenario, the formal model of only one open system—the specification—is given, but no formal model of the second open system—the implementation—is available. Instead of a formal model of the implementation, we use the observed behavior of the implementation as input. We referred to the latter as the event log. Figure 126 illustrates again our assumptions for the log-model scenario.

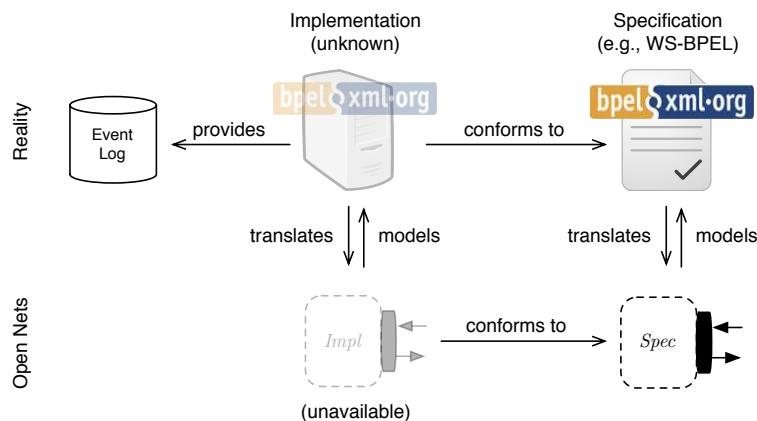


Figure 126: The log-model scenario

In the log-model scenario, we followed two goals. Our first goal was to investigate how to use the event log to check b -conformance of the unknown implementation to the given specification. To this end, we proposed a testing approach based on a necessary condition for b -conformance in Chap. 8. We analyzed whether there exists a b -conforming implementation which may produce the behavior seen in the event log without any mismatches. Thereby, we used the finite characterization of all b -conforming open nets, in the form of a maximal b -partner, that we developed in Part II. We demonstrated our testing approach using industrial-sized specifications and event logs, and the tools from Part II.

Our second goal was to support the design of b -responsive open systems in the log-model scenario by discovering a formal model of the unknown implementation based on the given event log. To this end, we presented a discovery technique in Chap. 9. We produced a system model for $Impl$ that b -conforms to $Spec$ and, in addition, balances the four conflicting quality dimensions fitness, simplicity, precision, and generalization. As an additional improvement, we proposed an abstraction technique to reduce the infinite search space to a finite one, and evaluated the discovery algorithm with and

without the abstraction technique using industrial-sized specifications and event logs.

10.2 WORK RELATED TO CONFORMANCE TESTING

In this section, we review work related to our conformance testing approach in Chap. 8.

Our conformance testing approach assumes recorded behavior (i.e., an event log) of the implementation to be given, and employs this recorded behavior to test for conformance of the unknown implementation to the known specification. Here, techniques are adapted from process mining [2]. Process mining techniques focus on extracting process models from event logs (“process discovery”), comparing normative models with the reality recorded in event logs (which is also called “conformance testing” [218] or “conformance checking” [12, 9, 219, 15, 5]), and extending models based on event logs (“extension”). In the following, we give a brief overview of conformance checking in process mining.

The goal of conformance checking in process mining is to find commonalities and discrepancies between the modeled behavior and the observed behavior of a process [2]. Cook and Wolf [71] compare event traces with process models to measure their similarity. The similarity of an event trace w and a process model N is quantified by the number of insertions and deletions that are necessary to transform w into a trace of N ; this is, in essence, the idea behind the alignments [15, 5] that we used in Chap. 8 and Chap. 9. Later, Cook et al. [70] extended their approach to also consider time aspects. Rozinat et al. [218, 219] propose a token-based replay approach to measure the fitness of an event trace w and a labeled net N : w is replayed on N by adding necessary tokens (i.e., missing tokens in the preset of a transition) and removing superfluous tokens (i.e., remaining tokens in the postset of a transition). A fitness metric is then calculated based on the number of added, superfluous, produced, and consumed tokens. In contrast to alignments, the fitness metric in [219] is sensitive to the structure of N . Goedertier et al. [106] augment an event trace w with artificial negative events before comparing w to a labeled net N in a way similar to [219]; negative events are then used to quantify the precision of w and N . Adriansyah et al. [15, 5] compute alignments between an event trace and a labeled net using the A^* algorithm and sophisticated heuristics. Based on these alignments, they also introduce the precision measure [16] that we employ in Chap. 9.

All approaches in [71, 70, 219, 106, 15, 5] focus on a closed system by relating observed process behavior (i.e., executed activities) to a process model. In contrast, we consider the interaction between (multiple) open systems, and relate observed interaction behavior (i.e., sent or received messages, possibly from two different viewpoints) to an open system model. In addition, replaying an event trace w on a labeled N without any mismatch but without reaching a final marking of N is considered erroneous in [219, 15, 5]; in other words, [219, 15, 5] implicitly assume w to reach a final marking of N . In contrast, we do not make any assumptions about w and N . Therefore, existing replay techniques require event logs of higher quality [6] (e.g., event logs with complete event traces), whereas our approach also works with event logs of lower quality (e.g., event logs with incomplete event traces).

In our setting, Van der Aalst et al. [9] map a service contract specified in WS-BPEL [130] onto a workflow net [1] (which, in that case, can be seen as the inner net resembling the replay environment) and employ confor-

mance checking techniques from process mining [219] on this workflow net. In contrast, we can measure the deviation of an implementation from its specification with respect to all possible b -conforming implementations; if there exists a deviation, then the implementation does not b -conform to the specification. In addition, the approach in [9] does not allow for a finite characterization of all implementations—in contrast to the maximal b -partner in Chap. 5.

Comuzzi et al. [69] investigate online conformance checking (that is, conformance checking with incomplete event traces) using a weaker refinement notion than our notion of b -conformance. Different conformance relations on a concurrency-enabled model have been studied by Ponce de León et al. [148]. As their considered conformance relations differ from b -conformance, their work is not applicable in our setting. Also, maximal partners have not been studied yet in the setting of [148].

Motahari-Nezhad et al. [186] investigate event correlation; that is, they try to find relationships between events that belong to the same process execution instance. In contrast to event correlation, we do not vary the system instances, but consider a conformance relation of an unknown implementation to the known specification.

Our notion of conformance testing is also called monitoring [219] or passive testing [239]: We solely rely on the given specification and the observed behavior recorded by the event log, and have no control over the test case (i.e., the open system that communicates with the unknown implementation and from whose communication the provided event log originates). Our passive testing approach is opposed to active testing, where a tester has active control over the test environment and especially a set of predefined tests that are executed [239, 49]. For example, Kaschner [132] constructs test cases from the operating guideline that we described in Sect. 7.3 to actively test for conformance of asynchronously communicating services.

Brinksma and Tretmans [49] present an annotated bibliography of test theory that is based on labeled transition systems. Another approach for formal testing is based on Mealy finite state machines [170] (also known as the FSM-approach); for overviews of the FSM-approach see [144, 207]. The link between the FSM-approach and test theory based on labeled transition systems is studied by Tan [231].

Note that our testing approach is not restricted to b -conformance; in general, we can test for every conformance relation that allows to compute a finite maximal partner.

10.3 WORK RELATED TO OPEN SYSTEM DISCOVERY

In this section, we review work related to our discovery approach in Chap. 9.

Discovering a formal model from observed behavior recorded in event logs is studied in the area of process mining [2], as already explained in Sect. 10.2. There exists a variety of discovery algorithms; for example, the α -algorithm [13], the ILP-miner [254], the heuristics miner [253], and genetic discovery algorithms [171, 52]. These discovery algorithms are all tailored toward closed systems. They discover a formal process model from an event log that recorded process activities. In contrast, the presented discovery algorithm in Chap. 9 operates in the setting of open systems. We discover a formal open system model from an event log that recorded communication behavior between two running open systems.

In the area of service-oriented computing [201], the term “discovery” is ambiguous: On the one hand, discovery describes techniques for producing a service model from observed communication behavior of services [6], and on the other hand, discovery describes techniques for finding a service model in a service repository in service-oriented architectures [201]. Process mining research has been focused on processes but during the last few years, process mining techniques have also been applied to services resulting in the term “service mining”. Van der Aalst [3] reviews service mining research and identifies two main challenges regarding the discovery of services: (1) the correlation of instances of a service with instances of another service (e.g., [32, 186]), and (2) the discovery of services based on observed behavior (e.g., [86, 23, 232, 195, 185]). A service can be seen as an open system, thus Chap. 9 contributes to the second challenge.

Dustdar and Gombotz [86] discover workflow models from service interaction. The authors of [23, 232] discover workflow models from interaction patterns. However, these approaches can only discover parts of a (complex) service in the form of service composition pattern, whereas our discovery algorithm produces a complete (service) model.

Musaraj et al. [195] correlate messages from an event log without correlation information and use this information in their discovery algorithm. In contrast, we abstract from correlation information and assume cases to be independent. Another difference is that our discovered model b -conforms to a given open system model $Spec$ and it balances the four conflicting quality dimensions with respect to a given event log, guided by user preferences.

Motahari-Nezhad et al. [185] present a user-driven refinement approach for discovering service models. In essence, their approach considers the fitness and the precision dimension, but ignores generalization and simplicity of the discovered service model. Like Musaraj et al. [195], Motahari-Nezhad et al. [185] do not assume a service model to be given and, thus, they cannot guarantee that their produced service model can interact correctly (i.e., b -responsively) with its environment.

The idea of using a genetic algorithm for discovery is inspired by the work of de Medeiros et al. [171]. Buijs et al. [52, 53] use a genetic algorithm to discover sound workflow models while balancing the four conflicting quality dimensions. In Sect. 9.1, we discussed the relation of our measures for these four quality dimensions and the measures used in [52, 53]. For the simplicity measure, we used the structure of the LTS $CSD_b(max_b)$, which does not exist for workflow models. Correctness in our setting is b -responsiveness, which is a weaker criterion than soundness in [53]; soundness additionally requires proper termination. To deal with b -responsiveness in the setting of open systems, we assume an open system $Spec$ to be given and we discover, from observed behavior of $Spec$ and its environment, an open system $Impl$ that is guaranteed to b -conform to $Spec$.

Part IV

CLOSURE

IN the previous two parts, we developed algorithms for verifying responsiveness for open systems by means of conformance checking in two distinct scenarios: the model-model scenario and the log-model scenario. In the model-model scenario (Part II), we decided whether an open system *Impl* *b*-conforms to an open system *Spec* based on a formal model of *Impl* and a formal model of *Spec*. In the log-model scenario (Part III), we tested whether *Impl* *b*-conforms to *Spec* based on an event log of *Impl* and a formal model of *Spec*. In addition, we discovered a formal model of *Impl*—under the assumption that *Impl* *b*-conforms to *Spec*—from an event log of *Impl* and a formal model of *Spec*. So far, we presented the theory and evaluated the developed algorithms using industrial-sized formal models and event logs. In this chapter, we present a practical use case and apply the previously developed approaches. We specify—both informally and formally—the emergency ward of a hospital as an open system. Thereby, the emergency ward is part of a stroke unit for treating stroke patients [199]. Our specification is inspired by a BPMN [63] model of the stroke treatment process that we modeled at the Charité Berlin [85], which is one of the largest university hospitals in Europe. We implement two variants of the emergency ward service in the industrial language WS-BPEL [130]; one implementation that 1-conforms to the specification, and one implementation that does not. We then demonstrate the developed approaches using these two implementations and the specification. For the model-model scenario, we automatically translate the WS-BPEL models into open nets and check for 1-conformance; this demonstrates the applicability of the approach in Part II. For the log-model scenario, we deploy the WS-BPEL models using a WS-BPEL engine and derive event logs from example executions. We then test for 1-conformance with the derived event logs and the specification, and discover a formal model of the 1-conforming implementation; this demonstrates the applicability of the approaches in Part III.

We specify the emergency ward service and its two implementations in Sect. 11.1.1. In Sect. 11.2 and Sect. 11.3, we demonstrate the applicability of the techniques and analysis tools for the model-model scenario and the log-model scenario, respectively. Section 11.4 concludes this chapter with a discussion.

11.1 THE EMERGENCY WARD SERVICE IN A STROKE UNIT

In this section, we present the emergency ward service as the running example of this chapter. We informally describe the specification of the emergency ward service in Sect. 11.1.1 and subsequently formalize (parts of) the specification as an open net in Sect. 11.1.2. In Sect. 11.1.3, we present two implementations in WS-BPEL.

11.1.1 An informal specification

A *stroke* is the loss of brain function due to disturbance in the blood supply to the brain [140]. A stroke can be classified either as ischemic or as

hemorrhagic: An ischemic stroke is the most frequently occurring kind of stroke; it is characterized by the interruption of the blood supply due to a clot blocking or narrowing one of the blood vessels that supply blood to the brain. The hemorrhagic stroke is characterized by the accumulation of blood anywhere within the skull vault due to the rupture of a blood vessel or an abnormal vascular structure. A stroke can lead to severe neurological deficits or death; cerebrovascular diseases associated with strokes were the second leading cause of death worldwide in 2004 [168].

An ischemic stroke is typically treated with thrombolysis therapy, which breaks down the clot and normalizes the blood flow to the brain [109]. Whether thrombolysis therapy can be applied depends on various factors: First, thrombolysis therapy is only permitted within the first three hours after the stroke symptoms started. Second, thrombolysis therapy cannot be applied to hemorrhagic strokes, as this would increase the accumulation of blood. A cerebral hemorrhage can be excluded via a CT scan. Last, thrombolysis therapy cannot be applied if the stroke patient is on blood thinning medication, which can be checked by analyzing the patient's blood in a laboratory. In general, the faster the thrombolysis therapy is started, the less damage is caused to the brain ("time is brain") [109].

To assure a time-efficient care of stroke patients, many hospitals operate a *stroke unit* [199]. A stroke unit is a special ward for stroke patients where nursing staff and doctors from different specializations and hospital units cooperate to stabilize and normalize the physiological functions and to initiate therapy. Five hospital units are involved in the stroke unit at the Charité Berlin: The emergency ward, the radiology unit, the neurology unit, the transport unit, and the hospital laboratory. In the following, we solely focus on the emergency ward; a more detailed description of the stroke unit at the Charité Berlin can be found in [85].

Figure 127 shows the BPMN [63] model of the emergency ward that we recorded by observing several stroke patients and by interviewing the involved staff and doctors [85]. The numbered message flows model the interaction with the other hospital units that are involved in the stroke unit.

If a stroke patient arrives at the emergency ward (start event "patient admission"), she is immediately examined by the nursing staff (activity "examination by nursing staff"). If the stroke patient exhibits stroke symptoms (i.e., is apoplectiform), the nursing staff triggers the stroke alarm [199] (activity "trigger stroke alarm"); otherwise, the patient is transferred to a different hospital unit or dismissed. The stroke alarm alerts the neurology unit, the transport unit, and the radiology unit (message flows 1, 2, and 5). Then, the emergency ward process concurrently waits for a transport service, a neurologist, and a phone call from the radiology unit to arrive (message flows 3, 4, and 6). The phone call from the radiology queries the patient infos (intermediate event "patient info queried"); the nursing staff reports the patient infos (activity "report patient info") and receives the number of a free CT (intermediate event "CT# received"). Concurrently, an internist and the nursing staff continue to examine the stroke patient (activity "examination by internist"), take a blood sample for the laboratory unit (activity "send blood sample to lab" and message flow 7), and compile the examination results into an internal medicine report (activity "create internal medicine report"). When a neurologist arrives, she performs a neurological examination of the patient to confirm the diagnosis and to determine the nature of the stroke (activity "examination by neurologist"); the results are compiled into a neurological report (activity "create neurological report"). Based on

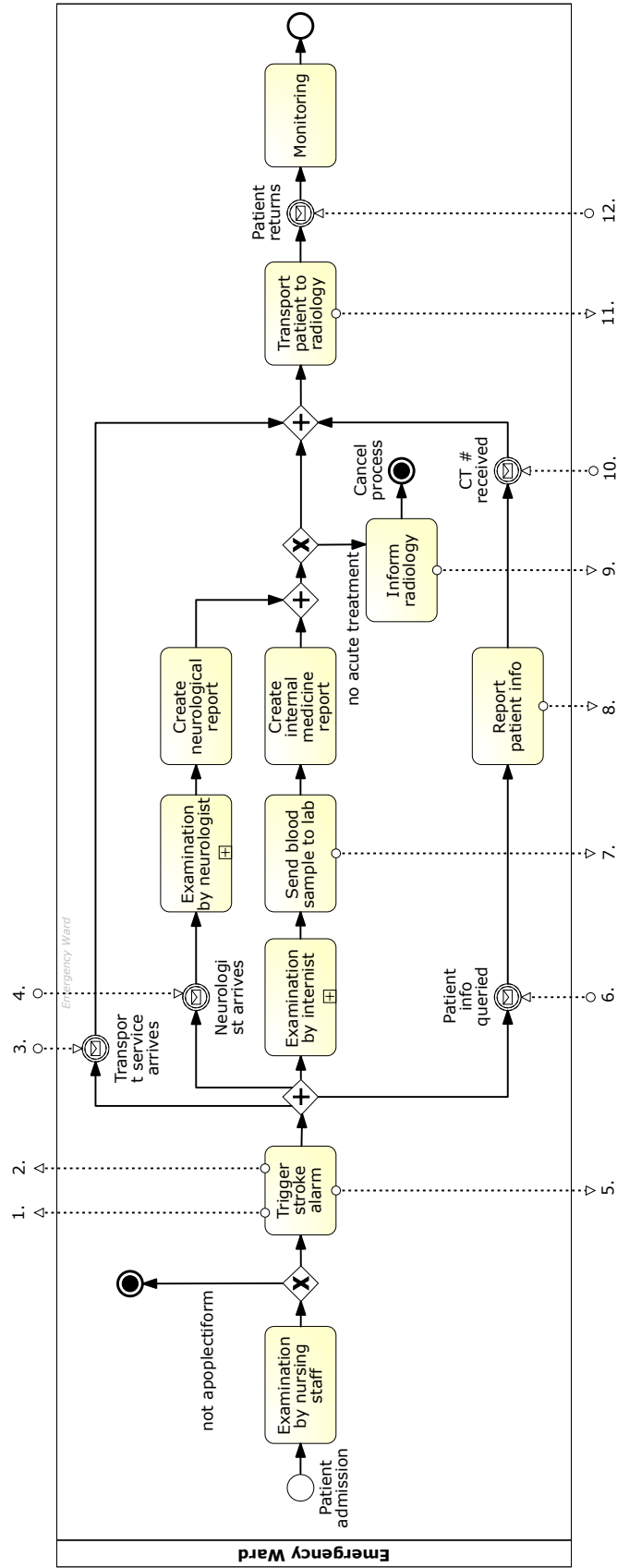


Figure 127: The emergency ward in the Charité stroke unit (taken from [85]).

the internal medicine report and the neurological report, the internist and the neurologist decide whether the patient can receive acute treatment or not. If the patient should not receive acute treatment, the nursing staff informs the radiology unit and cancels the (whole stroke unit) process. If the patient should receive acute treatment, the process proceeds as soon as the transport service (upper branch) and the CT number (lower branch) have arrived. Then, the transport service takes the stroke patient to the CT machine (activity “transport patient to radiology”), along with the neurologist, the thrombolytic drugs and all relevant medical files (message flow 11). At the radiology unit, a CT scan of the patient’s brain reveals whether there exists a cerebral hemorrhage. If all prerequisites are met (i.e., the stroke is not hemorrhagic, the patient is not on blood thinning medication, and the stroke symptoms appeared at most three hours ago), the neurologist begins the thrombolysis therapy while still at the radiology unit. The ischemic stroke patient is then transported back to the emergency ward along with the neurologist (intermediate event “patient returns” and message flow 12). Finally, the stroke patient is monitored (activity “monitoring”) until she can be moved to another hospital unit (end event).

11.1.2 A formal model of the specification

We formalize (parts of) the informal specification of the emergency ward from the previous section: Figure 128 depicts the resulting open net *ew*. We abstract from the communication with the transport unit (the outer left branch in Fig. 127) and the CT unit requesting patient information (the outer right branch in Fig. 127): The control flow of the open net *ew* already branches internally into two branches after the transition “send alarm”; a third and a fourth branch would only result in an increased size of *ew* without providing additional insights into the approaches that we are going to demonstrate on *ew* (and its implementations) in this chapter. In addition, we explicitly specify the continuation of the patient’s treatment: Regardless whether the stroke patient receives acute treatment via the thrombolysis therapy or not, we compile information about the further therapy and send them to the environment (i.e., other hospital units) via the transition “send further_therapy” and the output place “further_therapy”.

There exist tools like the “BPMN to Petri net transformer” [79] to automatically derive Petri net models from BPMN models like the one in Fig. 127. However, here we do not use any tool but derive the specification *ew* manually to incorporate the above mentioned design decisions—that is, abstracting from the communication with the transport unit and the CT unit, and additionally compiling a patient’s further therapy.

In the following section, we implement the specification of the emergency ward service (i.e., the open net *ew*) as executable WS-BPEL process. We present two slightly different implementations: the WS-BPEL process *ID* that does not conform to the specification *ew*, and the WS-BPEL process *MCT* that conforms to the specification *ew*. Subsequently, we use the WS-BPEL processes *ID* and *MCT* to demonstrate the approaches from this thesis, each with a negative and a positive example.

