Software Components and Formal Methods from a Computational Viewpoint

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von

Christian Lambertz
aus Viersen

Mannheim, 2012

Dekan: Prof. Dr. Heinz Jürgen Müller, Universität Mannheim
Referentin: Prof. Dr. Mila Majster-Cederbaum, Universität Mannheim
Korreferent: Prof. Dr. Jan Friso Groote, Eindhoven University of Technology

Tag der mündlichen Prüfung: 31. Januar 2013

Abstract

Software components and the methodology of component-based development offer a promising approach to master the design complexity of huge software products because they separate the concerns of software architecture from individual component behavior and allow for reusability of components. In combination with formal methods, the specification of a formal component model of the later software product or system allows for establishing and verifying important system properties in an automatic and convenient way, which positively contributes to the overall correctness of the system. Here, we study such a combined approach. As similar approaches, we also face the so-called state space explosion problem which makes property verification computationally hard. In order to cope with this problem, we derive techniques that are guaranteed to work in polynomial time in the size of the specification of the system under analysis, i.e., we put an emphasis on the computational viewpoint of verification. As a consequence, we consider interesting subclasses of component-based systems that are amenable to such analysis. We are particularly interested in ideas that exploit the compositionality of the component model and refrain from understanding a system as a monolithic block. The assumptions that accompany the set of systems that are verifiable with our techniques can be interpreted as general design rules that forbid to build systems at will in order to gain efficient verification techniques. The compositional nature of software components thereby offers development strategies that lead to systems that are correct by construction. Moreover, this nature also facilitates compositional reduction techniques that allow to reduce a given model to the core that is relevant for verification. We consider properties specified in Computation Tree Logic and put an emphasis on the property of deadlock-freedom. We use the framework of interaction systems as the formal component model, but our results carry over to other formal models for component-based development. We include several examples and evaluate some ideas with respect to experiments with a prototype implementation.

Zusammenfassung

Komponentenbasierte Softwareentwicklung ist ein vielversprechender Ansatz um die Entwurfskomplexität großer Softwareprodukte zu beherrschen, da die Anforderungen der Softwarearchitektur und des individuellen Komponentenverhaltens voneinander getrennt werden und die Wiederverwendbarkeit der Komponenten ermöglicht wird. In Kombination mit formalen Methoden wird durch die Spezifikation eines formalen Komponentenmodells des Softwareprodukts oder Systems die Prüfung und Verifikation von wichtigen Systemeigenschaften auf automatische Weise durchführbar, was positiv zur Korrektheit des Systems beiträgt. In dieser Arbeit untersuchen wir einen solchen kombinierten Ansatz. Wie vergleichbare Ansätze begegnen auch wir dem Problem der Zustandsraumexplosion, das die Verifikation von Eigenschaften algorithmisch schwer macht. Um dieses Problem zu bewältigen, betrachten wir Techniken, die eine Eigenschaftsprüfung in Polynomialzeit bezüglich der Spezifikation des zu analysierenden Systems garantieren, d. h. wir legen Wert auf die algorithmischen Gesichtspunkte der Verifikation. Als Konsequenz untersuchen wir Teilklassen von komponentenbasierten Systemen, die für eine solche Analyse zugänglich sind. Wir sind insbesondere an Techniken interessiert, welche die Kompositionalität des Komponentenmodells ausnutzen, und betrachten ein System nicht als monolithische Einheit. Die Annahmen, die durch die Menge der mit unseren Ansätzen verifizierbaren Systeme aufgestellt werden, können hierbei als allgemeine Entwurfsregeln verstanden werden, die es verbieten Systeme nach Belieben zu konstruieren um dafür effiziente Verifikationstechniken zu erhalten. Die kompositionelle Natur der Komponenten ermöglicht dabei Entwicklungsstrategien, die zu konstruktionsbedingt korrekten Systemen führen. Des Weiteren werden kompositionelle Reduktionstechniken begünstigt, die ein gegebenes Modell auf den für Verifikation relevanten Teil verkleinern. Wir betrachten in der Logik CTL spezifizierte Eigenschaften und legen einen Schwerpunkt auf Verklemmungsfreiheit. Wir benutzen Interaktionssysteme als formales Komponentenmodell, allerdings sind unsere Ergebnisse auch auf andere formale Modelle der komponentenbasierten Entwicklung übertragbar. Wir geben mehrere Beispiele an und evaluieren einige Ideen mittels einer Prototyp-Implementierung.

Acknowledgements

I would like to thank all the people who supported me with technical advices and personal encouragements while I was writing my dissertation.
I thank my advisor, Professor Mila Majster-Cederbaum, for introducing me to the area of formal methods, for providing research directions and insights, and for the time she spent reading and discussing preliminary versions of this work. Furthermore, thanks to Professor Jan Friso Groote for agreeing to evaluate my dissertation as second examiner and to participate in my defense. He provided valuable feedback that improved the quality of my dissertation.

Contents

List of Algorithms

List of Definitions

List of Figures

List of Tables

Chapter 1

Introduction

The spread of software continues and has become ubiquitous in many aspects of our daily lives, e.g., mobile phones becoming smart, the pervasive use of online services and social networking for daily tasks and duties, or the rise of embedded systems. We take the availability of such software for granted and put trust in its correctness, i.e., we become more and more dependent on reliable software. Moreover, software systems, which typically denote huge and complex software products, already control many critical industrial systems ranging from factory robots, that can interact with humans, over avionics and automotive systems to supervision of nuclear power plants. The modern stock exchange operates nearly autonomously and driverless trains emerge in many metros and urban railways. This pervasiveness of software creates new challenges because the malfunction and unexpected behavior of software systems can cause effects that are unpleasing such as the annoyance of users, painful such as financial loss, and unacceptable such as catastrophes and disasters.