11.1.3 Two implementations in WS-BPEL

The Web Services Business Process Execution Language Version 2.0 (WS-BPEL) [130] is a language for specifying business process behavior based on

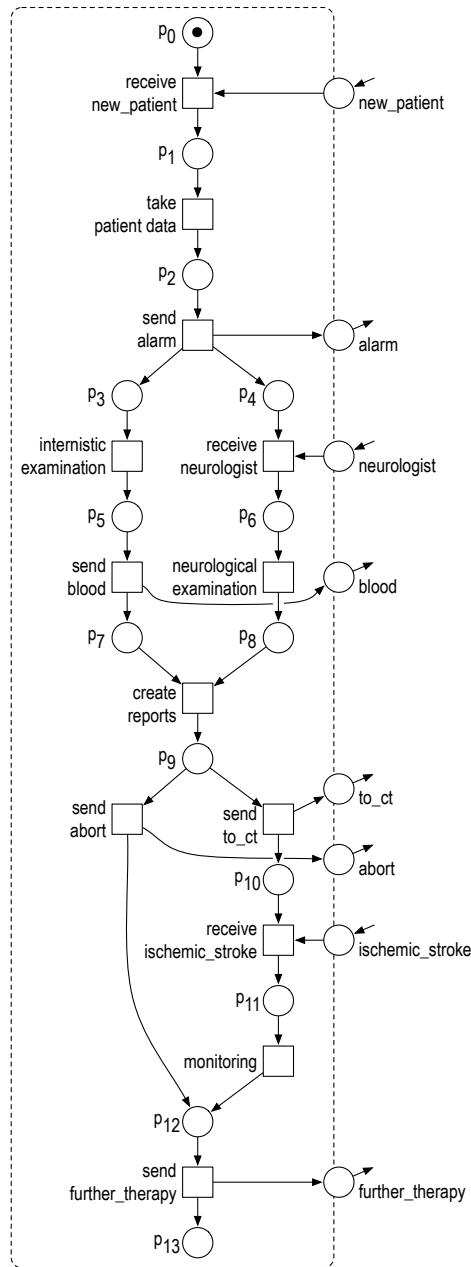


Figure 128: The specification ew . In addition to the figure, we have $\Omega_{ew} = \{p_{13}\}$.

web services. This makes WS-BPEL a language for the programming in the large paradigm [75]. A WS-BPEL process implements one web service by specifying its interactions with other web services exclusively through their interfaces. Thereby, WS-BPEL provides language features for advanced business process concepts such as instantiation, complex exception handling, and compensation of long running transactions. In the following, we briefly introduce the basic language constructs of WS-BPEL that are relevant for the remainder of this chapter. For a more in-depth treatment, we refer to the official WS-BPEL specification [130] or one of the detailed introductions, e.g., [252].

For specifying a business process, WS-BPEL provides *basic* and *structured activities*. A basic activity models an elementary action in the process; it can

communicate with other web services by exchanging messages (“invoke”, “receive”, and “reply” activity), manipulate or validate data (“assign”, or “validate” activity), wait for a period of time (“wait” activity) or do nothing (“empty” activity), signal faults (“throw” activity), invoke a compensation handler (through a “compensationHandler” wrapper for activities), or end the entire process instance (“exit” activity). A structured activity defines a causal order on (basic or structured) activities in the process. The structured activities include sequential or parallel execution (“sequence”, or “flow” activity), data-dependent branching (“if” activity), timeout- or message-dependent branching (“pick” activity), and repeated execution (“repeatUntil”, “while”, and “forEach” activity). In addition, the structured activity “scope” links fault, compensation, termination, and event handling to an activity. For communicating with other web services, a WS-BPEL process additionally defines partner links and port types with operations. In essence, a partner link models the interaction between two WS-BPEL processes (called “partners” in [130]), and a port type and its operations specify the involved message channels. Thereby, a WS-BPEL process represents all partners and interactions with these partners in terms of abstract WSDL [64] interfaces (i.e., the port types and operations).

WS-BPEL is intended as exchange and documentation format; it is based on XML and provides no graphical representation. Hence, many vendors of WS-BPEL development tools introduce their own graphical notations. In this thesis, we develop WS-BPEL processes using the free and open source software *Eclipse BPEL Designer* [235]. The Eclipse BPEL Designer adds comprehensive support for the definition, authoring, editing, deploying, testing and debugging of WS-BPEL processes to the well-known integrated development environment (IDE) Eclipse [236]. Therefore, we also use the graphical notation of the Eclipse BPEL Designer, as illustrated with Fig. 129. The WS-BPEL process in Fig. 129 implements the specification *ew* from Fig. 128. The implemented service starts—within the scope activity “emergencyward_main” of the whole process—with a receive activity receiving new patients (receive activity “emergencyward_receive_new_patient”). Then, the service manipulates data (assign activity “emergencyward_take_patient_data”) and invokes the neurology service by sending an alarm message (invoke activity “emergencyward_send_alarm”). The service continues with a flow activity, concurrently sending a blood sample to the laboratory service (invoke activity “emergencyward_send_blood” inside the left sequence activity) and receiving the alarmed neurologist (invoke activity “emergencyward_receive_neurologist” inside the right sequence activity). Just as specified in Fig. 128, the implemented service in Fig. 129 proceeds with a data-dependent branching (if activity “emergencyward_if”): In the left branch, the service decides against acute treatment of the stroke patient by sending an abort message to a documentation service (invoke activity “emergencyward_send_abort”), in the right branch, the service sends the stroke patient for an acute treatment to the CT service (invoke activity “emergencyward_send_to_ct” and receive activity “emergencyward_receive_ischemic_stroke”). In each case, the service in Fig. 129 finishes by compiling the further therapy for the stroke patient (assign activity “emergencyward_compile_further_therapy”) and returning this to other hospital services (reply activity “emergencyward_send_further_therapy”).

For demonstrating the approaches that we developed in this thesis, we present *two different* implementations of *ew*, which are slight modifications of the WS-BPEL process in Fig. 129: Figure 130 illustrates the first imple-

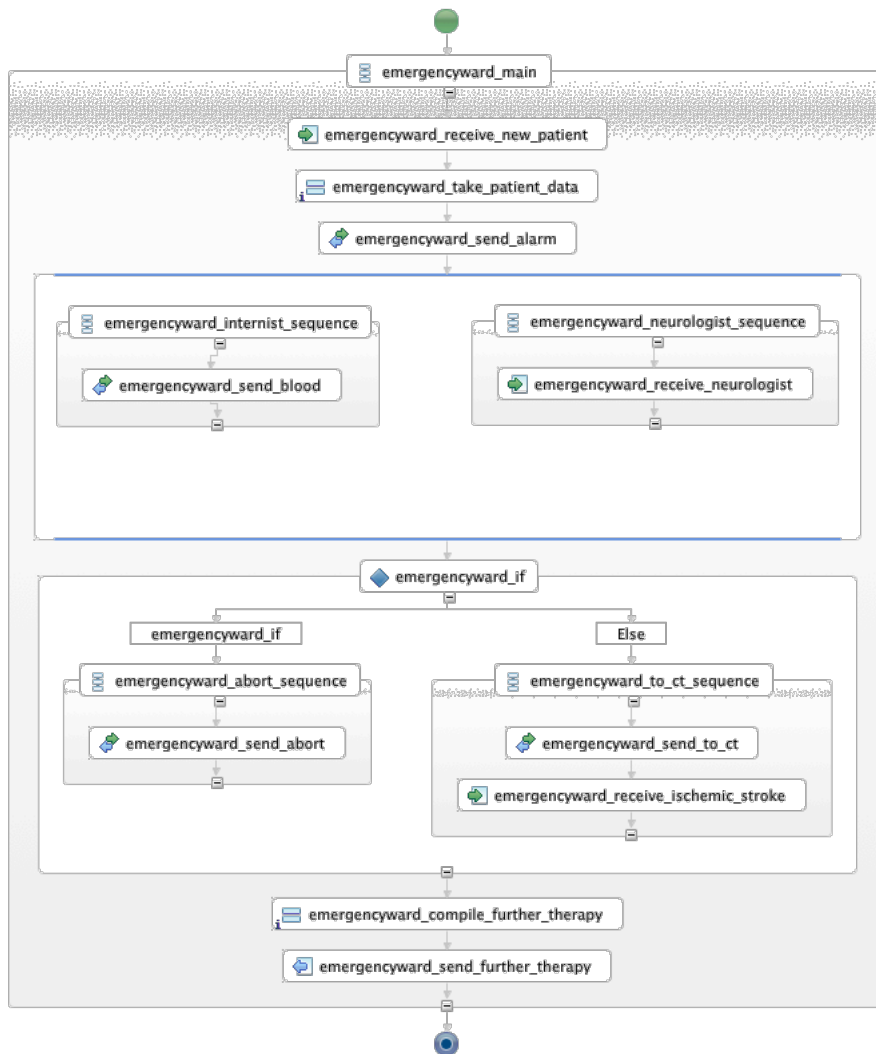


Figure 129: A straightforward implementation of the specification *ew* from Fig. 128 as a WS-BPEL process in Eclipse BPEL Designer.

mentation *ID* of *ew* as a WS-BPEL process. The service *ID* consists of the same activities as the WS-BPEL process in Fig. 129 except that we change their causal order: Rather than concurrently sending a blood sample to the laboratory service and receiving the alerted neurologist *before* making a decision about the patient’s acute treatment, the service *ID* concurrently sends a blood sample to the laboratory *and* makes a decision about the patient’s acute treatment while receiving the alerted neurologist. In other words, *ID* implements an emergency ward service where the decision for or against acute treatment is made solely by the treating internist rather than by the internist and the neurologist together. This clearly contradicts the regulations we outlined at the beginning of this section and in our specification *ew*. Therefore, *ID* should not 1-conform to *ew*.

Figure 131 illustrates the second implementation *MCT* of *ew* as a WS-BPEL process. The service implemented by *MCT* augments the service implemented by Fig. 129 by allowing for a second invocation: *MCT* may receive an ischemic stroke patient directly from the start (pick activity “emergency-

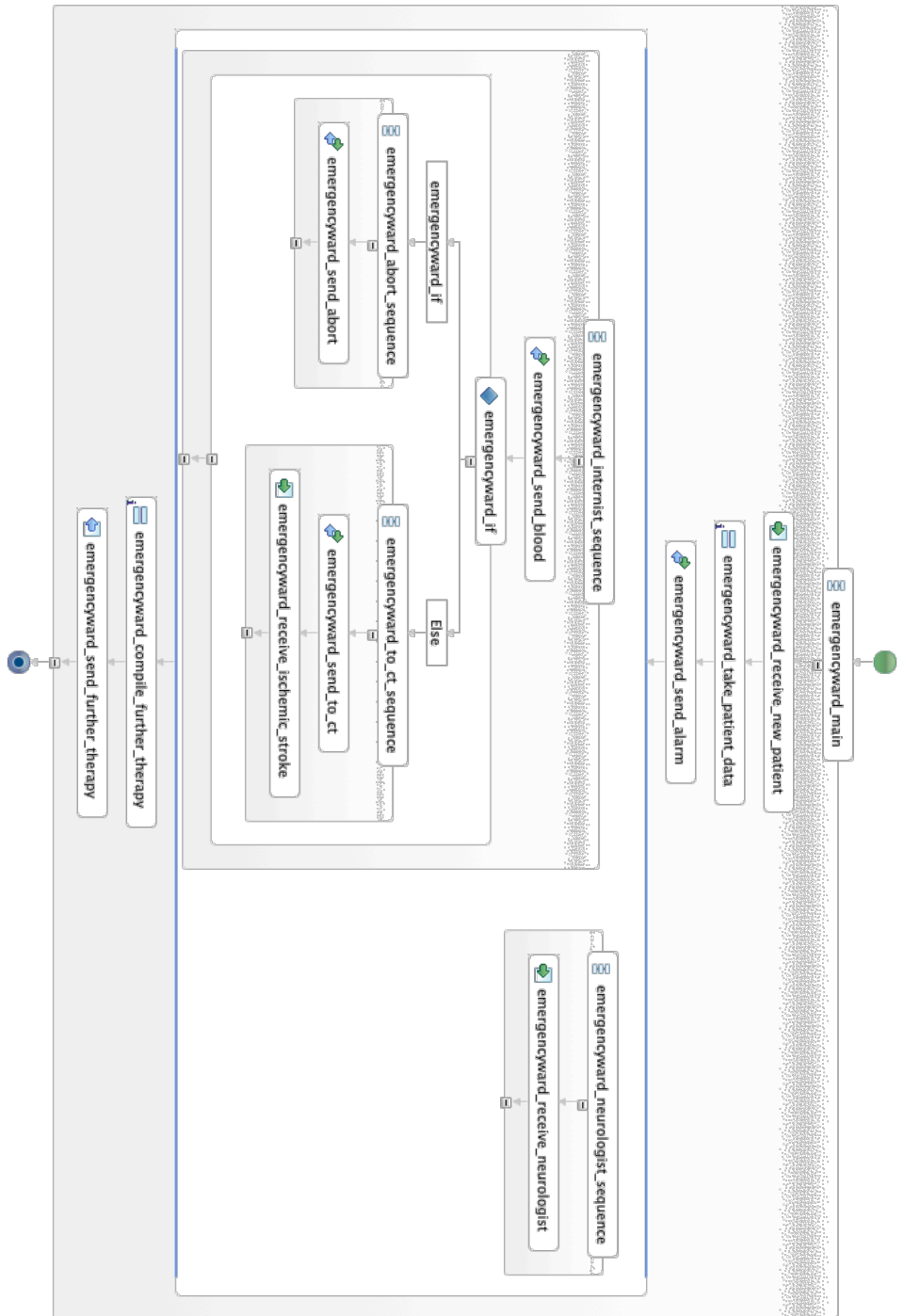


Figure 130: The WS-BPEL process *ID* as an implementation of our specification *ew* in Fig. 128 that does not 1-conform to *ew*.

ward_pick” and receive activity “ischemic_stroke”). The received ischemic stroke patient is then monitored (assign activity “emergencyward_monitor_the_patient”), a blood sample is send to the laboratory service (invoke activity “emergencyward_send_blood”), and the further therapy is compiled (assign activity “emergencyward_compile_further_therapy”) and subsequently send to other hospital services (reply activity “emergencyward_send_further_therapy”). *MCT* takes account of the recent development of “mobile stroke units” [93, 251]: A mobile stroke unit is a specialized ambulance containing a mobile CT unit and specialized staff, among others an internist and a neurologist. This allows to confirm an ischemic stroke and to directly start the acute treatment while the patient is delivered to the next emergency ward. As *MCT* additionally contains the behavior of *ew* (i.e., the left branch of *MCT* coincides with the WS-BPEL process in Fig. 129), *MCT* should 1-conform to the specification *ew*.

In the remainder of this chapter, we use the two implementations *ID* and *MCT* and the specification *ew* to demonstrate the approaches that we developed in this thesis.

11.2 THE MODEL-MODEL SCENARIO

In this section, we demonstrate our approach for conformance checking in the model-model scenario (see Chap. 5). Because of the model-model scenario, we assume the open net *ew* as specification and the two WS-BPEL processes *ID* and *MCT* as two implementations from Sect. 11.1 to be given.

For checking whether an implementation (e.g., the WS-BPEL process *ID*) 1-conforms to the specification (i.e., the open net *ew*), we have to take the following two steps:

1. Derive a formal model (i.e., an open net *id*) from the WS-BPEL process *ID*.
2. Compute the LTSs $CSD_1(id)$ and $CSD_1(ew)$ by Def. 104 and check $CSD_1(id)$ and $CSD_1(ew)$ according to Thm. 117.

In the following, we present each of the two steps in detail. As a convention for the remainder of this chapter, we easily distinguish a WS-BPEL process from its derived open net by writing the WS-BPEL process (e.g., *ID*) in uppercase letters and its derived open net (e.g., *id*) in lowercase letters. As in Part II and Part III, all computations are done on a MacBook Air model A1466 [21].

11.2.1 Step 1: Deriving formal models

For automatically deriving an open net from a WS-BPEL process, the free open source software BPEL2OWFN [149] implements a features-complete Petri nets semantics of WS-BPEL.

Figure 132 depicts the open net *id* that we derived automatically from *ID* using BPEL2OWFN. The transitions t_6 and t_7 visualize that deciding whether a stroke patient receives acute treatment or not is solely done by the internist; this contradicts what we specified by the open net *ew* in Sect. 11.1. Therefore, *id* does not 1-conform to *ew*: For example, the trace $(new_patient)(alarm)(blood)(abort)$ is in $L(id) \subseteq uL_1(id)$ but neither in $L(ew) \subseteq uL_1(ew)$ nor in $uncov_1(ew) \subseteq uL_1(ew)$. This implies that *id* does not 1-conform to *ew* by Thm. 97.

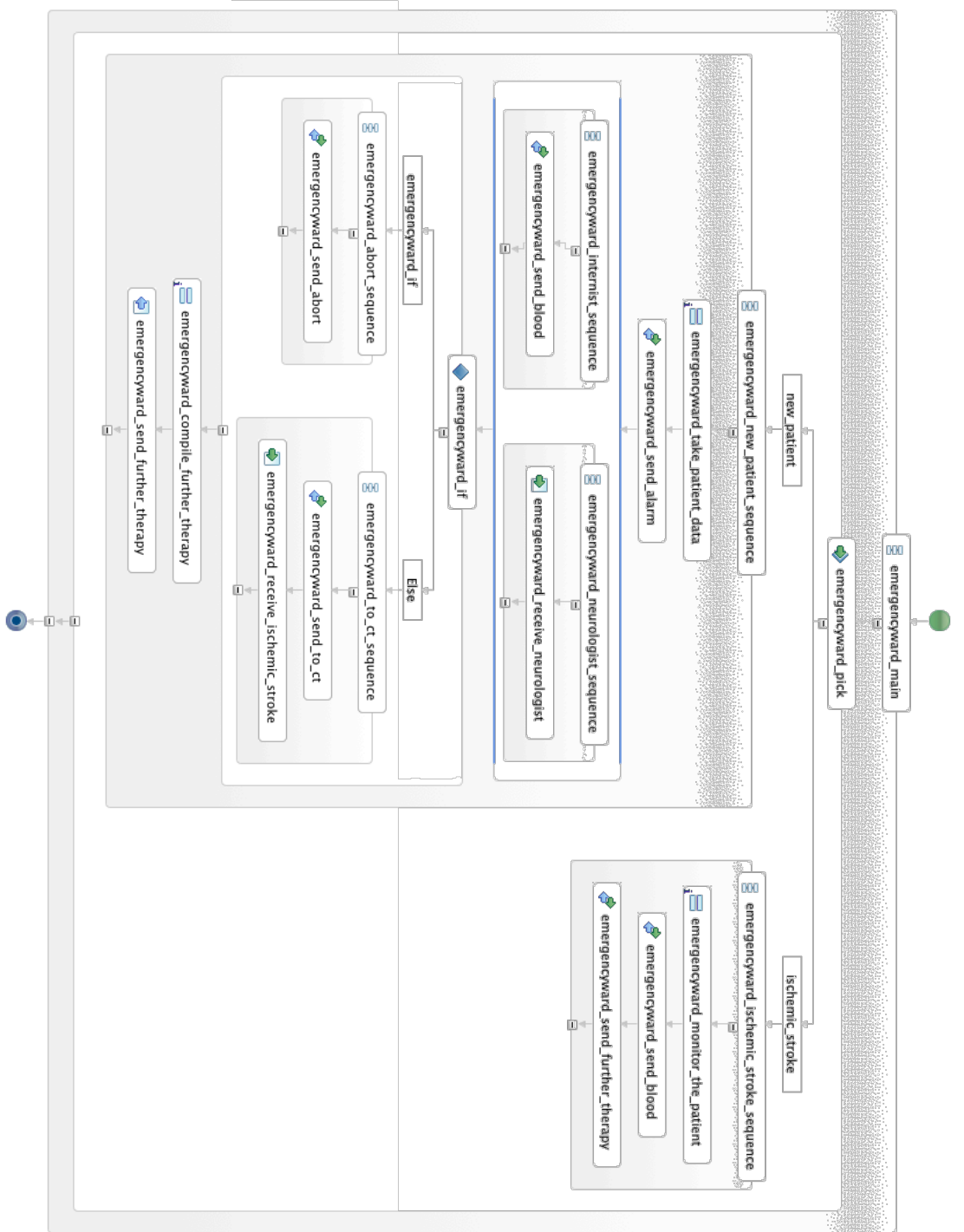


Figure 131: The WS-BPEL process *ID* as an implementation of our specification *ew* in Fig. 128 that 1-conforms to *ew*.

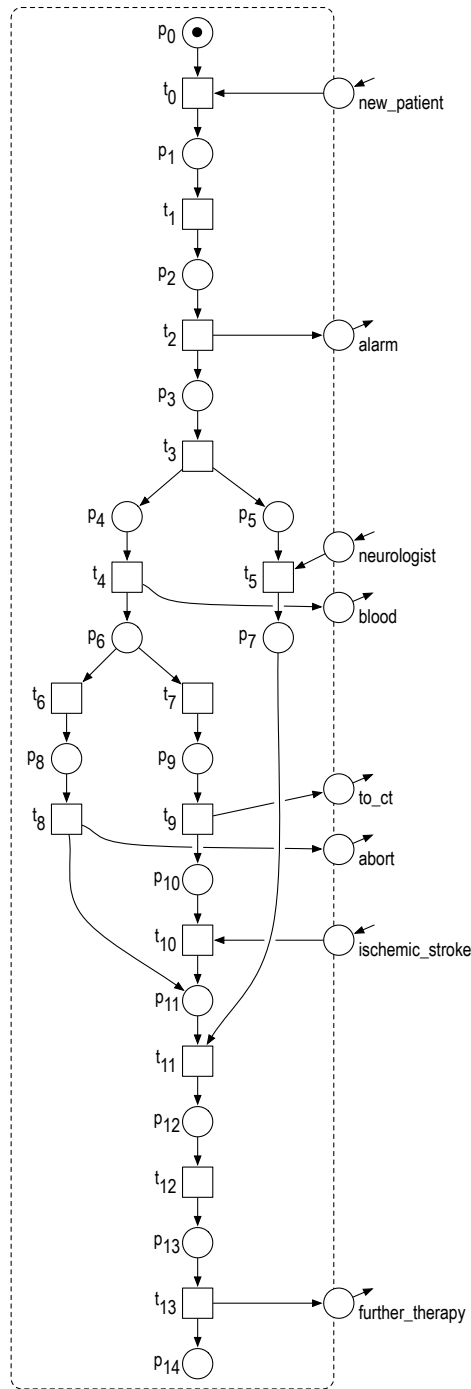


Figure 132: The open net id that we derived from the WS-BPEL process ID from Fig. 130. In addition to the figure, we have $\Omega_{id} = \{[p_{14}]\}$.

Figure 133 depicts the open net mct that we automatically derived from MCT using $BPEL2OWFN$. The control flow of mct immediately branches from the initial marking because of the transitions t_0 and t_6 : In the left branch, mct resembles the open net ew from Fig. 128 and the left branch of MCT . In the right branch, an ischemic stroke patient from a mobile CT is received and subsequently treated, which refers to the right branch of MCT . The open net mct 1-conforms to ew . Each trace of mct that is not a trace of ew starts with $ischemic_stroke$, and every trace starting with $ischemic_stroke$ is in

$uncov_1(ew)$ because of the transitions “send abort” and “send to_ct”: If a 1-partner C of ew puts a token onto the interface place $ischemic_stroke$ without receiving a token from to_ct first, ew may fire transition “send abort”. Then, the token on $ischemic_stroke$ cannot be removed and the final marking of ew is no longer reachable, which hinders 1-responsiveness of ew and C . Thus, mct 1-conforms to ew .

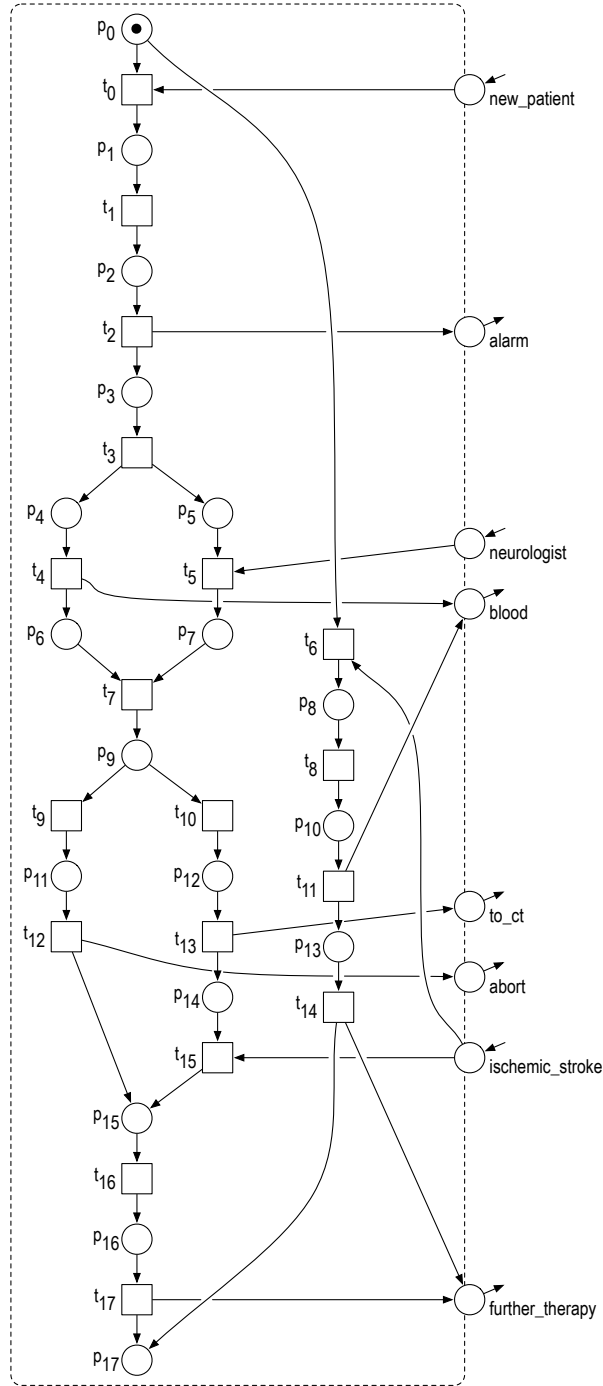


Figure 133: The open net mct that we derived from the WS-BPEL process MCT from Fig. 131. In addition to the figure, we have $\Omega_{mct} = \{[p_{17}]\}$.

Table 25 gives an overview over the characteristics of the open nets id and mct that we derived from the WS-BPEL processes ID and MCT . The size of

id and mct is between the size of the running examples from Part II and the industrial-sized open nets we used to evaluate our approach in Sect. 5.4.

open net	$ P $	$ I $	$ O $	$ T $	$ F $
ew	22	3	5	13	36
id	23	3	5	14	38
mct	26	3	5	18	49

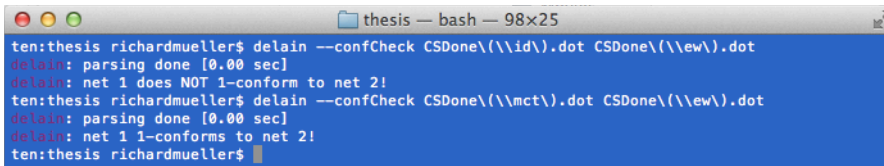
Table 25: The size of the open net ew from Sect. 11.1, and the open nets id and mct that we generated using the tool BPEL2OWFN. We do not include the memory usage and time because id and mct could be generated instantly from the given WS-BPEL processes ID and MCT , respectively.

11.2.2 Step 2: Checking for 1-conformance

We compute the LTSs $CSD_1(ew)$, $CSD_1(id)$, and $CSD_1(mct)$ using the tool Chloe [115]. All three LTSs can be computed instantly; Tab. 26 gives an overview over the size of the LTSs. Using the tool Delain [78], we check for 1-conformance by relating $CSD_1(id)$ and $CSD_1(ew)$, and $CSD_1(mct)$ and $CSD_1(ew)$, respectively, according to Thm. 117. Both checks can be performed instantly; Fig. 134 shows a screenshot of the two conformance checks using Delain. As a result, id does not 1-conform to ew and mct 1-conforms to ew , as we have already argued in the previous step.

LTS	$ Q $	$ \delta $	$ \Sigma^{in} $	$ \Sigma^{out} $	time (s)	memory (KiB)
$CSD_1(ew)$	32	256	3	5	0	2,044
$CSD_1(id)$	44	352	3	5	0	2,268
$CSD_1(mct)$	34	272	3	5	0	2,076

Table 26: The size of CSD_1 generated with the tool Chloe, including the used memory and time.



```

ten:thesis richardmuellers$ delain --confCheck CSDone(\\id\).dot CSDone(\\ew\).dot
delain: parsing done [0.00 sec]
delain: net 1 does NOT 1-conform to net 2!
ten:thesis richardmuellers$ delain --confCheck CSDone(\\mct\).dot CSDone(\\ew\).dot
delain: parsing done [0.00 sec]
delain: net 1 1-conforms to net 2!
ten:thesis richardmuellers$

```

Figure 134: A screenshot of the tool Delain checking for 1-conformance of id and ew , and mct and ew , respectively, by using the LTSs $CSD_1(ew)$, $CSD_1(id)$, and $CSD_1(mct)$ that we previously computed using the tool Chloe.

11.3 THE LOG-MODEL SCENARIO

In this section, we assume the specification ew to be given, but the WS-BPEL processes ID and MCT from Sect. 11.1 and their derived open nets id and mct from Sect. 11.2 are unavailable. Instead, we assume that ID and MCT are running in a WS-BPEL engine from which we are given observed behavior in the form of two event logs. In Sect. 11.3.1, we show the validity of this assumption by demonstrating how to deploy the WS-BPEL processes

into Apache ODE [233] and subsequently derive event logs. We test both unknown implementations for 1-conformance to ew based on the generated event logs in Sect. 11.3.2, and discover a high-quality open net of the 1-conforming implementation mct in Sect. 11.3.3.

For testing whether an unknown implementation (e.g., the open net id from the WS-BPEL process ID) 1-conforms to the specification (i.e., the open net ew) and—if id 1-conforms to ew —for discovering a high-quality formal model of ID , we have to take the following three steps:

1. Deploy ID using a WS-BPEL engine and derive an event log $IDLog$ from observed example interactions between ID and its environment.
2. Compute the open net $mp_1(max_1(ew))$ by Def. 131 and Def. 110 and test for 1-conformance of the unknown id to ew by replaying $IDLog$ on $env(mp_1(max_1(ew)))$ according to Thm. 177.
3. If the unknown id 1-conforms to ew , we can discover a high-quality formal model id' from $IDLog$ and ew using the genetic algorithm from Sect. 9.3.

In the following, we present each of the three steps in detail.

11.3.1 Step 1: Deriving event logs

As already explained at the beginning of this chapter, we assume that ID and MCT are running in a WS-BPEL engine. We demonstrate the validity of this assumption by deploying ID and MCT into Apache ODE [233] and subsequently deriving two event logs $IDLog$ and $MCTLog$ from observed example interactions.

Apache ODE is a widely used free and open source WS-BPEL engine; it executes processes which have been expressed in WS-BPEL by communicating with other (web) services, manipulating data, and handling exceptions. Apache ODE is a top-level project at the Apache Software Foundation [233] and is incorporated in various open source enterprise services buses (ESBs). Among several available open source WS-BPEL engines, Apache ODE has particular high compliance [111] to the WS-BPEL standard [130]. Although Apache ODE is open source, it is on par with proprietary WS-BPEL engines in terms of compliance to the WS-BPEL standard, performance, and language expressiveness in terms of workflow pattern support [112]. In the following setting, we run Apache ODE version 1.3.6 on a local Apache Tomcat server version 8.0.3, which is a free open source software implementation of the Java Servlet and JavaServer Pages technologies [234].

For deriving event logs from observed example interactions, we use ODE's event mechanism: Apache ODE produces detailed information about process executions (among others, the sending and receiving of messages) and persistently stores them in an internal database. Thereby, Apache ODE allows the registration of an event listener, which may catch and analyze all produced events before they are stored. An own event listener can be used by implementing the `org.apache.ode.bpel.iapi.BpelEventListener` interface of the Apache ODE API. For our setting, we implement a custom event listener [189] that exports sent or received messages from the viewpoint of the WS-BPEL process together with the identifier of the corresponding WS-BPEL process instance and a timestamp into comma-separated values. Comma-separated values in turn can be imported as event logs into ProM [212].

For generating example interactions with the deployed WS-BPEL processes *ID* and *MCT*, we use the existing tool *SoapUI* [225]. *SoapUI* is a free and open source web service testing application for service-oriented architectures. Its functionality covers web service inspection, invoking, development, simulation and mocking, functional testing, load and compliance testing [225]. Load testing is performed to determine a web service's behavior under load conditions by generating a high number of interactions between the web service and its environment. For our setting, we use two load tests with the standard (random) settings and a runtime of 60 seconds to generate example interactions between *ID* and its environment (i.e., *SoapUI*), and *MCT* and its environment, respectively.

In the following, we abbreviate the messages *new_patient* by *patient*, *to_ct* by *ct*, *ischemic_stroke* by *ischemic*, and *further_therapy* by *therapy*. For *ID*, we derive the event log *IDLog* in Tab. 27, which contains 405 event traces and 2700 events. Thereby, all event traces of *IDLog* start with the event *patient*. We can clearly recognize the event traces *patient alarm blood ct ischemic neurologist therapy*, *patient alarm blood abort neurologist therapy*, and *patient alarm blood ct neurologist ischemic therapy* that violate (1-)conformance of *ID* to *ew*: In all three cases, the decision whether the stroke patient receives acute treatment (i.e., the patient is sent to the CT) or not (i.e., the abort message is sent) is done before the neurologist arrives. For *MCT*, we derive the event log *MCTLog* in Tab. 28, which contains 354 event traces and 2327 events. In contrast to the event log *IDLog*, there exist event traces in *MCTLog* that do not start with the event *patient*: the event traces *ischemic blood therapy* representing the interaction with the mobile CT. Nevertheless, these traces should not hinder (1-)conformance of *MCT* to *ew*, as we already explained in Sect. 11.2. Both event logs *IDLog* and *MCTLog* have a size comparable to the size of the event logs we used to evaluate our approaches in Chap. 8 and Chap. 9.

cardinality	event trace
177	<i>patient alarm neurologist blood ct ischemic therapy</i>
81	<i>patient alarm neurologist blood abort therapy</i>
45	<i>patient alarm blood neurologist ct ischemic therapy</i>
42	<i>patient alarm blood ct ischemic neurologist therapy</i>
30	<i>patient alarm blood neurologist abort therapy</i>
24	<i>patient alarm blood abort neurologist therapy</i>
6	<i>patient alarm blood ct neurologist ischemic therapy</i>

Table 27: The event log *IDLog* that we derived by observing example interactions between the WS-BPEL process *ID* and its environment while taking the viewpoint of *ID*. We abbreviate the messages *new_patient* (*patient*), *to_ct* (*ct*), *ischemic_stroke* (*ischemic*), and *further_therapy* (*therapy*).

11.3.2 Step 2: Testing for 1-conformance

For testing whether the unknown implementation *id* of *ID* 1-conforms to *ew*, we have to compute the open net $mp_1(max_1(ew))$ and replay the derived event log *IDLog* on $env(mp_1(max_1(ew)))$ (see Chap. 8). If *IDLog* cannot be replayed on $env(mp_1(max_1(ew)))$ (i.e., the costs of replaying *IDLog* on $env(mp_1(max_1(ew)))$ exceed 0), then *id* does not 1-conform to *ew*. The same

cardinality	event trace
152	<i>patient alarm neurologist blood ct ischemic therapy</i>
78	<i>patient alarm blood neurologist ct ischemic therapy</i>
77	<i>patient alarm neurologist blood abort therapy</i>
38	<i>patient alarm blood neurologist abort therapy</i>
9	<i>ischemic blood therapy</i>

Table 28: The event log *MCTLog* that we derived by observing example interactions between the WS-BPEL process *MCT* and its environment while taking the viewpoint of *MCT*. We abbreviate the messages *new_patient* (*patient*), *to_ct* (*ct*), *ischemic_stroke* (*ischemic*), and *further_therapy* (*therapy*).

approach allows to test whether the unknown implementation *mct* of *MCT* 1-conforms to *ew*, too.

We compute the maximal 1-partner $max_1(ew)$ and the most-permissive 1-partner $mp_1(max_1(ew))$ of the open net *ew* using the tool Chloe [115]. The resulting open net $mp_1(max_1(ew))$ is too big to be shown here; it consists of 51 places, 174 transitions, and 514 arcs. We then test for 1-conformance of the unknown implementations *id* and *mct* to *ew* by replaying the event logs *IDLog* and *MCTLog* on the labeled net $env(mp_1(max_1(ew)))$, respectively. Replaying an event log on a labeled net can be done using ProM [212]. We use the package “PNetReplayer” that implements the A^* -algorithm [15] and that is part of the current ProM release version 6.3 [212]. All settings of “PNetReplayer” were left to the standard settings except for the cost function: As already detailed in Conv. 9, we use a cost function that assigns cost of 1 to each log move and to each non-silent model move, and cost of 0 to all other moves.

Table 29 shows the results: Replaying *IDLog* on $env(mp_1(max_1(ew)))$ results in costs greater than 0; thus, the unknown implementation *id* of *ID* cannot 1-conform to *ew* by Thm. 177. Figure 135 shows a screenshot of ProM visualizing the alignments of *IDLog* to $env(mp_1(max_1(ew)))$. The alignment in the middle contains a log move that is shown in yellow: This log move corresponds to the event *to_ct* preceding the event *neurologist*; in other words, sending the stroke patient to the CT is not allowed by the labeled net $env(mp_1(max_1(ew)))$ at the corresponding state of the process. In the upper right, ProM outputs the costs (labeled “Raw Fitness Cost”) for replaying the whole event log on $env(mp_1(max_1(ew)))$.

Replaying *MCTLog* on $env(mp_1(max_1(ew)))$ yields costs 0. Therefore, at least with respect to the observed behavior captured in *MCTLog*, we cannot make any statement whether the unknown implementation *mct* of *MCT* 1-conforms to *ew*. In other words, there is no erroneous behavior captured by the event log *MCTLog*. The runtime of replaying the event logs *IDLog* and *MCTLog* on the labeled net $env(mp_1(max_1(ew)))$ using the A^* -algorithm [15] was 0.002 and 0.107 seconds, respectively, which is nearly instantaneously.

labeled net	event log	replay costs	time (s)
$env(mp_1(max_1(ew)))$	<i>IDLog</i>	0.1778	0.107
$env(mp_1(max_1(ew)))$	<i>MCTLog</i>	0	0.002

Table 29: The time and cost ProM reported for replaying the event logs *IDLog* and *MCTLog* on the labeled net $env(mp_1(max_1(ew)))$. We used the standard settings of the package “PNetReplayer” and the cost function from Conv. 9.

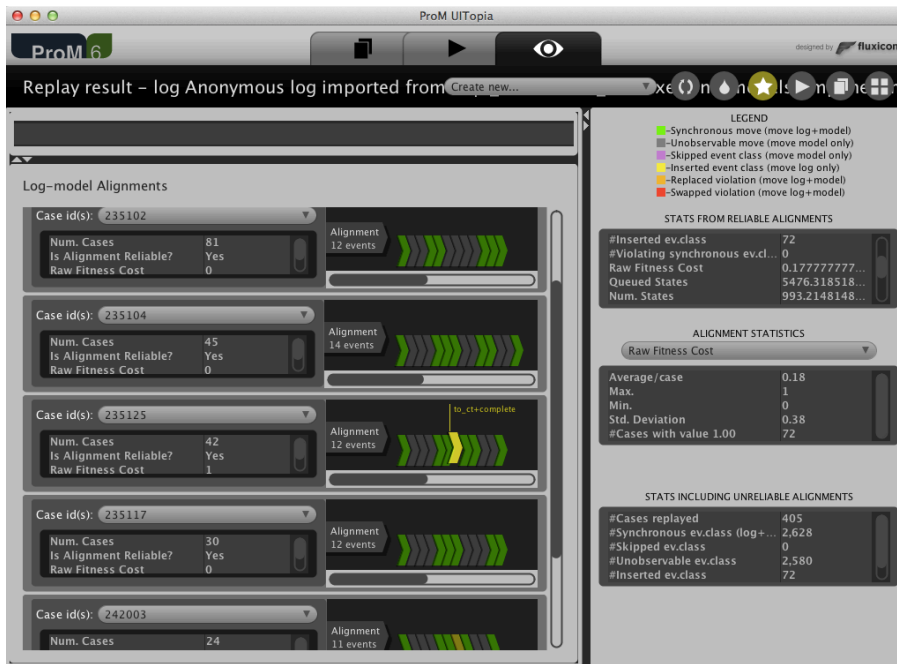


Figure 135: Visualizing the alignments of $IDLog$ to $env(mp_1(max_1(ew)))$ in ProM.

Note that it is not possible to infer whether id or mct 1-conform to ew by solely examining $IDLog$ and ew , or $MCTLog$ and ew , respectively: Both event logs $IDLog$ and $MCTLog$ contain traces that are not in the language of the labeled net $env(ew)$ —for example, the trace *patient alarm blood ct neurologist ischemic therapy* of $IDLog$ and the trace *ischemic blood therapy* of $MCTLog$. In other words, the distinction between the unknown implementations id and mct (i.e., id does not 1-conform to ew , but mct does) is not obvious in $IDLog$ and $MCTLog$; this again illustrates that Thm. 177 is nontrivial.

11.3.3 Step 3: Discovering a high-quality model of a 1-conforming implementation

In Sect. 11.3.2, we showed that the unknown implementation id cannot 1-conform to the specification ew , whereas mct may 1-conform to ew . In this final step, we assume that the unknown implementation mct 1-conforms to ew ; in practice, this assumption may be justified by the previous (non-negative) testing result. Then, we discover a high-quality open net mct' from $MCTLog$ and ew that 1-conforms to ew . In other words, mct' may serve—instead of mct —as a formal model of MCT .

For discovering mct' from $MCTLog$ and ew , we use our discovery approach from Chap. 9 with the following four inputs:

- The LTS $MP_1(max_1(ew))$ with its Boolean annotation ϕ , which we can compute from ew using the tool Chloe [115].
- An event log $MCTLog$ of an open net that 1-conforms to ew , which we already generated in the first step using Apache ODE [233], a custom event listener, and the testing tool SoapUI [225].
- The weights for the four quality dimensions, which we set to 1 for simplicity and to 2 for all other quality dimensions as we did in Chap. 9.

- The parameters and termination criteria for the genetic algorithm: Like in Chap. 9, we use an initial population of 100 individuals, a mutation/crossover/replacement probability of 0.3 with at most 1 crossover point, and elitism of 0.3. The computation of a new generation stops after 1,000 generations, if the highest quality stagnates for 750 generations, if a quality of 0.999 is reached, or if the algorithm ran for 60 minutes.

We employ the abstraction technique from Sect. 9.2, because this, in general, produces significantly better results while taking significantly less time (cf. Sect. 9.4). Note that this implies $mct' \neq mct$, i.e., we cannot discover mct from $MCTLog$ and ew : The open net mct is not a 1-subnet of ew , and by using the abstraction technique, we restrict our search space to 1-subnets of ew only. Nevertheless, mct may serve as a benchmark of the quality of the discovered open net mct' . The open net mct has a fitness value of 1.0 (i.e., every event trace of $MCTLog$ can be replayed on $inner(mct)$), a simplicity value of 0.6944, a precision value of 1.0, and a generalization value of 0.9970 with respect to $MCTLog$ and ew . Thus, the quality of mct with respect to $MCTLog$ and ew is approx. 0.9555.

The output of our implementation is an open net mct' that 1-conforms to ew and has high quality with respect $MCTLog$ and ew ; Fig. 136 depicts the discovered open net mct' . Discovering mct' took approx. 33 seconds. The open net mct' has a fitness value of 1.0 (i.e., every event trace of $MCTLog$ can be replayed on $inner(mct')$), a simplicity value of 1.0 (because it is a 1-subnet of ew), a precision value of 0.8963 and a generalization value of 0.9970 with respect to $MCTLog$ and ew . Thus, the quality of mct' with respect to $MCTLog$ and ew is approx. 0.9695.

The quality of mct' is higher than the quality of mct with respect to $MCTLog$ and ew , mainly because mct' is simpler than mct . The open net mct' is smaller than mct and, in contrast to mct , τ -free. This is because $inner(mct')$ derives from (an initialized subsystem of) the LTS $MP_1(max_1(ew))$, which in turn is τ -free by construction. Compared to the unknown open net mct , mct' explicitly allows for the neurologist to arrive *before* the alarm message was send (i.e., the neurologist may stay at the emergency ward). This also illustrates why the open net mct' is likely not discovered using traditional discovering techniques from the area of process mining [2]: An event trace $w = \text{neurologist patient alarm blood ct ischemic therapy}$ was never observed in $MCTLog$ in Tab. 28. Nevertheless, every open net that 1-conforms to ew must allow for w , as *neurologist* is an input place of ew and consuming a token from *neurologist* may be delayed due to the asynchronous communication.

Figure 137 highlights the specification ew from Fig. 128 inside the discovered open net mct' . Clearly, mct' allows for more traces than ew ; for example, the trace *ischemic blood therapy*.

11.4 CONCLUSIONS

In this chapter, we demonstrated the applicability of the techniques and analysis tools that we developed in Part II and Part III on a real-life example: We formally specified the emergency ward of a hospital as an open system in form of a (web) service and checked two slightly different WS-BPEL implementations for 1-conformance to this specification. In addition, we showed how to derive event logs from the implementations using the well-known and widely used Apache ODE engine. Based on the derived

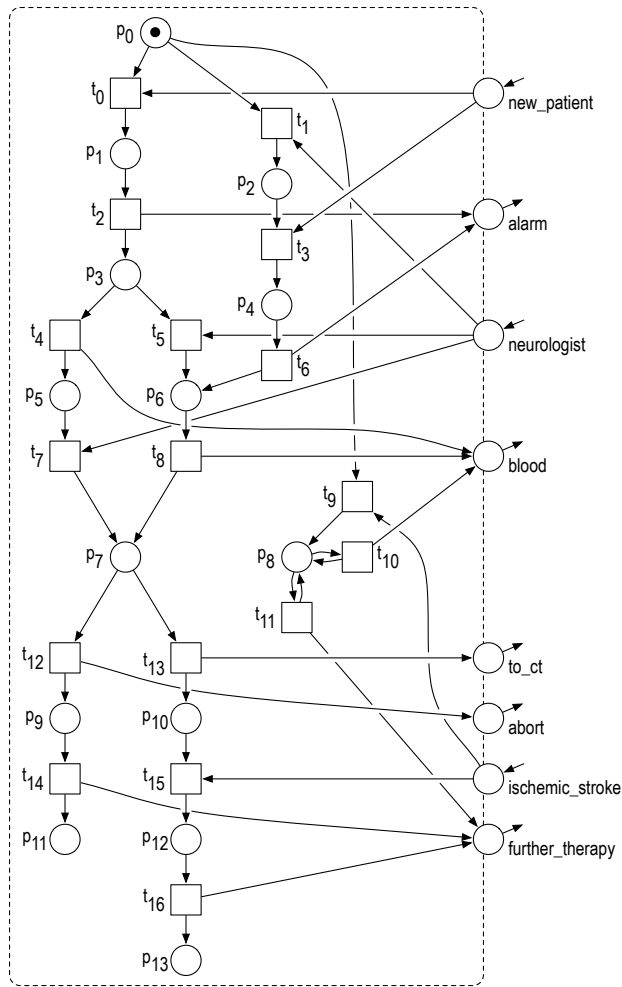


Figure 136: The open net mct' that we discovered from the event log $MCTLog$ and the LTS $MP_1(max_1(ew))$ with Boolean annotation. In addition to the figure, we have $\Omega_{mct} = \{[p_8], [p_{11}], [p_{13}]\}$.

event logs, we tested both implementations for 1-conformance and discovered a high-quality formal model of the conforming implementation. All tools used in this chapter—either developed in the context of this thesis, or already existing tools—are free and open source software.

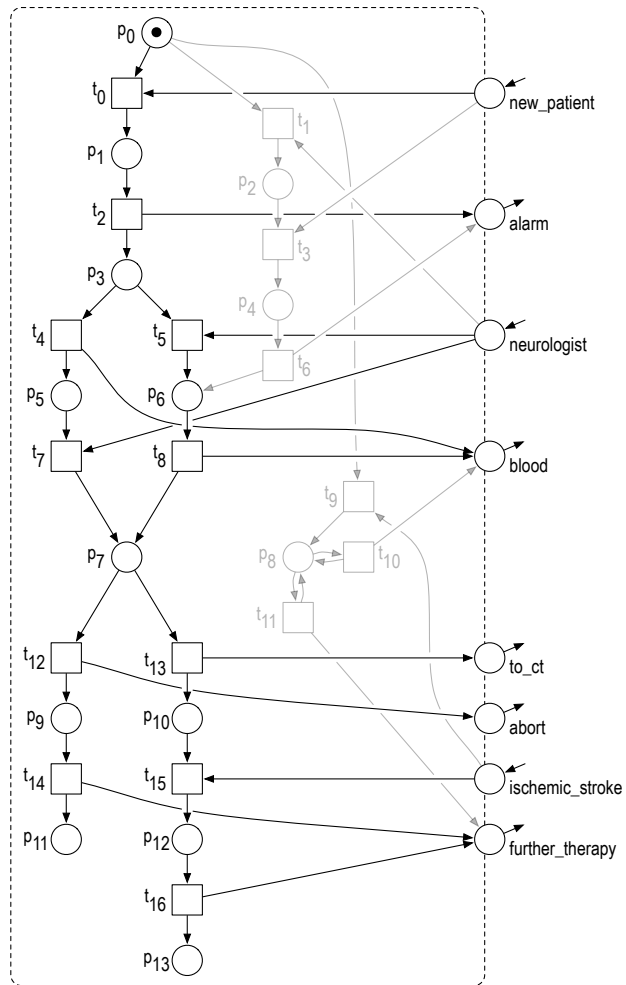


Figure 137: Highlighting the specification *ew* from Fig. 128 inside the discovered open net *mct'* from Fig. 136.

THIS final chapter concludes our thesis. We summarize the main contributions of our approach for verifying responsiveness for open systems by means of conformance checking in Sect. 12.1. In Sect. 12.2, we discuss open research questions of this approach and theoretical and practical limitations of the presented results. Finally, Sect. 12.3 sketches ideas for future research.

12.1 SUMMARY OF CONTRIBUTIONS

The central research topic of this thesis was to *verify responsiveness for open systems by means of conformance checking*. Responsiveness ensures mutual termination or perpetual communication between two systems. It is a fundamental behavioral correctness criterion for open systems; a nonterminating composition of two open systems that do not have the possibility to communicate is certainly ill-designed. In Chap. 3, we motivated two variants of responsiveness—responsiveness and b -responsiveness—and compared them to other behavioral correctness criteria for open systems. The notion of b -responsiveness is a variant of responsiveness where the number of pending messages never exceeds a previously known bound b . Although respecting a bound b may seem restricting, b -responsiveness is practically relevant: Distributed systems operate on a middleware with buffers that are of bounded size. The actual buffer size can be the result of a static analysis of the underlying middleware or of the communication behavior of an open system, or simply be chosen sufficiently large.

A conformance relation for responsiveness describes when one open system (i.e., the specification) can be safely replaced by another open system (i.e., the implementation) without affecting responsiveness with an unknown environment (i.e., other open systems called partners). We defined the conformance relations for responsiveness and b -responsiveness—that is, conformance and b -conformance—and the coarsest precongruences contained therein—that is, compositional conformance and compositional b -conformance.

We aimed to verify responsiveness by means of conformance checking in two distinct scenarios—the model-model scenario and the log-model scenario—each of which we investigated in a separate part of this thesis.

12.1.1 The model-model scenario

For the model-model scenario, we assume that both the specification and the implementation of an open system are given as formal models. Then, we can verify responsiveness by checking for conformance between the two formal models.

In Chap. 4, we analyzed conformance and compositional conformance in detail. We provided open nets with the *stopdead*-semantics and showed that set-wise inclusion of the *stopdead*-semantics characterizes conformance. In addition, we detailed that compositional conformance cannot be characterized with the *stopdead*-semantics or, in general, a denotational semantics

weaker than standard failures semantics. Therefore, we provided open nets with the \mathcal{F}_{fin}^+ -semantics (i.e., an extension of standard failure semantics) and showed that refinement on the \mathcal{F}_{fin}^+ -semantics characterizes compositional conformance. Based on the characterizations of conformance and compositional conformance, we showed that both relations are undecidable.

In Chap. 5, we investigated the b -conformance relation. We provided open nets with a trace-based semantics—the b -coverable *stopdead*-semantics—and showed that set-wise inclusion of the b -coverable *stopdead*-semantics characterizes b -conformance. Giving an answer to an open question, we showed that b -conformance is strictly larger than compositional b -conformance (i.e., b -conformance is a preorder but not a precongruence).

In contrast to conformance, b -conformance is decidable. Thus, we elaborated a decision procedure to decide whether an open net *Impl* b -conforms to an open net *Spec* based on two LTSs $CSD_b(Impl)$ and $CSD_b(Spec)$. For a given open net, we additionally developed a finite characterization of all b -conforming open nets based on the notion of a maximal b -partner; this finite characterization serves as an alternative decision procedure for b -conformance.

In Chap. 6, we investigated compositional b -conformance—that is, the coarsest precongruence that is contained in the b -conformance relation. We provided open nets with a failure-based semantics (the b -bounded \mathcal{F}_{fin}^+ -semantics) and showed that refinement on the b -bounded \mathcal{F}_{fin}^+ -semantics characterizes compositional b -conformance. Based on our characterization, we proved compositional b -conformance to be decidable by reducing it to deciding should testing. Thereby, the decision procedure presented in Chap. 6 does not depend on open nets but is independent from the concrete model.

12.1.2 The log-model scenario

For the log-model scenario, we assume the specification of an open system to be given as a formal model, but no formal model of the implementation is available. Instead, we assume that observed behavior of the running but unavailable implementation is given in the form of an event log.

In Chap. 8, we presented a testing approach for b -conformance. Testing for b -conformance can show that the implementation does not b -conform to the specification if the event log contains some erroneous behavior. To this end, we elaborated a necessary condition for b -conformance of the implementation *Impl* to the specification *Spec* based on the open net $mp_b(max_b(Spec))$ of the specification *Spec*: If the event log cannot be replayed on the environment of $mp_b(max_b(Spec))$, then *Impl* does not b -conform to *Spec*. We showed the existence of the open net $mp_b(max_b(Spec))$ and demonstrated that it can be automatically constructed.

In Chap. 9, we presented a technique to discover a system model of an unknown implementation from a given system model *Spec* and observed behavior of that implementation interacting with its environment. Our technique produces an open net *Impl* that b -conforms to *Spec* and, in addition, balances four conflicting quality dimensions: fitness, simplicity, precision, and generalization. As an additional improvement, we proposed an abstraction technique to reduce the infinite search space to a finite one. We can also apply our approach to discover a b -partner *C* of an open net *N* such that *C* has, among the set of all b -partners of *N*, high quality with respect to *N* and a given event log.

12.1.3 *Tool support*

We proposed several algorithms in this thesis: For the model-model scenario, we defined verification algorithms to decide b -conformance. For the log-model scenario, we defined a testing algorithm and a discovery algorithm. All algorithms of this thesis have been prototypically implemented in software tools. These tools follow a “one tool - one purpose” policy, which has been proven helpful in implementing a theory of correctness for open systems [155]. In this thesis, we used the following tools:

CHLOE is a tool to represent the semantics of an open net N . It represents the b -bounded *stopdead*-semantics and the b -coverable *stopdead*-semantics of N by computing the LTSs $BSD_b(N)$ and $CSD_b(N)$, respectively (see Chap. 5). As a side-effect of computing $CSD_b(N)$, Chloe can output the LTS $MP_b(N)$ with or without Boolean annotation, the most-permissive b -partner $mp_b(N)$, and the maximal b -partner $max_b(N)$ of N ; these three artifacts serve as a basis for conformance testing (see Chap. 8) and system discovery (see Chap. 9). We use version 2.0 [115], which is licensed under the GNU Affero General Public License.

DELAIN is a tool to decide whether an open net $Impl$ b -conforms to an open net $Spec$. To this end, Delain checks for set-wise language inclusion of their respective b -coverable *stopdead*-semantics using the previously computed LTS $CSD_b(Impl)$ and $CSD_b(Spec)$ (see Chap. 5). We use version 0.3 [78], which is licensed under the GNU Affero General Public License.

LOCRETIA is a tool to randomly create an artificial event log Log from an open net N , using either the viewpoint of N or the viewpoint of N 's environment (see Chap. 8). We used artificial event logs for evaluating our approaches for conformance testing (see Chap. 8) and system discovery (see Chap. 9). As a side-effect, Locretia can output the labeled nets $env(N)$ and $inner(N)$ for any open net N . We use version 1.1 [116], which is licensed under the GNU Affero General Public License.

SERVICE DISCOVERY is a tool for open system discovery: Given an open net $Spec$ and an event log Log with observed behavior of an unknown implementation of $Spec$, we can discover an open net $Impl$ that b -conforms to $Spec$ and, in addition, has high or even highest quality with respect to Log and $Spec$ (see Chap. 9). Service Discovery [188] is a ProM plug-in licensed under the GNU Public License.

CSVEXPORT is an event listener for Apache ODE [233]. CSVExport outputs sent or received messages from the viewpoint of a deployed WS-BPEL process together with the identifier of the corresponding process instance and a timestamp into comma-separated values. Comma-separated values in turn can be imported as event logs into ProM [212]. CSVExport [189] is licensed under the Apache License version 2.

ECLIPSE BPEL DESIGNER adds comprehensive support for the definition, authoring, editing, deploying, testing and debugging of WS-BPEL processes to the well-known integrated development environment (IDE) Eclipse [236]. We use version 1.0.3 [235], which is licensed under the Eclipse Public License.

`BPEL2OWFN` is a tool to translate WS-BPEL processes to open nets [149].

We use version 2.4, which is licensed under the GNU Affero General Public License.

`APACHE ODE` executes WS-BPEL processes by communicating with other (web) services, manipulating data, and handling exceptions. We use Apache ODE version 1.3.6 [233] on an Apache Tomcat server version 8.0.3 [234]; both are licensed under the Apache License version 2.

`SOAPUI` is a web service testing application for service-oriented architectures. Its functionality covers web service inspection, invoking, development, simulation, mocking, functional testing, and load and compliance testing. We use version 4.6.4 [225], which is licensed under the GNU Lesser General Public License.

`PROM` is an extensible plugin-based framework that supports a wide variety of process mining techniques. Replaying an event log on a labeled net can be done with the plug-in “PNetReplayer” that implements the A^* -algorithm [15]. We use version 6.3 [212], which is licensed under the GNU Public License.

The first five tools (Chloe, Delain, Locretia, Service Discovery, and CSVExport) were originally developed to conduct the experiments presented in this thesis. These experiments proved the basic applicability of the results of this thesis (see Sect. 5.4, Sect. 8.4, Sect. 9.4, and Chap. 11) using open nets and event logs of industrial size on a computer with average computing power. Table 30 relates the used tools and their purpose. Existing tools are mainly used to derive the inputs to the proposed algorithms (e.g., open nets and event logs), whereas the originally developed tools primarily implement the algorithms.

purpose	original tools	reused tools
derive open nets		BPEL Designer, BPEL2OWFN
derive event logs	CSVExport	Apache ODE, SoapUI
decide b -conformance (Chap. 5)	Chloe, Delain	
test for b -conformance (Chap. 8)	Chloe, Locretia	ProM
high-quality discovery (Chap. 9)	Chloe, Service Discovery	ProM

Table 30: The tools used in the thesis.

All tools used in this thesis—either developed under the course of this thesis, or existing tools—are free and open source software. This greatly eases the usage of the presented approaches. An integration of the developed tools into industrial modeling tools and acceptance tests are out of scope of this thesis.

12.2 LIMITATIONS AND OPEN QUESTIONS

In this section, we discuss open research questions of our approach and the theoretical and practical limitations of the presented results.

12.2.1 *Incomplete or unsound specifications*

In this thesis, we focused on verifying responsiveness for open systems by means of conformance checking in two different scenarios: In the model-model scenario, we assumed that both the specification and the implementation of an open system are given as formal models, and in the log-model scenario, we assumed that the specification is given as a formal model and we have observed behavior of the unknown implementation in form of an event log. Thus, in both scenarios, we are given a formal specification. The inherent assumption of conformance checking is that this formal specification is *valid* with respect to the system we want to implement: We assume that the specification represents exactly the intended system design, and we then verify that the implementation is an intended system design by checking whether the implementation conforms to the specification.

However, the assumption of a valid specification may not hold for complex specifications. Like other engineering processes, constructing a formal specification is a difficult and error-prone task [139]. Errors in the formal specification $Spec$ can lead to two flaws: (1) A specification may be *incomplete*; that is, it allows for implementations $Impl$ that do not represent the intended system design. (2) A specification may be *unsound*; that is, it disallows implementations $Impl$ that actually do represent intended system design. As conformance checking inherently relies on the validity of the specification, we cannot decide whether $Impl$ is an actual intended design or not by verifying that $Impl$ conforms to $Spec$. However, this is not a specific limitation of conformance checking, but a general limitation of formal methods [110, 241]. Somewhere in system development, the link to the informal reality (i.e., the intended system) has to be made, and the validity of this link (that is, the formal specification represents the intended system design) can only be assumed, not proved. The challenges of incomplete and unsound specifications have been already addressed before in various ways and there exists a rich body of literature, e.g., [133, 261, 209, 66, 108, 65, 139].

A recent approach to circumvent this problem is the notion of quality of an implementation to its specification [19]: Here, the traditional Boolean verification problem (e.g., $Impl$ either conforms to $Spec$, or not) is substituted by a multi-valued problem by introducing a quantitative aspect to verification (e.g., to which extent $Impl$ conforms to $Spec$).

12.2.2 *Measuring quality is subjective*

Given a specification $Spec$ and an event log Log , our discovery approach in Chap. 9 computes an open net $Impl$ that b -conforms to $Spec$ and has high quality with respect to Log and $Spec$. The quality of $Impl$ with respect to Log and $Spec$ is the weighted average over the four quality dimensions fitness, simplicity, precision, and generalization. For measuring simplicity, we compare the size of $inner(Impl)$ with the size of the smallest subsystem G of $MP_b(max_b(Spec))$ that weakly simulates $RG(inner(Impl))$. However, this simplicity metric is not precise enough: There exist cases in which $inner(Impl)$ is even smaller than G due to “unrolled loops” in $MP_b(max_b(Spec))$ (see also

Sect. 9.2). Then, $Impl$ is not distinguished in terms of simplicity from an open net $Impl'$ that b -conforms to $Spec$ and whose inner net's size is equal to G ; both $Impl$ and $Impl'$ have simplicity 1.

One idea to address this limitation is to change the simplicity metric such that the size of $inner(Impl)$ is compared with the smallest subsystem G of $MP_b(max_b(Spec))$ that weakly simulates $RG(inner(Impl))$ and is reduced modulo b -conformance. That way, we could ensure that $inner(Impl)$ (and, therefore, $Impl$) cannot be smaller than the respective subsystem of $MP_b(max_b(Spec))$. However, it is an open question how to do this. Another idea is to consider concurrency in the simplicity metric. To this end, we have to transform the LTS G into a Petri net and somehow compare it to $Impl$. However, transforming G into a Petri net may come at the price of drastically increasing runtimes, even when applying state-of-the-art tools [58].

12.2.3 Abstraction only preserves fitness and simplicity

We showed Chap. 9 that, in general, our genetic discovery algorithm produces better results (i.e., a higher quality of the discovered open net in less time) on the finite abstraction of the search space (i.e., using b -subnets of $Spec$) than on the complete search space (i.e., using arbitrary open nets that b -conform to $Spec$). However, our proposed abstraction technique—the b -subnets of $Spec$ —only preserves fitness and simplicity; the values of the precision and the generalization dimensions may be higher for arbitrary open nets that b -conform to $Spec$. It is an open question how the abstraction technique based on b -subnets can be improved such that it preserves all four quality dimensions, and how a more precise abstraction technique would influence the quality of the discovered open nets.

An idea to circumvent the problem of excluding open nets with high generalization and precision from the search space (by restricting the search space to b -subnets) is to post-process the discovered b -subnet $Impl$ of $Spec$. By Def. 195, $Impl$ may not have the highest precision due to a “furled” loop in its inner net's reachability graph. The idea is to subsequently (i.e., after discovering $Impl$) unroll that loop to increase precision, thereby transforming the b -subnet $Impl$ to an open net $Impl'$ that is no longer a b -subnet of $Spec$ but still b -conforms to $Spec$. This post-processing may increase the precision of $Impl'$ while preserving its fitness, simplicity, and generalization; in other words, the quality of $Impl'$ may be higher than the quality of $Impl$ with respect to $Spec$ and the given event log.

12.3 FUTURE WORK

In this section, we sketch directions for possible extensions of the work presented.

12.3.1 Refined conformance relations

In this thesis, we motivated and fixed responsiveness as a fundamental behavioral correctness criterion for interacting open systems. Once correctness with respect to responsiveness is established for an implementation, one can easily think of additional criteria that should hold: An example is Microsoft's asynchronous event driven programming language P [76], which—in addition to responsiveness—requires that *no message in any message channel is ignored forever*. However, this additional criterion induces a confor-

mance relation that is different from the ones considered in this thesis. Another issue is the minimal requirement *weak termination* (e.g., [181, 162, 44]): Reaching a final state should always be possible. This criterion is very close to the idea of should testing, but it is not clear how to characterize the respective conformance relation (which is a precongruence itself). In contrast, we characterized precongruences related to responsiveness—that is, compositional conformance and compositional b -conformance—with semantical ideas that also worked for should testing. Another idea is to extend responsiveness to also consider communication over ports (e.g., as for web service interfaces defined in WSDL [64]) in the sense that, for every port P , it should always be possible to communicate over P .

In a more general view, we imagine arbitrary correctness criteria that are described as temporal formulae, e.g., in CTL* [210], and which should hold in the composition of two open systems. It is an interesting research question whether the approaches for conformance checking in this thesis can be generalized to deal with behavioral correctness criteria formulated as temporal formulae.

12.3.2 Improved algorithms

In Chap. 5, we showed how to decide whether an open net $Impl$ b -conforms to an open net $Spec$ based on the LTSs $CSD_b(Impl)$ and $CSD_b(Spec)$, or with help of the maximal b -partner $max_b(Spec)$ of $Spec$. Although we showed that computing CSD_b is feasible for open nets of industrial size (see Sect. 5.4), the construction algorithm, in general, suffers from the state-space explosion problem [243]. There exist several effective state-space reduction techniques for verification [243]. It is an interesting research question how these techniques can be employed to speed up the construction of CSD_b or $max_b(Spec)$.

12.3.3 Compositionality in the log-model scenario

In Chap. 8, we presented a passive testing approach for b -conformance: Given a specification $Spec$ and an event log Log that derives from an implementation $Impl$, we can construct an open net $mp_b(max_b(Spec))$ and showed that if Log cannot be replayed on $mp_b(max_b(Spec))$, then $Impl$ does not b -conform to $Spec$. In Chap. 9, we presented a system discovery approach for b -conformance: Given a specification $Spec$ and an event log Log , we showed how to compute an open net $Impl$ that b -conforms to $Spec$ and has high or even highest quality with respect to Log and $Spec$. Thereby, the presented implementation depends on several inputs, among others the LTS $MP_b(max_b(Spec))$ that we can compute from $Spec$. In other words, both approaches in Chap. 8 and Chap. 9 rely on the maximal b -partner $max_b(Spec)$ of $Spec$; this is because $max_b(Spec)$ finitely characterizes all b -conforming open nets of $Spec$ (see Chap. 5). Therefore, both approaches in Chap. 8 and Chap. 9 are currently limited to b -conformance and cannot be applied to compositional b -conformance: It is an open and interesting research question whether there exists a “maximal” b -partner of $Spec$ that finitely characterizes all open nets $Impl$ that compositionally b -conform to $Spec$. Such a b -partner may also serve as an alternative to decide compositional b -conformance, just like the maximal b -partner max_b serves as an alternative decision procedure for b -conformance (see Sect. 5.3).

12.3.4 *Refined discovery*

In Chap. 9, we steer the genetic discovery algorithm by user-given weights to the four quality dimensions fitness, simplicity, precision, and generalization. These four quality dimensions compete with each other and their interplay is of a complex nature, as shown in [53]. Consequently, it is an interesting research question to study the impact of different weights of the quality dimensions on the quality of the discovered b -conforming open net.

Another problem is that there are certain disadvantages to measuring the quality of the discovered b -conforming open net $Impl$ by assigning user weights to the quality dimensions and aggregating them into a single quality measure: For example, determining the weights upfront is difficult if structural changes on $Impl$ have unknown or too complex effects on the value of a single quality dimension. Another disadvantage is that by returning only $Impl$, the user is not provided with any insights in the trade-offs between the quality dimensions. One idea to overcome these problems is to return a Pareto front of n discovered b -conforming open nets $\{Impl_1, \dots, Impl_n\}$ instead of a single open net $Impl$: A Pareto front is a set of *mutually non-dominating* open nets, whereas an open net $Impl_i$ dominates an open net $Impl_j$ (for $i, j \in \mathbb{N}$) if, for all quality dimensions, the quality of $Impl_i$ is equal to or higher than the quality of $Impl_j$, and for one quality dimension, the quality of $Impl_i$ is strictly higher than the quality of $Impl_j$ [245]. Recently, this idea was successfully employed to a discovery algorithm in the area of process mining [54].

12.3.5 *Introducing additional aspects*

In this thesis, we entirely focused on the communication protocol of an open system and abstracted from other aspects such as the location of the open system, the underlying middleware, instantiation of the open system and the correlation of messages, the content of messages, or nonfunctional properties (e.g., time). Especially the abstract concept of time is crucial for many real-world systems; hence, there exist numerous approaches to incorporate time for example in workflow systems [35], web services [84], or any kind of protocol [211]. These aspects are not considered during conformance checking, and their integration into our conformance checking approach would broaden the applicability of our results.

BIBLIOGRAPHY

- [1] van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. *Journal of circuits, systems, and computers* **8**(01), 21–66 (1998) (Cited on pages 37, 166, and 224.)
- [2] van der Aalst, W.M.P.: *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer (2011) (Cited on pages 10, 14, 174, 195, 224, 225, and 246.)
- [3] van der Aalst, W.M.P.: Service Mining: Using Process Mining to Discover, Check, and Improve Service Behavior. *IEEE Transactions on Services Computing* (99), 1 (2012). doi: 10.1109/TSC.2012.25 (Cited on page 226.)
- [4] van der Aalst, W.M.P.: Configurable Services for Local Governments (CoSeLoG), project description available at <http://www.win.tue.nl/coselog/> (2013). Last accessed June 16, 2014 (Cited on page 187.)
- [5] van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying History on Process Models for Conformance Checking and Performance Analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **2**(2), 182–192 (2012). doi: 10.1002/widm.1045 (Cited on pages 174, 176, 183, 195, 196, 201, 202, and 224.)
- [6] van der Aalst, W.M.P., Adriansyah, A., de Medeiros, A.K.A., Arcieri, F., Baier, T., Blicke, T., Bose, J.C., Brand, P., Brandtjen, R., Buijs, J., Burattin, A., Carmona, J., Castellanos, M., Claes, J., Cook, J., Costantini, N., Curbera, F., Damiani, E., de Leoni, M., Delias, P., van Dongen, B.F., Dumas, M., Dustdar, S., Fahland, D., Ferreira, D., Gaaloul, W., Geffen, F., Goel, S., Günther, C., Guzzo, A., Harmon, P., Hofstede, A., Hoogland, J., Ingvaldsen, J., Kato, K., Kuhn, R., Kumar, A., Rosa, M., Maggi, F., Malerba, D., Mans, R., Manuel, A., McCreesh, M., Mello, P., Mendling, J., Montali, M., Motahari-Nezhad, H., Muehlen, M., Munoz-Gama, J., Pontieri, L., Ribeiro, J., Rozinat, A., Seguel Pérez, H., Seguel Pérez, R., Sepúlveda, M., Sinur, J., Soffer, P., Song, M., Sperduti, A., Stilo, G., Stoel, C., Swenson, K., Talamo, M., Tan, W., Turner, C., Vanthienen, J., Varvaressos, G., Verbeek, E., Verdonk, M., Vigo, R., Wang, J., Weber, B., Weidlich, M., Weijters, T., Wen, L., Westergaard, M., Wynn, M.: *Process Mining Manifesto*. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) *Lecture Notes in Business Information Processing* 99, pp. 169–194. Springer Berlin Heidelberg (2012) (Cited on pages 183, 224, and 226.)
- [7] van der Aalst, W.M.P., Basten, T.: Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science* **270**(1-2), 125–203 (2002). doi: 10.1016/S0304-3975(00)00321-2 (Cited on page 166.)
- [8] van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: *Workflow Mining: A Survey of Issues and Approaches*. *Data & Knowledge Engineering* **47**(2), 237–267 (2003) (Cited on pages 10 and 171.)

- [9] van der Aalst, W.M.P., Dumas, M., Ouyang, C., Rozinat, A., Verbeek, E.: Conformance Checking of Service Behavior. *ACM Transactions on Internet Technology* **8**(3), 1–30 (2008). doi: 10.1145/1361186.1361189 (Cited on pages 10, 198, 224, and 225.)
- [10] van der Aalst, W.M.P., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: From Public Views to Private Views – Correctness-by-Design for Services. In: Dumas, M., Heckel, R. (eds.) *Lecture Notes in Computer Science* 4937, pp. 139–153. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). doi: 10.1007/978-3-540-79230-7_10 (Cited on page 164.)
- [11] van der Aalst, W.M.P., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: Multiparty Contracts: Agreeing and Implementing Interorganizational Processes. *The Computer Journal* **53**(1), 90–106 (2009). doi: 10.1093/comjnl/bxn064 (Cited on pages 4, 138, 162, 164, 165, and 166.)
- [12] van der Aalst, W.M.P., de Medeiros, A.K.A.: Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance. *Electronic Notes in Theoretical Computer Science* **121**, 3–21 (2005). doi: 10.1016/j.entcs.2004.10.013 (Cited on pages 10 and 224.)
- [13] van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering* **16**(9), 1128–1142 (2004). doi: 10.1109/TKDE.2004.47 (Cited on page 225.)
- [14] Acciai, L., Boreale, M.: Responsiveness in Process Calculi. *Theoretical Computer Science* **409**(1), 59–93 (2008). doi: 10.1016/j.tcs.2008.08.017 (Cited on page 52.)
- [15] Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance Checking Using Cost-Based Fitness Analysis. In: *EDOC '11: Proceedings of the 2011 IEEE 15th International Enterprise Distributed Object Computing Conference*. IEEE Computer Society (2011) (Cited on pages 10, 174, 176, 183, 187, 224, 244, and 252.)
- [16] Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.P.: Alignment Based Precision Checking. In: Rosa, M., Soffer, P. (eds.) *Lecture Notes in Business Information Processing* 132, pp. 137–149. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). doi: 10.1007/978-3-642-36285-9_15 (Cited on pages 195, 199, and 224.)
- [17] Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless Sensor Networks: A Survey. *Computer networks* **38**(4), 393–422 (2002) (Cited on page 5.)
- [18] de Alfaro, L., Henzinger, T.A.: Interface Automata. *ACM SIGSOFT Software Engineering Notes* **26**(5), 109–120 (2001). doi: 10.1145/503209.503226 (Cited on pages 37, 107, and 164.)
- [19] Almagor, S., Boker, U., Kupferman, O.: Formalizing and Reasoning about Quality. In: *ICALP'13: Proceedings of the 40th international conference on Automata, Languages, and Programming*. Springer-Verlag (2013) (Cited on page 253.)

- [20] Alur, R., Etessami, K., Yannakakis, M.: Inference of Message Sequence Charts. *Software Engineering, IEEE Transactions on* **29**(7), 623–633 (2003). doi: 10.1109/TSE.2003.1214326 (Cited on page 37.)
- [21] Apple Inc.: MacBook Air brochure, available at <http://www.apple.com/macbook-air/specs.html> (2013). Last accessed June 16, 2014 (Cited on pages 136, 185, 214, and 237.)
- [22] Arbab, F.: Computing and Interaction. In: Goldin, D., Smolka, S.A., Wegner, P. (eds.) *Interactive Computation*, pp. 9–23. Springer Berlin Heidelberg (2006). doi: 10.1007/3-540-34874-3_2 (Cited on page 5.)
- [23] Asbagh, M.J., Abolhassani, H.: Web Service Usage Mining: Mining for Executable Sequences **7**, 266–271 (2007) (Cited on page 226.)
- [24] Back, R.J.R., Wright, J.: Refinement Calculus, Part I: Sequential Non-deterministic Programs. In: Bakker, J.W., Roever, W.P., Rozenberg, G. (eds.) *Lecture Notes in Computer Science* 430, pp. 42–66. Springer Berlin Heidelberg (1990) (Cited on pages 5 and 162.)
- [25] Badouel, E., Darondeau, P.: Theory of Regions. In: Reisig, W., Rozenberg, G. (eds.) *Lecture Notes in Computer Science* 1491, pp. 529–586. Springer Berlin Heidelberg, Berlin, Heidelberg (1998). doi: 10.1007/3-540-65306-6_22 (Cited on pages 9 and 38.)
- [26] Baeten, J., Weijland, W.P.: *Process Algebra*, Volume 18 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press Cambridge, UK (1990) (Cited on page 36.)
- [27] Baeten, J.C.: A Brief History of Process Algebra. *Theoretical Computer Science* **335**(2), 131–146 (2005) (Cited on page 36.)
- [28] Baier, C., Katoen, J.P.: *Principles of Model Checking*, vol. 26202649. MIT press Cambridge (2008) (Cited on pages 5, 9, 18, and 38.)
- [29] Banavar, G., Chandra, T., Strom, R., Sturman, D.: A Case for Message Oriented Middleware. In: Jayanti, P. (ed.) *Lecture Notes in Computer Science* 1693, pp. 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg (1999). doi: 10.1007/3-540-48169-9_1 (Cited on page 6.)
- [30] Basten, T., van der Aalst, W.M.P.: Inheritance of Behavior. *The Journal of Logic and Algebraic Programming* **47**(2), 47–145 (2001). doi: 10.1016/S1567-8326(00)00004-7 (Cited on page 166.)
- [31] Basu, S., Bultan, T., Ouederni, M.: Synchronizability for Verification of Asynchronously Communicating Systems. In: Kuncak, V., Rybalchenko, A. (eds.) *Lecture Notes in Computer Science* 7148, pp. 56–71. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). doi: 10.1007/978-3-642-27940-9_5 (Cited on page 36.)
- [32] Basu, S., Casati, F., Daniel, F.: Toward Web Service Dependency Discovery for SOA Management. In: *Services Computing, 2008. SCC '08. IEEE International Conference on*, pp. 422–429 (2008). doi: 10.1109/SCC.2008.45 (Cited on page 226.)
- [33] Berardi, D., Calvanese, D., Giacomo, G., Lenzerini, M., Mecella, M.: Automatic Composition of E-services That Export Their Behavior. In:

- Orlowska, M., Weerawarana, S., Papazoglou, M., Yang, J. (eds.) *Lecture Notes in Computer Science 2910*, pp. 43–58. Springer Berlin Heidelberg, Berlin, Heidelberg (2003). doi: 10.1007/978-3-540-24593-3_4 (Cited on page 37.)
- [34] Bergstra, J.A., Klop, J.W., Tucker, J.V.: *Process Algebra with Asynchronous Communication Mechanisms*. *Seminar on Concurrency* pp. 76–95 (1985) (Cited on page 36.)
- [35] Bettini, C., Wang, X.S., Jajodia, S.: *Temporal Reasoning in Workflow Systems*. *Distributed and Parallel Databases* **11**(3), 269–306 (2002) (Cited on page 256.)
- [36] Bloom, B., Istrail, S., Meyer, A.R.: *Bisimulation Can’t Be Traced*. *Journal of the ACM* **42**(1), 232–268 (1995). doi: 10.1145/200836.200876 (Cited on page 161.)
- [37] Bochmann, G., Sunshine, C.: *Formal Methods in Communication Protocol Design*. *IEEE Transactions on Communications* **28**(4), 624–631 (1980). doi: 10.1109/TCOM.1980.1094685 (Cited on pages 4, 6, 36, and 162.)
- [38] Boender, C.G.E., Kan, A.H.G.R.: *A Bayesian Analysis of the Number of Cells of a Multinomial Distribution*. *Journal of the Royal Statistical Society. Series D (The Statistician)* **32**(1/2), 240–248 (1983) (Cited on page 201.)
- [39] de Boer, F.S., Klop, J.W., Palamidessi, C.: *Asynchronous Communication in Process Algebra*. In: *Logic in Computer Science, 1992. LICS ’92., Proceedings of the Seventh Annual IEEE Symposium on*, pp. 137–147 (1992). doi: 10.1109/LICS.1992.185528 (Cited on page 36.)
- [40] Bonchi, F., Brogi, A., Corfini, S., Gadducci, F.: *A Behavioural Congruence for Web Services*. In: *Arbab, F., Sirjani, M. (eds.) Lecture Notes in Computer Science 4767*, pp. 240–256–256. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). doi: 10.1007/978-3-540-75698-9_16 (Cited on page 166.)
- [41] Boudol, G.: *Asynchrony and the Pi-calculus*. Tech. Rep. RR-1702, MEIJE - INRIA Sophia Antipolis (1992) (Cited on page 37.)
- [42] Brand, D., Zafiropulo, P.: *On Communicating Finite-State Machines*. *Journal of the ACM* **30**(2), 323–342 (1983). doi: 10.1145/322374.322380 (Cited on page 37.)
- [43] Bravetti, M., Zavattaro, G.: *Contract Based Multi-party Service Composition*. In: *Arbab, F., Sirjani, M. (eds.) Lecture Notes in Computer Science 4767*, pp. 207–222. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). doi: 10.1007/978-3-540-75698-9_14 (Cited on pages 92 and 163.)
- [44] Bravetti, M., Zavattaro, G.: *A Foundational Theory of Contracts for Multi-party Service Composition*. *Fundamenta Informaticae* **89**(4), 451–478 (2008) (Cited on pages 4, 49, 162, 163, 165, and 255.)
- [45] Bravetti, M., Zavattaro, G.: *Contract-Based Discovery and Composition of Web Services*. In: *Bernardo, M., Padovani, L., Zavattaro, G. (eds.) Lecture Notes in Computer Science 5569*, pp. 261–295. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). doi: 10.1007/978-3-642-01918-0_7 (Cited on page 163.)

- [46] Bravetti, M., Zavattaro, G.: Contract Compliance and Choreography Conformance in the Presence of Message Queues. In: *Web Services and Formal Methods*, pp. 37–54. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). doi: 10.1007/978-3-642-01364-5_3 (Cited on pages 37, 163, and 167.)
- [47] Brinksma, E.: A Theory for the Derivation of Tests. University of Twente, Department of Computer Science (1988) (Cited on page 163.)
- [48] Brinksma, E., Rensink, A., Vogler, W.: Fair Testing. In: Lee, I., Smolka, S. (eds.) *Lecture Notes in Computer Science* 962, pp. 313–327. Springer Berlin Heidelberg, Berlin, Heidelberg (1995). doi: 10.1007/3-540-60218-6_23 (Cited on pages 69, 161, and 167.)
- [49] Brinksma, E., Tretmans, J.: Testing Transition Systems: An Annotated Bibliography. In: Cassez, F., Jard, C., Rozoy, B., Ryan, M. (eds.) *Lecture Notes in Computer Science*, pp. 187–195. Springer Berlin Heidelberg (2001) (Cited on pages 18, 172, and 225.)
- [50] Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A Theory of Communicating Sequential Processes. *Journal of the ACM* 31(3), 560–599 (1984). doi: 10.1145/828.833 (Cited on pages 64, 65, and 166.)
- [51] Broy, M.: Compositional Refinement of Interactive Systems. *Journal of the ACM* 44(6), 850–891 (1997) (Cited on pages 7 and 12.)
- [52] Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: A Genetic Algorithm for Discovering Process Trees. *Evolutionary Computation (CEC)*, 2012 IEEE Congress on pp. 1–8 (2012). doi: 10.1109/CEC.2012.6256458 (Cited on pages 209, 225, and 226.)
- [53] Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In: Meersman, R., Panetto, H., Dillon, T., Rinderle-Ma, S., Dadam, P., Zhou, X., Pearson, S., Ferscha, A., Bergamaschi, S., Cruz, I. (eds.) *Lecture Notes in Computer Science* 7565, pp. 305–322. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). doi: 10.1007/978-3-642-33606-5_19 (Cited on pages 186, 187, 209, 214, 226, and 256.)
- [54] Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Discovering and Navigating a Collection of Process Models using Multiple Quality Dimensions. to appear in *Workshop proceedings of BPM 2013* (2013) (Cited on page 256.)
- [55] Bultan, T.: Analyzing Interactions of Asynchronously Communicating Software Components. In: Beyer, D., Boreale, M. (eds.) *Lecture Notes in Computer Science* 7892, pp. 1–4. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). doi: 10.1007/978-3-642-38592-6_1 (Cited on page 6.)
- [56] Bultan, T., Fu, X., Hull, R., Su, J.: Conversation Specification: A new Approach to Design and Analysis of E-service Composition. In: *WWW '03: Proceedings of the 12th international conference on World Wide Web*. ACM (2003). doi: 10.1145/775152.775210 (Cited on page 36.)

- [57] Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M., Patrizi, F.: Automatic Service Composition and Synthesis: The Roman Model. *IEEE Data Eng. Bull.* **31**(3), 18–22 (2008) (Cited on page 37.)
- [58] Carmona, J., Cortadella, J., Kishinevsky, M.: Genet: A Tool for the Synthesis and Mining of Petri Nets. In: 2009 Ninth International Conference on Application of Concurrency to System Design (ACSD), pp. 181–185. IEEE (2009). doi: 10.1109/ACSD.2009.6 (Cited on page 254.)
- [59] Castagna, G., Gesbert, N., Padovani, L.: A Theory of Contracts for Web Services. *ACM Transactions on Programming Languages and Systems* **31**(5), 1–61 (2009). doi: 10.1145/1538917.1538920 (Cited on pages 163 and 164.)
- [60] Charette, R.N.: This Car runs on Code. *IEEE Spectrum* **46**(3), 3 (2009) (Cited on pages 4 and 5.)
- [61] Chen, T., Chilton, C., Jonsson, B., Kwiatkowska, M.: A Compositional Specification Theory for Component Behaviours. In: Seidl, H. (ed.) *Lecture Notes in Computer Science* 7211, pp. 148–168. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). doi: 10.1007/978-3-642-28869-2_8 (Cited on page 164.)
- [62] Chilton, C., Jonsson, B., Kwiatkowska, M.: An Algebraic Theory of Interface Automata. Tech. Rep. CS-RR-13-02 (2013) (Cited on page 164.)
- [63] Chinosi, M., Trombetta, A.: BPMN: An Introduction to the Standard. *Computer Standards & Interfaces* **34**(1), 124–134 (2012). doi: 10.1016/j.csi.2011.06.002 (Cited on pages 38, 229, and 230.)
- [64] Christensen, E., Curbera, F., Meredith, G.: *Web Services Description Language (WSDL) 1.1* (2001) (Cited on pages 234 and 255.)
- [65] Cimatti, A., Roveri, M., Schuppan, V., Tchaltsev, A.: Diagnostic Information for Realizability. In: Logozzo, F., Peled, D., Zuck, L. (eds.) *Lecture Notes in Computer Science* 4905, pp. 52–67. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). doi: 10.1007/978-3-540-78163-9_9 (Cited on page 253.)
- [66] Claessen, K.: A Coverage Analysis for Safety Property Lists. *Formal Methods in Computer Aided Design, 2007. FMCAD '07* pp. 139–145 (2007). doi: 10.1109/FAMCAD.2007.32 (Cited on page 253.)
- [67] Clarke, E.M., Grumberg, O., Peled, D.A.: *Model Checking*. MIT press Cambridge (1999) (Cited on pages 5, 9, and 190.)
- [68] Cleaveland, R., Sokolsky, O.: Equivalence and Preorder Checking for Finite-State Systems. *Handbook of Process Algebra* pp. 391–424 (2001) (Cited on pages 4 and 162.)
- [69] Comuzzi, M., Vonk, J., Grefen, P.: Measures and Mechanisms for Process Monitoring in Evolving Business Networks. *Data & Knowledge Engineering* **71**(1), 1–28 (2012). doi: 10.1016/j.datak.2011.07.004 (Cited on page 225.)
- [70] Cook, J.E., He, C., Ma, C.: Measuring Behavioral Correspondence to a Timed Concurrent Model. In: *Software Maintenance, 2001. Proceedings. IEEE International Conference on*, pp. 332–341 (2001). doi: 10.1109/ICSM.2001.972746 (Cited on page 224.)

- [71] Cook, J.E., Wolf, A.L.: Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology* **8**(2), 147–176 (1999). doi: 10.1145/304399.304401 (Cited on page 224.)
- [72] De Nicola, R., Hennessy, M.C.B.: Testing Equivalences for Processes. *Theoretical Computer Science* **34**(1-2), 83–133 (1984). doi: 10.1016/0304-3975(84)90113-0 (Cited on pages 4, 161, 162, and 163.)
- [73] Debian: SLOCCount Web for Debian Lenny, available at <http://debian-counting.libresoft.es/lenny/index.php?menu=Statistics> (2009). Last accessed June 16, 2014 (Cited on page 4.)
- [74] Decker, G., Barros, A., Kraft, F., Lohmann, N.: Non-desynchronizable Service Choreographies. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) *Lecture Notes in Computer Science* 5364, pp. 331–346. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). doi: 10.1007/978-3-540-89652-4_26 (Cited on page 36.)
- [75] DeRemer, F., Kron, H.H.: Programming-in-the-large versus Programming-in-the-small. *Software Engineering, IEEE Transactions on* (2), 80–86 (1976) (Cited on pages 5 and 233.)
- [76] Desai, A., Gupta, V., Jackson, E., Qadeer, S., Rajamani, S., Zufferey, D.: P: Safe Asynchronous Event-Driven Programming. In: *PLDI '13: Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*. ACM (2013). doi: 10.1145/2491956.2462184 (Cited on pages 8, 52, and 254.)
- [77] Desel, J., Reisig, W.: The Synthesis Problem of Petri Nets. *Acta Informatica* **33**(4), 297–315 (1996) (Cited on pages 9 and 38.)
- [78] Dewender, J., Müller, R.: Deciding Language Inclusions (Delain) tool, available at <http://service-technology.org/delain> (2014). Last accessed June 16, 2014 (Cited on pages 12, 14, 136, 139, 241, and 251.)
- [79] Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and Analysis of Business Process Models in BPMN. *Information and Software technology* **50**(12), 1281–1294 (2008) (Cited on pages 38 and 232.)
- [80] Dijkstra, E.W.: Guarded Commands, Nondeterminacy and Formal Derivation of Programs. *Communications of the ACM* **18**(8), 453–457 (1975) (Cited on page 162.)
- [81] Dill, D.L.: Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits. Ph.D. thesis, MIT press Cambridge (1989) (Cited on pages 4, 107, 162, 163, and 164.)
- [82] Dingle, N.J., Knottenbelt, W.J., Suto, T.: PIPE2: A Tool for the Performance Evaluation of Generalised Stochastic Petri Nets. *SIGMETRICS Performance Evaluation Review* **36**(4) (2009). doi: 10.1145/1530873.1530881 (Cited on page 193.)
- [83] Duffy, D.A.: Principles of Automated Theorem Proving. John Wiley & Sons, Inc. (1991) (Cited on page 5.)
- [84] Duske, K., Müller, R.: A Survey on Approaches for Timed Services. In: Schönberger, A., Kopp, O., Lohmann, N. (eds.) *Proceedings of the*

- 4th Central-European Workshop on Services and their Composition (ZEUS 2012), *CEUR Workshop Proceedings*, vol. 847, pp. 1–8. CEUR-WS.org, Bamberg, Germany (2012) (Cited on page 256.)
- [85] Duske, K., Müller, R., Prüfer, R., Stöhr, D.: A BPMN Model of the Charité Stroke Treatment Process. Technical report, Humboldt-Universität zu Berlin (2014). To be published (Cited on pages 229, 230, and 231.)
- [86] Dustdar, S., Gombotz, R.: Discovering Web Service Workflows using Web Services Interaction Mining. *International Journal of Business Process Integration and Management* **1**(4), 256–266 (2006) (Cited on page 226.)
- [87] Dyer, D.W.: Watchmaker Framework for Evolutionary Computation, available at <http://watchmaker.uncommons.org> (2013). Last accessed June 16, 2014 (Cited on page 211.)
- [88] Ebert, C., Jones, C.: Embedded Software: Facts, Figures, and Future. *Computer* **42**(4), 42–52 (2009). doi: 10.1109/MC.2009.118 (Cited on page 3.)
- [89] Eiben, A.E., Smith, J.E.: *Introduction to Evolutionary Computing*, vol. 2. Springer Berlin (2010) (Cited on pages 209 and 216.)
- [90] Ellson, J., Gansner, E., Koutsofios, L., North, S., Woodhull, G.: Graphviz—Open Source Graph Drawing Tools. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) *Lecture Notes in Computer Science* 2265, pp. 483–484. Springer Berlin Heidelberg (2002). doi: 10.1007/3-540-45848-4_57 (Cited on page 137.)
- [91] Ellson, J., Gansner, E.R., Koutsofios, E., North, S.C., Woodhull, G.: Graphviz and Dynagraph — Static and Dynamic Graph Drawing Tools. In: Jünger, M., Mutzel, P. (eds.) *Mathematics and Visualization*, pp. 127–148. Springer Berlin Heidelberg, Berlin, Heidelberg (2004). doi: 10.1007/978-3-642-18638-7_6 (Cited on page 137.)
- [92] Esparza, J.: Decidability and Complexity of Petri Net Problems—An Introduction. *Lectures on Petri Nets I: Basic Models* pp. 374–428 (1998) (Cited on page 167.)
- [93] Fassbender, K., Walter, S., Liu, Y., Muehlhauser, F., Ragoschke, A., Kuehl, S., Mielke, O.: "Mobile Stroke Unit" for Hyperacute Stroke Treatment. *Stroke* **34**(6), e44–e44 (2003). doi: 10.1161/01.STR.0000075573.22885.3B (Cited on page 237.)
- [94] Fenton, N.E., Neil, M.: Software Metrics: Successes, Failures and New Directions. *Journal of Systems and Software* **47**(2-3), 149–157 (1999). doi: 10.1016/S0164-1212(99)00035-7 (Cited on page 4.)
- [95] Floyd, R.W.: Algorithm 97: Shortest Path. *Communications of the ACM* **5**(6), 345 (1962). doi: 10.1145/367766.368168 (Cited on pages 118 and 132.)
- [96] Floyd, R.W.: Assigning Meanings to Programs. *Mathematical aspects of computer science* **19**(19-32), 1 (1967) (Cited on page 162.)

- [97] Fournet, C., Hoare, T., Rajamani, S., Rehof, J.: Stuck-Free Conformance. In: Alur, R., Peled, D. (eds.) *Lecture Notes in Computer Science* 3114, pp. 242–254. Springer Berlin Heidelberg (2004) (Cited on pages 37 and 163.)
- [98] Fromm, J.: *The Emergence of Complexity*. Kassel university press Kassel (2004) (Cited on page 5.)
- [99] Fu, X., Bultan, T., Su, J.: Synchronizability of Conversations among Web Services. *Software Engineering, IEEE Transactions on* 31(12), 1042–1055 (2005). doi: 10.1109/TSE.2005.141 (Cited on page 36.)
- [100] Gamboni, M., Ravara, A.: Responsive Choice in Mobile Processes. In: Wirsing, M., Hofmann, M., Rauschmayer, A. (eds.) *Lecture Notes in Computer Science* 6084, pp. 135–152. Springer Berlin Heidelberg (2010) (Cited on page 52.)
- [101] Gansner, E., North, S.: The Specification of the DOT Language, available at <http://www.graphviz.org/content/dot-language> (2013). Last accessed June 16, 2014 (Cited on page 137.)
- [102] Gershenfeld, N.A.: *The Nature of Mathematical Modeling*. Cambridge University Press Cambridge, UK (1999) (Cited on page 193.)
- [103] Glabbeek, R.J.: The Coarsest Precongruences Respecting Safety and Liveness Properties. In: Calude, C., Sassone, V. (eds.) *IFIP Advances in Information and Communication Technology*, pp. 32–52. Springer Berlin Heidelberg (2010) (Cited on page 86.)
- [104] Glabbeek, R.J.v.: The Linear Time - Branching Time Spectrum. In: Baeten, J.C.M., Klop, J.W. (eds.) *Lecture Notes in Computer Science* 458, pp. 278–297. Springer Berlin Heidelberg (1990). doi: 10.1007/BFb0039066 (Cited on page 161.)
- [105] Glabbeek, R.J.v.: The Linear Time-Branching Time Spectrum II pp. 66–81 (1993) (Cited on page 161.)
- [106] Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust Process Discovery with Artificial Negative Events. *The Journal of Machine Learning Research* 10, 1305–1340 (2009) (Cited on page 224.)
- [107] Grimson, J., Grimson, W., Hasselbring, W.: The SI Challenge in Health Care. *Communications of the ACM* 43(6), 48–55 (2000) (Cited on page 5.)
- [108] Grosse, D., Kuhne, U., Drechsler, R.: Estimating Functional Coverage in Bounded Model Checking. In: *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07*, pp. 1–6. EDA Consortium (2007). doi: 10.1109/DATE.2007.364454 (Cited on page 253.)
- [109] Hacke, W.: *Neurologie*. Springer Medizin Verlag (2010) (Cited on page 230.)
- [110] Hall, A.: Seven Myths of Formal Methods. *Software, IEEE* 7(5), 11–19 (1990). doi: 10.1109/52.57887 (Cited on page 253.)
- [111] Harrer, S., Lenhard, J., Wirtz, G.: BPEL Conformance in Open Source Engines. In: *Service-Oriented Computing and Applications (SOCA), 2012 5th IEEE International Conference on*, pp. 1–8 (2012). doi: 10.1109/SOCA.2012.6449467 (Cited on page 242.)

- [112] Harrer, S., Lenhard, J., Wirtz, G.: Open Source versus Proprietary Software in Service-Orientation: The Case of BPEL Engines. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) *Lecture Notes in Computer Science* 8274, pp. 99–113. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). doi: 10.1007/978-3-642-45005-1_8 (Cited on page 242.)
- [113] Hedderich, J., Sachs, L.: *Angewandte Statistik: Methodensammlung mit R*. Springer (2012) (Cited on pages 217 and 218.)
- [114] van Hee, K.M., Mooij, A.J., Sidorova, N., van der Werf, J.M.E.M.: Soundness-Preserving Refinements of Service Compositions. In: Bravetti, M., Bultan, T. (eds.) *Lecture Notes in Computer Science* 6551, pp. 131–145–145. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). doi: 10.1007/978-3-642-19589-1_9 (Cited on page 165.)
- [115] Heiden, S., Müller, R.: Characterizing Languages of Open Net Environments (Chloe) tool, available at <http://service-technology.org/chloe> (2013). Last accessed June 16, 2014 (Cited on pages 12, 14, 135, 139, 183, 191, 212, 241, 244, 245, and 251.)
- [116] Heiden, S., Müller, R.: Log Creator I Am (Locretia) tool, available at <http://service-technology.org/locretia> (2013). Last accessed June 16, 2014 (Cited on pages 183, 185, 187, 191, 214, and 251.)
- [117] Herrero-Perez, D., Martinez-Barbera, H.: Modeling Distributed Transportation Systems Composed of Flexible Automated Guided Vehicles in Flexible Manufacturing Systems. *Industrial Informatics, IEEE Transactions on* 6(2), 166–180 (2010). doi: 10.1109/TII.2009.2038691 (Cited on page 5.)
- [118] Hoare, C.A.R.: An Axiomatic Basis for Computer Programming. *Communications of the ACM* 12(10), 576–580 (1969). doi: 10.1145/363235.363259 (Cited on page 162.)
- [119] Hoare, C.A.R.: Communicating Sequential Processes. *Communications of the ACM* 21(8), 666–677 (1978) (Cited on pages 4, 36, and 162.)
- [120] Hoare, C.A.R., Misra, J., Leavens, G.T., Shankar, N.: The Verified Software Initiative. *ACM Comput Surv* 41(4), 1–8 (2009). doi: 10.1145/1592434.1592439 (Cited on page 4.)
- [121] Honda, K., Tokoro, M.: An Object Calculus for Asynchronous Communication. In: America, P. (ed.) *Lecture Notes in Computer Science* 512, pp. 133–147. Springer Berlin Heidelberg, Berlin/Heidelberg (1991). doi: 10.1007/BFb0057019 (Cited on page 37.)
- [122] Honda, K., Tokoro, M.: On Asynchronous Communication Semantics. In: Tokoro, M., Nierstrasz, O., Wegner, P. (eds.) *Lecture Notes in Computer Science* 612, pp. 21–51. Springer Berlin Heidelberg, Berlin, Heidelberg (1992). doi: 10.1007/3-540-55613-3_2 (Cited on page 37.)
- [123] Hopcroft, J.E., Motwani, J.E., Ullman, J.D.: *Introduction to Automata Theory, Languages, and Computation*. 3rd edn. Prentice Hall (2006) (Cited on pages 19, 100, and 151.)
- [124] Hull, R., Benedikt, M., Christophides, V., Su, J.: E-services. In: the twenty-second ACM SIGMOD-SIGACT-SIGART symposium, pp. 1–14. ACM Press, New York, New York, USA (2003). doi: 10.1145/773153.773154 (Cited on page 37.)

- [125] IBM Systems and Technology Group: IBM BladeCenter brochure, available at <http://public.dhe.ibm.com/common/ssi/ecm/en/blb03002usen/BLB03002USEN.PDF> (2013). Last accessed June 16, 2014 (Cited on page 136.)
- [126] Jančar, P.: Undecidability of Bisimilarity for Petri Nets and some Related Problems. *Theoretical Computer Science* **148**(2), 281–301 (1995) (Cited on pages 76, 77, 79, 83, and 167.)
- [127] Jobs, S.: Apple WWDC Keynote, available at <http://www.engadget.com/2006/08/07/live-from-wwdc-2006-steve-jobs-keynote/> (2006). Last accessed June 16, 2014 (Cited on page 4.)
- [128] Johnson, V.E.: Revised Standards for Statistical Evidence. *Proceedings of the National Academy of Sciences* **110**(48), 19,313–19,317 (2013). doi: 10.1073/pnas.1313476110/-/DCSupplemental (Cited on page 218.)
- [129] Jonsson, B.: Compositional Specification and Verification of Distributed Systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)* **16**(2), 259–303 (1994). doi: 10.1145/174662.174665 (Cited on page 164.)
- [130] Jordan, D., Evdemon, J., Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y.: Web Services Business Process Execution Language Version 2.0. OASIS Standard **11**, 1–264 (2007) (Cited on pages 8, 38, 136, 184, 224, 229, 232, 233, 234, and 242.)
- [131] Kartson, D., Balbo, G., Donatelli, S., Franceschinis, G., Conte, G.: Modelling with Generalized Stochastic Petri Nets. John Wiley & Sons, Inc. (1994) (Cited on page 193.)
- [132] Kaschner, K.: Conformance Testing for Asynchronously Communicating Services. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H. (eds.) *Lecture Notes in Computer Science* 7084, pp. 108–124. Springer Berlin Heidelberg (2011). doi: 10.1007/978-3-642-25535-9_8 (Cited on page 225.)
- [133] Katz, S., Grumberg, O., Geist, D.: "Have I Written Enough Properties?" - A Method of Comparison Between Specification and Implementation. In: Pierre, L., Kropf, T. (eds.) *Lecture Notes in Computer Science* 1703, pp. 280–297. Springer Berlin Heidelberg, Berlin, Heidelberg (1999). doi: 10.1007/3-540-48153-2_21 (Cited on page 253.)
- [134] Kazhamiakin, R., Pistore, M., Santuari, L.: Analysis of Communication Models in Web Service Compositions. In: the 15th international conference, pp. 267–276. ACM Press, New York, New York, USA (2006). doi: 10.1145/1135777.1135819 (Cited on pages 6 and 36.)
- [135] Kernighan, B.W., Ritchie, D.M., Ejeclint, P.: *The C Programming Language* (1988) (Cited on pages 9 and 38.)
- [136] Kindler, E.: A Compositional Partial Order Semantics for Petri Net Components. In: Azéma, P., Balbo, G. (eds.) *Lecture Notes in Computer Science* 1248, pp. 235–252. Springer Berlin Heidelberg, Berlin, Heidelberg (1997). doi: 10.1007/3-540-63139-9_39 (Cited on page 37.)
- [137] Kobayashi, N.: A Type System for Lock-Free Processes. *Information and Computation* **177**(2), 122–159 (2002). doi: 10.1006/inco.2002.3171 (Cited on page 52.)

- [138] Kokol, P., Podgorelec, V., Cardoso, A.I., Dion, F.: Assessing the State of the Software Process Development using the Chaos Theory. *ACM SIGSOFT Software Engineering Notes* **25**(3), 41–43 (2000) (Cited on page 4.)
- [139] Könighofer, R., Hofferek, G., Bloem, R.: Debugging Formal Specifications: A Practical Approach using Model-based Diagnosis and Counterstrategies. *International Journal on Software Tools for Technology Transfer* **15**(5-6), 563–583–583 (2013). doi: 10.1007/s10009-011-0221-y (Cited on page 253.)
- [140] Kumar, V., Abbas, A.K., Fausto, N., Aster, J.C.: *Robbins and Cotran Pathologic Basis of Disease, Professional Edition: Expert Consult-Online*. Elsevier Health Sciences (2009) (Cited on page 229.)
- [141] Laneve, C., Padovani, L.: The Must Preorder Revisited. In: Caires, L., Vasconcelos, V. (eds.) *Lecture Notes in Computer Science* 4703, pp. 212–225. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). doi: 10.1007/978-3-540-74407-8_15 (Cited on pages 4, 162, 163, and 166.)
- [142] Larsen, K.G.: Modal Specifications. In: Sifakis, J. (ed.) *Lecture Notes in Computer Science* 407, pp. 232–246. Springer Berlin Heidelberg, Berlin, Heidelberg (1990). doi: 10.1007/3-540-52148-8_19 (Cited on page 164.)
- [143] Leduc, G.: A Framework Based on Implementation Relations for Implementing LOTOS Specifications. *Computer networks and ISDN systems* **25**(1), 23–41 (1992) (Cited on pages 4 and 162.)
- [144] Lee, D., Yannakakis, M.: Principles and Methods of Testing Finite State Machines-A Survey. *Proceedings of the IEEE* **84**(8), 1090–1123 (1996). doi: 10.1109/5.533956 (Cited on page 225.)
- [145] Lee, E.A.: *Embedded Software*. pp. 55–95. Elsevier (2002). doi: 10.1016/S0065-2458(02)80004-3 (Cited on page 3.)
- [146] Lee, K.W., Ko, B.J., Calo, S.: Adaptive Server Selection for Large Scale Interactive Online Games. *Computer networks* **49**(1), 84–102 (2005). doi: 10.1016/j.comnet.2005.04.006 (Cited on page 5.)
- [147] Lenic, M., Zorman, M., Povalej, P., Kokol, P.: Alternative Measurement of Software Artifacts. In: *Computational Cybernetics, 2004. ICC 2004. Second IEEE International Conference on*, pp. 231–235 (2004). doi: 10.1109/ICCCYB.2004.1437715 (Cited on page 4.)
- [148] Ponce de León, H., Haar, S., Longuet, D.: Conformance Relations for Labeled Event Structures. In: Brucker, A., Julliand, J. (eds.) *Lecture Notes in Computer Science* 7305, pp. 83–98. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). doi: 10.1007/978-3-642-30473-6_8 (Cited on page 225.)
- [149] Lohmann, N.: A Feature-Complete Petri Net Semantics for WS-BPEL 2.0. In: Dumas, M., Heckel, R. (eds.) *Lecture Notes in Computer Science* 4937, pp. 77–91. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). doi: 10.1007/978-3-540-79230-7_6 (Cited on pages 8, 38, 136, 185, 237, and 252.)

- [150] Lohmann, N.: Communication Models for Services. Proceedings of the 2nd Central-European Workshop on Services and their Composition pp. 9–16 (2010) (Cited on pages 6 and 36.)
- [151] Lohmann, N.: Correctness of Services and their Composition. Ph.D. thesis, Technische Universiteit Eindhoven (2010) (Cited on page 38.)
- [152] Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing Interacting WS-BPEL Processes using Flexible Model Generation. *Data & Knowledge Engineering* **64**(1), 38–54 (2008). doi: 10.1016/j.datak.2007.06.006 (Cited on page 164.)
- [153] Lohmann, N., Massuthe, P., Wolf, K.: Operating Guidelines for Finite-State Services. In: Kleijn, J., Yakovlev, A. (eds.) *Lecture Notes in Computer Science* 4546, pp. 321–341. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). doi: 10.1007/978-3-540-73094-1_20 (Cited on pages 8, 24, 27, 37, 38, 138, 165, and 211.)
- [154] Lohmann, N., Verbeek, E., Dijkman, R.: Petri Net Transformations for Business Processes – A Survey. In: Jensen, K., van der Aalst, W.M.P. (eds.) *Lecture Notes in Computer Science* 5460, pp. 46–63. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). doi: 10.1007/978-3-642-00899-3_3 (Cited on page 38.)
- [155] Lohmann, N., Wolf, K.: How to Implement a Theory of Correctness in the Area of Business Processes and Services. In: Hull, R., Mendling, J., Tai, S. (eds.) *Lecture Notes in Computer Science* 6336, pp. 61–77. Springer Berlin Heidelberg (2010) (Cited on pages 136 and 251.)
- [156] Lohmann, N., Wolf, K.: Compact Representations and Efficient Algorithms for Operating Guidelines. *Fundamenta Informaticae* **108**(1), 43–62 (2011) (Cited on pages 165, 167, and 211.)
- [157] Lorenz, R., Mauser, S., Juhas, G.: How to Synthesize Nets from Languages - A Survey. In: *Simulation Conference, 2007 Winter*, pp. 637–647 (2007). doi: 10.1109/WSC.2007.4419657 (Cited on pages 9 and 38.)
- [158] Loveland, D.W.: *Automated Theorem Proving: A Logical Basis*. *Fundamental Studies in Computer Science* (1978) (Cited on page 5.)
- [159] Lynch, N.A.: *Distributed Algorithms*. Morgan Kaufmann (1996) (Cited on page 37.)
- [160] Lynch, N.A., Tuttle, M.R.: *An Introduction to Input/Output Automata* (1988) (Cited on page 164.)
- [161] Mahoney, M.S.: The Roots of Software Engineering. *CWI Quarterly* **3**(4), 325–334 (1990) (Cited on page 5.)
- [162] Malik, R., Streader, D., Reeves, S.: Conflicts and Fair Testing. *International Journal of Foundations of Computer Science* **17**(04), 797–813 (2013). doi: 10.1142/S012905410600411X (Cited on pages 49, 92, 139, 165, and 255.)
- [163] Martens, A.: *Verteilte Geschäftsprozesse*. Ph.D. thesis, Humboldt-Universität zu Berlin (2004) (Cited on pages 37 and 166.)

- [164] Martens, A.: Analyzing Web Service Based Business Processes. In: Cerioli, M. (ed.) *Lecture Notes in Computer Science* 3442, pp. 19–33–33. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). doi: 10.1007/978-3-540-31984-9_3 (Cited on page 166.)
- [165] Massuthe, P.: *Operating Guidelines for Services*. Ph.D. thesis, Humboldt University of Berlin (2009) (Cited on pages 37 and 38.)
- [166] Massuthe, P., Reisig, W., Wolf, K.: An Operating Guideline Approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics* pp. 35–43 (2005) (Cited on pages 37, 38, and 49.)
- [167] Massuthe, P., Serebrenik, A., Sidorova, N., Wolf, K.: Can I Find a Partner? Undecidability of Partner Existence for Open Nets. *Information Processing Letters* **108**(6), 374–378 (2008). doi: 10.1016/j.ipl.2008.07.006 (Cited on pages 37 and 49.)
- [168] Mathers, C.D., Boerma, T., Ma Fat, D.: Global and Regional Causes of Death. *British Medical Bulletin* **92**(1), 7–32 (2009). doi: 10.1093/bmb/ldp028 (Cited on page 230.)
- [169] McIlroy, M.D.: Mass-produced Software Components. *Proceedings of the 1st International Conference on Software Engineering* pp. 88–98 (1968) (Cited on page 5.)
- [170] Mealy, G.H.: A Method for Synthesizing Sequential Circuits. *Bell System Technical Journal* **34**(5), 1045–1079 (1955) (Cited on pages 37 and 225.)
- [171] de Medeiros, A.K.A., Weijters, A.J.M.M., van der Aalst, W.M.P.: Genetic Process Mining: An Experimental Evaluation. *Data Mining and Knowledge Discovery* **14**(2), 245–304 (2007). doi: 10.1007/s10618-006-0061-7 (Cited on pages 209, 225, and 226.)
- [172] Menasce, D.: MOM vs. RPC: Communication Models for Distributed Applications. *Internet Computing, IEEE* **9**(2), 90–93 (2005). doi: 10.1109/MIC.2005.42 (Cited on pages 6 and 36.)
- [173] Mendling, J., Neumann, G., van der Aalst, W.M.P.: Understanding the Occurrence of Errors in Process Models Based on Metrics. In: Meersman, R., Tari, Z. (eds.) *Lecture Notes in Computer Science* 4803, pp. 113–130. Springer Berlin Heidelberg (2007) (Cited on page 198.)
- [174] Merlin, P.M.: Specification and Validation of Protocols. *Communications, IEEE Transactions on* **27**(11), 1671–1680 (1979). doi: 10.1109/TCOM.1979.1094323 (Cited on pages 6 and 36.)
- [175] Meyer, B.: *Object-Oriented Software Construction*. Prentice Hall New York (1988) (Cited on page 5.)
- [176] Microsoft: A History of Windows, available at <http://windows.microsoft.com/en-US/windows/history> (2001). Last accessed June 16, 2014 (Cited on page 4.)
- [177] Milner, R.: *Communication and Concurrency*. Prentice Hall (1989) (Cited on pages 20, 36, 162, and 163.)
- [178] Minsky, M.L.: *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc (1967) (Cited on pages 75, 76, 78, and 167.)

- [179] Mooij, A., Parnjai, J., Stahl, C., Voorhoeve, M.: Constructing Replaceable Services Using Operating Guidelines and Maximal Controllers. In: Bravetti, M., Bultan, T. (eds.) *Lecture Notes in Computer Science* 6551, pp. 116–130–130. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). doi: 10.1007/978-3-642-19589-1_8 (Cited on pages 138, 139, 140, and 165.)
- [180] Mooij, A., Voorhoeve, M.: Proof Techniques for Adapter Generation. In: Bruni, R., Wolf, K. (eds.) *Lecture Notes in Computer Science* 5387, pp. 207–223–223. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). doi: 10.1007/978-3-642-01364-5_13 (Cited on pages 137, 139, and 165.)
- [181] Mooij, A.J., Stahl, C., Voorhoeve, M.: Relating Fair Testing and Accordance for Service Replaceability. *The Journal of Logic and Algebraic Programming* 79(3-5), 233–244 (2010). doi: 10.1016/j.jlap.2009.12.001 (Cited on pages 49, 92, 165, and 255.)
- [182] Morgan, C.: The Specification Statement. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 10(3), 403–419 (1988) (Cited on page 162.)
- [183] Morris, F.L., Jones, C.B.: An Early Program Proof by Alan Turing. *IEEE Annals of the History of Computing* 6(2), 139–143 (1984) (Cited on page 4.)
- [184] Morris, J.M.: A Theoretical Basis for Stepwise Refinement and the Programming Calculus. *Science of Computer programming* 9(3), 287–306 (1987) (Cited on page 162.)
- [185] Motahari-Nezhad, H.R., Saint-Paul, R., Benatallah, B., Casati, F.: Deriving Protocol Models from Imperfect Service Conversation Logs. *IEEE Transactions on Knowledge and Data Engineering* 20(12), 1683–1698 (2008) (Cited on page 226.)
- [186] Motahari-Nezhad, H.R., Saint-Paul, R., Casati, F., Benatallah, B.: Event Correlation for Process Discovery from Web Service Interaction Logs. *The VLDB Journal* 20(3), 417–444 (2010). doi: 10.1007/s00778-010-0203-9 (Cited on pages 225 and 226.)
- [187] Müller, R.: On the Notion of Deadlocks in Open Nets. In: Schwarick, M., Heiner, M. (eds.) *Proceedings of the 17th German Workshop on Algorithms and Tools for Petri Nets (AWPN 2010), CEUR Workshop Proceedings*, vol. 643, pp. 130–135. CEUR-WS.org, Cottbus, Germany (2010) (Cited on pages 52 and 166.)
- [188] Müller, R.: Service Discovery ProM plugin, available at <http://www.promtools.org/prom6/> (2013). Last accessed June 16, 2014 (Cited on pages 14, 211, and 251.)
- [189] Müller, R.: CSVExport Event Listener for Apache ODE, available at <https://bitbucket.org/richardmueller/csvexport> (2014). Last accessed June 16, 2014 (Cited on pages 242 and 251.)
- [190] Müller, R., van der Aalst, W.M.P., Stahl, C.: Conformance Checking of Services Using the Best Matching Private View. In: Beek, M.H., Lohmann, N. (eds.) *Web Services and Formal Methods, Lecture*

- Notes in Computer Science*, vol. 7843, pp. 49–68. Springer Berlin Heidelberg (2013). doi: 10.1007/978-3-642-38230-7_4 (Cited on pages 171 and 178.)
- [191] Müller, R., Stahl, C., van der Aalst, W.M.P., Westergaard, M.: Service Discovery from Observed Behavior while Guaranteeing Deadlock Freedom in Collaborations. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) *Service-Oriented Computing, Lecture Notes in Computer Science*, vol. 8274, pp. 358–373. Springer Berlin Heidelberg (2013). doi: 10.1007/978-3-642-45005-1_25 (Cited on page 193.)
- [192] Müller, R., Stahl, C., Vogler, W.: Deciding Conformance for Bounded Responsiveness (2014). Invited to a special issue of *Science of Computer Programming*, submitted on February 15, 2014 (Cited on page 85.)
- [193] Müller, R., Stahl, C., Vogler, W.: Undecidability of Accordance for Open Systems with Unbounded Message Queues (2014). Accepted for publication in *Information Processing Letters* on April 26, 2014 (Cited on pages 55 and 167.)
- [194] Murata, T.: Petri Nets: Properties, Analysis and Applications. In: *Proceedings of the IEEE*, pp. 541–580 (1989). doi: 10.1109/5.24143 (Cited on page 37.)
- [195] Musaraj, K., Yoshida, T., Daniel, F., Hacid, M.S., Casati, F., Benatallah, B.: Message Correlation and Web Service Protocol Mining from Inaccurate Logs. In: *Web Services (ICWS), 2010 IEEE International Conference on*, pp. 259–266 (2010). doi: 10.1109/ICWS.2010.104 (Cited on page 226.)
- [196] Natarajan, V., Cleaveland, R.: Divergence and Fair Testing. In: Fülöp, Z., Gécseg, F. (eds.) *Lecture Notes in Computer Science* 944, pp. 648–659. Springer Berlin Heidelberg, Berlin, Heidelberg (1995). doi: 10.1007/3-540-60084-1_112 (Cited on pages 69, 161, and 167.)
- [197] National Institute of Standards and Technology: Software Errors Cost U.S. Economy \$59.5 Billion Annually, available at <http://www.nist.gov/director/planning/upload/report02-3.pdf> (2002). Last accessed June 16, 2014 (Cited on page 3.)
- [198] Nielson, F., Nielson, H.R., Hankin, C.: *Principles of Program Analysis*. Springer (1999) (Cited on pages 9 and 38.)
- [199] Nolte, C.H., Malzahn, U., Kühnle, Y., Ploner, C.J., Müller-Nordhorn, J., Möckel, M.: Improvement of Door-to-Imaging Time in Acute Stroke Patients by Implementation of an All-Points Alarm. *Journal of Stroke and Cerebrovascular Diseases* **22**(2), 149–153 (2013). doi: 10.1016/j.jstrokecerebrovasdis.2011.07.004 (Cited on pages 229 and 230.)
- [200] Padovani, L.: From Lock Freedom to Progress using Session Types. *PLACES 2013* p. 2 (2013) (Cited on page 52.)
- [201] Papazoglou, M.: *Web Services: Principles and Technology*. Pearson Education (2008) (Cited on pages 5, 8, 136, 184, and 226.)
- [202] Park, D.: Concurrency and Automata on Infinite Sequences. In: Deussen, P. (ed.) *Lecture Notes in Computer Science* 104, pp. 167–183.

- Springer Berlin Heidelberg, Berlin/Heidelberg (1981). doi: 10.1007/BFb0017309 (Cited on pages 20 and 161.)
- [203] Parnjai, J.: Behavioral Service Substitution: Analysis and Synthesis. Ph.D. thesis, Berlin, Humboldt Universität zu Berlin, Diss., 2013 (2013) (Cited on pages 38, 138, 139, 140, and 165.)
- [204] Parnjai, J.: Maxis tool, available at <http://svn.gna.org/viewcvs/service-tech/trunk/maxis/> (2013). Last accessed June 16, 2014 (Cited on page 139.)
- [205] Parnjai, J., Stahl, C., Wolf, K.: A Finite Representation of all Substitutable Services and its Applications. ZEUS (2009) (Cited on pages 140 and 165.)
- [206] Parrow, J.: An Introduction to the π -Calculus. In: Bergstra, Ponse, Smolka (eds.) Handbook of Process Algebra, pp. 1–72 (2013) (Cited on page 37.)
- [207] Petrenko, A.: Fault Model-Driven Test Derivation from Finite State Models: Annotated Bibliography. In: Cassez, F., Jard, C., Rozoy, B., Ryan, M. (eds.) Lecture Notes in Computer Science 2067, pp. 196–205–205. Springer Berlin Heidelberg, Berlin, Heidelberg (2001). doi: 10.1007/3-540-45510-8_10 (Cited on page 225.)
- [208] PHP: Hypertext Preprocessor 5.4.20, available at <http://www.php.net> (2013). Last accessed June 16, 2014 (Cited on pages 9 and 38.)
- [209] Pill, I., Semprini, S., Cavada, R., Rovers, M., Bloem, R., Cimatti, A.: Formal Analysis of Hardware Requirements. In: Design Automation Conference, 2006 43rd ACM/IEEE, pp. 821–826 (2006). doi: 10.1109/DAC.2006.229231 (Cited on page 253.)
- [210] Pnueli, A.: The Temporal Logic of Programs. Foundations of Computer Science, 1977., 18th Annual Symposium on pp. 46–57 (1977). doi: 10.1109/SFCS.1977.32 (Cited on page 255.)
- [211] Ponge, J., Benatallah, B., Casati, F., Toumani, F.: Analysis and Applications of Timed Service Protocols. ACM Transactions on Software Engineering and Methodology (TOSEM) 19(4), 11 (2010). doi: 10.1145/1734229.1734230 (Cited on page 256.)
- [212] Process Mining Group, Eindhoven Technical University: ProM Process Mining Workbench, available at <http://www.promtools.org/prom6/> (2013). Last accessed June 16, 2014 (Cited on pages 15, 183, 191, 211, 242, 244, 251, and 252.)
- [213] Rajamani, S.K., Rehof, J.: Conformance Checking for Models of Asynchronous Message Passing Software. In: Computer Aided Verification, pp. 166–179. Springer Berlin Heidelberg, Berlin, Heidelberg (2002). doi: 10.1007/3-540-45657-0_13 (Cited on pages 162 and 163.)
- [214] Reed, J.N., Roscoe, A.W., Sinclair, J.E.: Responsiveness and Stable Revivals. Formal Aspects of Computing 19(3), 303–319–319 (2007). doi: 10.1007/s00165-007-0032-9 (Cited on page 52.)
- [215] Reisig, W.: The Synthesis Problem. In: Jensen, K., van der Aalst, W.M.P., Balbo, G., Koutny, M., Wolf, K. (eds.) Lecture Notes in Computer Science 7480, pp. 300–313. Springer Berlin Heidelberg (2013) (Cited on pages 9 and 38.)

- [216] Reisig, W.: *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Springer (2013) (Cited on pages 8, 21, 22, 37, and 49.)
- [217] Rensink, A., Vogler, W.: Fair Testing. *Information and Computation* **205**(2), 125–198 (2007). doi: 10.1016/j.ic.2006.06.002 (Cited on pages 68, 69, 70, 82, 150, 157, 161, 164, 166, and 167.)
- [218] Rozinat, A., van der Aalst, W.M.P.: Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In: Bussler, C., Haller, A. (eds.) *Lecture Notes in Computer Science* 3812, pp. 163–176. Springer Berlin Heidelberg, Berlin, Heidelberg (2006). doi: 10.1007/11678564_15 (Cited on pages 10 and 224.)
- [219] Rozinat, A., van der Aalst, W.M.P.: Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems* **33**(1), 64–95 (2008). doi: 10.1016/j.is.2007.07.001 (Cited on pages 10, 172, 183, 195, 224, and 225.)
- [220] Rozinat, A., Mans, R.S., Song, M., van der Aalst, W.M.P.: Discovering Simulation Models. *Information Systems* **34**(3), 305–327 (2009). doi: 10.1016/j.is.2008.09.002 (Cited on page 193.)
- [221] Rushby, J.M.: Automated Formal Methods Enter the Mainstream. *J. UCS* **13**(5), 650–660 (2007) (Cited on page 4.)
- [222] Sharygina, N., Chaki, S., Clarke, E., Sinha, N.: Dynamic Component Substitutability Analysis. In: Fitzgerald, J., Hayes, I., Tarlecki, A. (eds.) *Lecture Notes in Computer Science* 3582, pp. 512–528. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). doi: 10.1007/11526841_34 (Cited on pages 9 and 38.)
- [223] Sharygina, N., Kröning, D.: Model Checking with Abstraction for Web Services. In: Baresi, L., Nitto, E. (eds.) *Test and Analysis of Web Services*, pp. 121–145–145. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). doi: 10.1007/978-3-540-72912-9_5 (Cited on pages 9 and 38.)
- [224] Sipser, M.: *Introduction to the Theory of Computation*, vol. 2. Thomson Course Technology Boston (2006) (Cited on pages 18, 19, 37, 100, and 151.)
- [225] SmartBear Software: SoapUI, available at <http://www.soapui.org> (2013). Last accessed June 16, 2014 (Cited on pages 243, 245, and 252.)
- [226] Stahl, C., Massuthe, P., Bretschneider, J.: Deciding Substitutability of Services with Operating Guidelines. In: Jensen, K., van der Aalst, W.M.P. (eds.) *Lecture Notes in Computer Science* 5460, pp. 172–191. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). doi: 10.1007/978-3-642-00899-3_10 (Cited on pages 4, 37, 38, 127, 138, 162, 164, 165, and 166.)
- [227] Stahl, C., Vogler, W.: A Trace-Based View on Operating Guidelines. In: Hofmann, M. (ed.) *Lecture Notes in Computer Science* 6604, pp. 411–425. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). doi: 10.1007/978-3-642-19805-2_28 (Cited on pages 88 and 166.)
- [228] Stahl, C., Vogler, W.: A Trace-Based Service Semantics Guaranteeing Deadlock Freedom. *Acta Informatica* **49**(2), 69–103 (2012). doi: 10.1007/s00236-012-0151-5 (Cited on pages 34, 57, 83, 88, 166, and 167.)

- [229] Stroustrup, B.: after a quote in http://www.stroustrup.com/Programming/1-2_programming.ppt (2003). Last accessed June 16, 2014 (Cited on page 3.)
- [230] Szyperski, C.: *Component Software: Beyond Object-oriented Programming*. Pearson Education (2002) (Cited on page 5.)
- [231] Tan, Q.M.: *On Conformance Testing of Systems Communicating by Rendezvous*. Ph.D. thesis, Universite de Montreal (1998) (Cited on page 225.)
- [232] Tang, R., Zou, Y.: An Approach for Mining Web Service Composition Patterns from Execution Logs. *Web Systems Evolution (WSE)*, 2010 12th IEEE International Symposium on pp. 53–62 (2010). doi: 10.1109/WSE.2010.5623568 (Cited on page 226.)
- [233] The Apache Software Foundation: Apache ODE (Orchestration Director Engine) software, available at <http://ode.apache.org> (2013). Last accessed June 16, 2014 (Cited on pages 242, 245, 251, and 252.)
- [234] The Apache Software Foundation: Apache Tomcat software, available at <http://tomcat.apache.org> (2013). Last accessed June 16, 2014 (Cited on pages 242 and 252.)
- [235] The Eclipse Foundation: Eclipse BPEL Designer Project, available at <http://www.eclipse.org/bpel/> (2013). Last accessed June 16, 2014 (Cited on pages 234 and 251.)
- [236] The Eclipse Foundation: Eclipse IDE, available at <https://www.eclipse.org> (2013). Last accessed June 16, 2014 (Cited on pages 234 and 251.)
- [237] Thurley, M.: sharpSAT – Counting Models with Advanced Component Caching and Implicit BCP. In: Biere, A., Gomes, C. (eds.) *Lecture Notes in Computer Science 4121*, pp. 424–429. Springer Berlin Heidelberg (2006) (Cited on page 209.)
- [238] Tretmans, J.: *Conformance Testing with Labelled Transition Systems: Implementation Relations and Test Generation*. *Computer networks and ISDN systems* 29(1), 49–79 (1996) (Cited on pages 4 and 162.)
- [239] Tretmans, J.: *Testing Concurrent Systems: A Formal Approach*. In: Baeten, J.M., Mauw, S. (eds.) *Lecture Notes in Computer Science 1664*, pp. 46–65. Springer Berlin Heidelberg, Berlin, Heidelberg (1999). doi: 10.1007/3-540-48320-9_6 (Cited on pages 172 and 225.)
- [240] Tretmans, J., Belinfante, A.: *Automatic Testing with Formal Methods*. *EuroSTAR’99: 7th European Int Conference on Software Testing, Analysis & Review* pp. 8–12 (1999) (Cited on page 4.)
- [241] Tretmans, J., Wijbrans, K., Chaudron, M.: *Software Engineering with Formal Methods: The Development of a Storm Surge Barrier Control System Revisiting Seven Myths of Formal Methods*. *Formal Methods in System Design* 19(2), 195–215–215 (2001) (Cited on page 253.)
- [242] Valiant, L.G.: *The Complexity of Enumeration and Reliability Problems*. *SIAM Journal on Computing* 8(3), 410–421 (1979) (Cited on page 209.)

- [243] Valmari, A.: The State Explosion Problem. In: Reisig, W., Rozenberg, G. (eds.) *Lecture Notes in Computer Science* 1491, pp. 429–528. Springer Berlin Heidelberg, Berlin, Heidelberg (1998). doi: 10.1007/3-540-65306-6_21 (Cited on page 255.)
- [244] Valmari, A.: External Behaviour of Systems of State Machines with Variables. In: Jensen, K., van der Aalst, W.M.P., Balbo, G., Koutny, M., Wolf, K. (eds.) *Lecture Notes in Computer Science* 7480, pp. 255–299. Springer Berlin Heidelberg (2013) (Cited on page 36.)
- [245] Van Veldhuizen, D.A., Lamont, G.B.: Evolutionary Computation and Convergence to a Pareto Front. Morgan Kaufmann pp. 221–228 (1998) (Cited on page 256.)
- [246] Vogler, W.: Modular Construction and Partial Order Semantics of Petri Nets, vol. 625. *Lecture notes in computer science* edn. Springer-Verlag Berlin (1992) (Cited on pages 8, 24, 28, 30, 33, 37, 38, 65, 67, 74, 82, 83, 88, 165, and 167.)
- [247] Vogler, W., Stahl, C., Müller, R.: A Trace-Based Semantics for Responsiveness. In: 12th International Conference on Application of Concurrency to System Design (ACSD 2012), pp. 42–51. IEEE (2012). doi: 10.1109/ACSD.2012.10 (Cited on pages 52 and 55.)
- [248] Vogler, W., Stahl, C., Müller, R.: Trace- and Failure-Based Semantics for Bounded Responsiveness. In: Canal, C., Villari, M. (eds.) *Advances in Service-Oriented and Cloud Computing, Communications in Computer and Information Science*, vol. 393, pp. 129–143. Springer Berlin Heidelberg (2013). doi: 10.1007/978-3-642-45364-9_12 (Cited on pages 85 and 141.)
- [249] Vogler, W., Stahl, C., Müller, R.: Trace- and Failure-Based Semantics for Responsiveness (2014). Accepted for publication in *Acta Informatica* on April 5, 2014 (Cited on pages 52, 55, 72, 85, and 141.)
- [250] Voorhoeve, M., Mauw, S.: Impossible Futures and Determinism. *Information Processing Letters* 80(1), 51–58 (2001). doi: 10.1016/S0020-0190(01)00217-4 (Cited on pages 65 and 167.)
- [251] Weber, J.E., Ebinger, M., Rozanski, M., Waldschmidt, C., Wendt, M., Winter, B., Kellner, P., Baumann, A., Fiebach, J.B., Villringer, K.: Pre-hospital Thrombolysis in Acute Stroke Results of the PHANTOM-S Pilot Study. *Neurology* 80(2), 163–168 (2013) (Cited on page 237.)
- [252] Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.F.: *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR (2005) (Cited on page 233.)
- [253] Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering Workflow Models from Event-based Data using Little Thumb. *Integrated Computer-Aided Engineering* 10(2), 151–162 (2003) (Cited on page 225.)
- [254] van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process Discovery using Integer Linear Programming. *Fundamenta Informaticae* 94(3), 387–412 (2009) (Cited on page 225.)

- [255] Wiels, V., Delmas, R., Doose, D., Garoche, P.L., Cazin, J., Durrieu, G.: Formal Verification of Critical Aerospace Software. *AerospaceLab Journal* (4) (2012) (Cited on page 4.)
- [256] Wirth, N.: Program Development by Stepwise Refinement. *Communications of the ACM* **14**(4), 221–227 (1971) (Cited on pages 4, 5, and 162.)
- [257] Wolf, K.: Generating Petri Net State Spaces. In: Kleijn, J., Yakovlev, A. (eds.) *Lecture Notes in Computer Science* 4546, pp. 29–42. Springer Berlin Heidelberg (2007) (Cited on page 135.)
- [258] Wolf, K.: Does My Service Have Partners? *Transactions on Petri Nets and Other Models of Concurrency II* (2009) (Cited on pages 49, 51, 52, 111, 127, 165, and 167.)
- [259] Wolf, K., Stahl, C., Weinberg, D., Ott, J., Danitz, R.: Guaranteeing Weak Termination in Service Discovery. *Fundamenta Informaticae* **108**(1-2) (2011) (Cited on page 49.)
- [260] Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.: Formal Methods. *ACM Comput Surv* **41**(4), 1–36 (2009). doi: 10.1145/1592434-1592436 (Cited on page 4.)
- [261] Yoshiura, N.: Finding the Causes of Unrealizability of Reactive System Formal Specifications. In: *Software Engineering and Formal Methods, 2004. SEFM 2004. Proceedings of the Second International Conference on*, pp. 34–43 (2004). doi: 10.1109/SEFM.2004.1347501 (Cited on page 253.)
- [262] Zhivich, M., Cunningham, R.K.: The Real Cost of Software Errors. *Security & Privacy, IEEE* **7**(2), 87–90 (2009) (Cited on page 3.)

INDEX

- \emptyset , 17
- $[\]$, 17
- \uplus , 17
- $\sqsubseteq_{b, \text{conf}}$, 47
- $\sqsubseteq_{b, \text{conf}}^c$, 48
- $\sqsubseteq_{\mathcal{F}_{b, \text{fin}}^+}$, 146
- $\sqsubseteq_{\text{conf}}$, 42
- $\sqsubseteq_{\text{conf}}^c$, 43
- $\sqsubseteq_{\text{fin}}^+$, 68
- $AA(\text{Log}, N)$, 199
- accordance relation, 4
- ACP, 36
- action-equivalence, 19, 23
- agreement, 34
- alignment, 174
- alphabet, 18, 19, 23
- Apache ODE, 242
- asynchronous communication, 36
- automaton, 19
- b -bounded deadlock freedom, 49
- b -bounded $\mathcal{F}_{\text{fin}}^+$ -semantics, 142
- b -bounded *stopdead*-semantics, 86
- b -bounded weak termination, 49
- b -boundedness, 22
- b -conformance, 47
- b -coverable *stopdead*-semantics, 92
- b -partner, 46
- b -responsiveness, 45
- b -subnet, 203
- b -uncoverable trace, 92
- $\text{Bags}(A)$, 17
- basic activity, 233
- basic net, 76
- bound_b -violator, 86
- behavior, 22
- behavioral correctness, 8
- $\text{BEH}_b(N)$, 154
- bisimulation, 161
- bisimulation relation, 20
- Boolean annotation, 211
- $\text{bound}_b(N)$, 86
- bound_b -maximal b -partner, 127
- BPEL2OWFN, 136
- BPMN, 230
- branching states, 76
- $\text{BSD}_b(N)$, 100
- CCS, 36
- Chloe, 135
- closed net, 25
- communication, 6
- communication protocol, 6
- complement, 18
- completed trace preorder, 161
- components, 5
- composable, 26, 30
- composition, 26
- compositional b -conformance, 48
- compositional conformance, 43
- compositional conformance relation, 6
- compositionality, 5
- conformance, 42
- conformance checking, 4
- conformance relation, 4
- conformation relation, 4
- continuation, 18
- correctness criterion, 4
- correctness-by-construction, 4
- correctness-by-verification, 4
- cost, 176
- cost-function, 176
- cost-minimal alignment, 176
- counter machine, 76
- crossover operation, 210
- $\text{CSD}_b(N)$, 108
- CSP, 36
- CSVExport, 242
- dc -pattern, 77
- $\text{dead}(N)$, 57
- dead except for inputs, 57
- dead-trace , 57
- $\text{dead}_b(N)$, 86
- dead_b -maximal b -partner, 127
- deadlock freedom, 22, 49
- decidability, 103, 116, 157
- Delain, 136
- denotational semantics, 12
- determinism, 19
- discovery, 10, 194
- dominated, 68
- \mathcal{E} , 173
- Eclipse BPEL Designer, 234
- elitism, 210

- empty state, 101
- empty word, 18
- enabled transition, 22
- environment, 5, 28
- equivalence, 17
- error state, 100
- event log, 9, 174
- event trace, 174
- events, 173
- $\mathcal{F}_{b,fin}^+(N)$, 142
- $\mathcal{F}_{b,fin}^+$ -refinement, 146
- $\mathcal{F}_{fin}^+(N)$, 65
- \mathcal{F}_{fin}^+ -refinement, 68
- \mathcal{F}_{fin}^+ -semantics, 65
- failure, 65
- fair testing, 161
- fin-refusal set, 65
- final marking, 21
- $finbound_b$ -violator, 142
- $finbound_b(N)$, 142
- finiteness, 19
- fintree failure, 65
- firing, 22
- fitness, 196
- flow relation, 21
- formal methods, 4
- formal model, 8
- functions, 5
- generalization, 202
- genetic algorithm, 209
- halting problem, 78
- hiding, 30
- I/O automaton, 37
- implementation, 4
- implementation relation, 4
- individual, 209
- initial marking, 21
- initial state, 18
- inner net, 27
- input action, 19, 23
- input place, 24
- interaction, 5
- interface, 6
- interface automaton, 37
- interface place, 25
- interface-equivalent, 25
- internal action, 19, 23
- internal place, 25
- Jančar-Patterns, 76
- $\kappa(\gamma)$, 176
- label, 19
- labeled net, 23
- labeled transition relation, 18
- labeled transition system, 18
- labeling function, 23
- language, 18, 19, 23
- $L_b(N)$, 86
- L_b -maximal b -partner, 127
- linear time - branching time spectrum, 161
- Locretia, 183
- log move, 174
- log-model scenario, 9
- LoLa, 135
- marked place, 22
- marking, 21
- matching, 124
- matching relation, 124
- maximal b -partner, 127
- model move, 174
- model-model scenario, 9
- modularization, 5
- modules, 5
- monitoring, 172
- most-permissive b -partner, 127
- move, 174
- $MP_b(N)$, 108
- $mp_b(N)$, 112
- multiset, 17
- must testing, 161
- mutation operation, 210
- \mathbb{N} , 17
- \mathbb{N}^+ , 17
- net, 21
- objects, 5
- $\mathcal{O}_N(w)$, 177
- open net, 24
- open system, 5
- oracle function, 177
- output action, 19, 23
- output place, 24
- $\mathcal{P}(A)$, 17
- parallel composition, 30
 - with hiding, 31
- Parikh vector, 23
- partial order, 17
- partner, 41
- parts, 5
- passive testing, 172

- place, 21
- population, 209
- postset, 22
- precision, 201
- precongruence, 17
- prefix, 18
- prefix closure, 18
- preorder, 17
- preorder relation, 4
- preset, 22
- procedures, 5
- process mining, 10
- productive state, 150
- productive subautomaton, 152
- projection, 18
- ProM, 183

- Q_{\emptyset} , 101
- qualitative correctness, 8
- quality, 202

- reachability graph, 22
- reachable marking, 22
- reachable state, 19
- ready simulation, 161
- refinement relation, 4
- refusal set, 65
- remainder, 18
- replacement operation, 210
- replay environment, 178
- responsiveness, 41
- restricted language, 19
- run, 19, 22

- saturation conditions, 70, 147
- semantical correctness, 8
- sequentially communicating, 25
- ServiceDiscovery ProM plugin, 211
- services, 5
- set, 17
- should testing, 161
- silent move, 174
- simplicity, 198
- simulation relation, 20
- SoapUI, 243
- specification, 4
- state, 18
- state labeling function, 19
- $stop(N)$, 57
- stop except for inputs, 57
- $stop$ -trace, 57
- $stop_b(N)$, 86
- $stop_b$ -maximal b -partner, 127
- $stopdead$ -semantics, 57
- strict $bound_b$ -violator, 86
- stroke unit, 230
- strong agreement, 34
- structured activity, 233
- subcontract relation, 4
- subsystem, 19
- suffix closure, 18
- synchronous communication, 36
- synchronous move, 174
- syntactical correctness, 7

- τ -freeness, 19, 23
- termination criteria, 210
- testing, 4, 179
- trace, 19, 23
- trace preorder, 161
- $trace(\gamma)$, 175
- transition, 21
- tree failure, 65

- \mathbb{U} , 100
- $udead_b(N)$, 92
- $uL_b(N)$, 92
- $uncov_b(N)$, 92
- undecidability, 82, 83
- $ustop_b(N)$, 92

- verification, 4
- viewpoint, 177

- weak bisimulation relation, 20
- weak simulation relation, 20
- weak termination, 22, 49
- word, 18
- workflow net, 37
- WS-BPEL, 232

ACKNOWLEDGEMENTS

I would not have finished this thesis without the help of various people. In the following I am going to express my gratitude to them.

First of all, I want to thank Wolfgang Reisig for introducing me to the scientific world. Without him I would have never thought about pursuing a Ph.D. at all. He taught me that presentation does matter and gave me the freedom to pursue my own research interests. Second, I thank Wil van der Aalst for his supervision within the binational Ph.D. program. He taught me to always look at the bigger picture and to strive for results that are actually usable in practice. I am grateful for his invaluable feedback, his encouraging stimulation, and the opportunity to finish my thesis while living one year in Eindhoven.

I would have not written this thesis without the support of my co-promotor Christian Stahl. His research led me to my thesis topic and his encouragement made me never lose sight of it. Thank you very much for all the discussions, for all the constructive critique, for the warm welcome in Eindhoven, and your time invested in reading several drafts of this thesis as well as in the co-supervision as a whole. I also thank Walter Vogler for the detailed theoretical discussions and the numerous collaborations. Most of the results of this thesis are joint work with him and Christian Stahl. It was a real pleasure and I learned a lot!

I thank Johann-Christoph Freytag, Jan Friso Groote, and Uwe Nestmann for their service as committee members and their valuable feedback.

I thank my colleagues from the Theory of Programming group in Berlin and from the graduate school SOAMED for all the support over the last years. I really enjoyed the warm and pleasant working atmosphere, our productive discussions, and the spirit of critically evaluating a presentation. I also would like to thank all colleagues from the Architecture of Information Systems group in Eindhoven and from the Theory of Programming Languages and Programming group in Rostock. Thank you for the discussions and hints concerning my work and for the pleasant and familiar atmosphere in general.

Many thanks I would like to address to the secretaries in Berlin and Eindhoven Birgit Heene, Sabrina Melchert, Diana Walter, Riet van Buul, and Ine van der Ligt for their help and support in so many administrative things.

Finally, I thank my loving family and all of my friends for their kind support over the past four years and for making my life more pleasant, more interesting and more fun than I ever imagined. You are too numerous to be mentioned here explicitly without me running the risk of unintentionally omitting someone, which I would rather not. Thanks to you all.

CURRICULUM VITÆ

- October 18, 1984 Born in Dresden, Germany
- 09/1991–08/1997 Primary School “Bernhard Grzimek” in Berlin, Germany
- 09/1997–06/2004 Grammar School “Georg Forster” in Berlin, Germany; university entrance exam
- 10/2004–09/2005 Voluntary Conscript at the 352nd Military Police Battalion in Potsdam, Germany
- 10/2005–07/2010 Studies of Computer Science with minor subject Business Administration at the Humboldt-Universität zu Berlin, Germany; Specialization in modeling, specification, and verification of distributed systems, formal methods; degree *Diplom-Informatiker*
- 04/2007–09/2007 Trainee and Working Student at Exozet Games GmbH in Berlin, Germany; developer for artificial intelligence, JavaME applications and games
- 04/2008–07/2010 Working Student at the Humboldt-Universität zu Berlin, Germany; DFG-Project “Synthesis of Behavioral Adapters” and Theory of Programming Group (Prof. Dr. Wolfgang Reisig)
- since 07/2010 binational Ph.D. Student in Computer Science at the Humboldt-Universität zu Berlin, Germany, and the Eindhoven University of Technology, The Netherlands; Member of the B.E.S.T-Project (Berlin-Rostock-Eindhoven Service Technology)
- 07/2010–07/2013 Research Associate at the Humboldt-Universität zu Berlin, Germany; Graduate School SOAMED (“Service-oriented Architectures for the Integration of Software-based Processes, exemplified by Health Care Systems and Medical Technology”) and Theory of Programming Group (Prof. Dr. Wolfgang Reisig)
- since 07/2013 Research Associate at the Eindhoven University of Technology, The Netherlands; Architecture of Information Systems Group (Prof.dr.ir. Wil M.P. van der Aalst)

TABELLARISCHER LEBENSLAUF

18. Oktober 1984	geboren in Dresden, Deutschland
09/1991 – 08/1997	Bernhard-Grzimek-Grundschule in Berlin, Deutschland
09/1997 – 06/2004	Georg-Forster-Gymnasium mit mathematisch-naturwissenschaftlichem Profil in Berlin, Deutschland; Abitur
10/2004 – 09/2005	Freiwillig Wehrdienstleistender im Feldjägerbataillon 352 in Potsdam, Deutschland
10/2005 – 07/2010	Studium der Informatik mit Nebenfach Betriebswirtschaftslehre an der Humboldt-Universität zu Berlin, Deutschland; Studienschwerpunkte: Modellierung, Spezifikation und Verifikation verteilter Systeme, formale Methoden; Abschluss als <i>Diplom-Informatiker</i>
04/2007 – 09/2007	Praktikant und Werkstudent bei der Exozet Games GmbH in Berlin, Deutschland; Entwickler für Künstliche Intelligenz, JavaME Anwendungen und Spiele
04/2008 – 07/2010	Werkstudent an der Humboldt-Universität zu Berlin, Deutschland; DFG-Project "Synthese von Verhaltensadaptern" und am Lehrstuhl Theorie der Programmierung (Prof. Dr. Wolfgang Reisig)
seit 07/2010	binationaler Promotionsstudent in Informatik an der Humboldt-Universität zu Berlin, Deutschland, und an der Eindhoven University of Technology, Niederlande; Mitglied im B.E.S.T-Projekt (Berlin-Rostock-Eindhoven Service Technology)
07/2010 – 07/2013	Wissenschaftlicher Mitarbeiter an der Humboldt-Universität zu Berlin, Deutschland; Graduiertenkolleg SOAMED ("Service-orientierte Architekturen zur Integration Softwaregestützter Prozesse am Beispiel des Gesundheitswesens und der Medizintechnik") und Lehrstuhl Theorie der Programmierung (Prof. Dr. Wolfgang Reisig)
seit 07/2013	Wissenschaftlicher Mitarbeiter an der Eindhoven University of Technology, Niederlande; Architecture of Information Systems Group (Prof.dr.ir. Wil M.P. van der Aalst)

SELBSTSTÄNDIGKEITSERKLÄRUNG

Ich erkläre hiermit, dass

- ich die vorliegende Dissertationsschrift "Verifying Responsiveness For Open Systems By Means Of Conformance Checking" selbstständig und ohne unerlaubte Hilfe angefertigt habe;
- ich mich nicht bereits anderwärts um einen Doktorgrad beworben habe oder einen solchen besitze;
- mir die Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät II der Humboldt-Universität zu Berlin bekannt ist (veröffentlicht im Amtlichen Mitteilungsblatt Nr. 34/2006).

Berlin, den 9. April 2014

Richard Müller