Modern software development has to take these issues into account and master the inherent design complexity of software systems. In this direction, structured and systematic approaches have been successfully applied that decompose a system's specification into various, often heterogeneous parts with respect to the system's functionality and first consider a model of the later software product, where the initial correctness of this model can be used to support and guarantee the correctness of further development steps. The advantage of such an approach is that faulty design decisions become detectable as early as possible and can be pinpointed exactly in the software development process. It is well known that the earlier an error is detected, the more inexpensive and effortless is its revision and mitigation [43, 173, 262]. Furthermore, the model can be used to correctly integrate already trusted and verified system parts,

which enhances the interoperability and reusability, where missing functionality can also be bought as prefabricated components—so-called commercial off-the-shelf components—on software markets. Such a model also allows to generate executable code as methodologies known as component-based and model-driven software development have shown.

In this thesis, we focus on the correctness of the initial model particularly with regard to the methodology of component-based development. The verification techniques used to establish this correctness are part of a broader methodology known as formal methods. Our research question is: How can the particular features of component-based systems, such as their compositional nature and the separation of the concerns of software architecture from individual component behavior, be exploited for the application of formal methods, and how can we measure this potential benefit from a computational viewpoint. In the next section, we introduce and discuss these concepts that also constitute the title of this thesis.

1.1 What This Thesis Is About

By means of the title of this thesis, which is

- *Software Components* and
- *Formal Methods* from a
- *Computational Viewpoint*

we introduce and discuss the thesis' topic in the following sections.

1.1.1 What Are Software Components?

The word "software" has become everyday language in most countries and typically denotes everything related to the virtual parts that run a computing device—having its origins in the word "hardware" which is used to describe the electrical/physical parts. The word "component" has been used quite differently in the literature, e.g., there are electrical components, thermodynamic components, or graph-theoretical components. The word itself means to put something together.

Its first occurrence with respect to software dates back to the year 1968 where McIlroy [191] proposed to change the way software is developed to a more industrialized way, which he calls mass production techniques. He envisions a standard catalog of software routines called *software components* that can

be applied on different machines, i.e., the components make no assumptions about their environment and are thus more independent with respect to their deployment. He compares this approach to the way hardware components are employed: they are usually interchangeable and cataloged.

However, a more methodical and scientific approach to component-based software began not until the 1990s, although similar ideas can be found in paradigms such as object-oriented programming, e.g., Cox's software integrated circuits [78] in the 1980s, and the importance of the modularity and structure of software systems was already emphasized by Dijkstra [91], Brooks and Iverson [52], and Parnas [223] in the 1970s. The work on so-called software architectures as a system design pattern, which also distinguishes components and their interconnection, was brought forward by Garlan and his coauthors [3, 7, 109, 110], culminating in a 1996 book on the topic [242]. The re-use of design patterns was addressed in the seminal book of the so-called Gang of Four [107] around 1994. The now annual international conference on *component-based software engineering* was initially happening as a workshop in 1998. The first textbooks on component-based development appeared around the millennium, e.g., the books of Heineman and Councill [134], Veryard [260], and Szyperski [249]. Since then, components have also been successfully used in the enterprise world, e.g., OMG's Common Object Request Broker Architecture (CORBA)[1], Microsoft's Component Object Model (COM)[2], or Oracle's (formerly Sun's) Enterprise JavaBeans (EJB)[3]. Early ideas can also be found in IBM's System Object Model (SOM), but it was discontinued in 1997. For a thorough historical overview, we refer to the work of Barroca et al. [27].

So what is a software component? To quote Szyperski [249, page xxi]: "Software components are binary units of independent production, acquisition, and deployment that interact to form a functioning system."

Here, we follow this definition and understand a software component as an entity with a specific but independent behavior that offers so-called *ports* to interact and cooperate with its environment, which consists of the other components. Naturally, this independence and the related properties of interchangeability and interoperability are only achievable with a strong separation of concerns, a principle suggested by Dijkstra [92], that applied here translates to separating the way the components are put or glued together—which is thus called the *glue code*—and their individual behavior. Therefore, each component has two layers: One for cooperation that specifies the ports and that can be referred to in the glue code and an internal one that describes the be-

[1] `http://www.omg.org/spec/CORBA/`
[2] `http://www.microsoft.com/com/`
[3] `http://www.oracle.com/technetwork/java/javaee/ejb/`

havior and that cannot be accessed by other components or the glue code. This separation ensures the reusability of the components since no assumptions are made on the context where a component is deployed in. Furthermore, the independent production of functionality or its acquisition is supported in this way which is in accordance with the definition of Szyperski [249].

In the introduction to this thesis, we wrote that a model of the later software product should be available as early as possible to guarantee the correctness of further development steps. Thus, a component model needs to be defined that is compliant with the definition of a software component as above and that allows for correctness proofs, i.e., enables the usage of formal methods that we address in the next section.

1.1.2 Approaching Formal Methods

The use of *formal methods* aims at applying mathematical techniques to the process of specifying and verifying software and hardware systems [67], i.e., the specification stipulates *what* the system should do by means of formal properties and *how* the system behaves by means of a formal model whereas the verification ensures that the properties are satisfied by the model. Moreover, verification techniques can also be used to show the conformance of two models where typically one model refines the other other one, e.g., when parts of an abstract model are substituted by a more concrete implementation. The level of mathematical rigor makes it possible that all deductions about a system can be formally proven if the structures and methods are well defined, i.e., we have a formal syntax and semantics. However, as pointed out by Clarke and Wing [67], formal methods do not a priori guarantee correctness, and DeMillo et al. [88] already argued convincingly in 1979 that program verification will never play the role of proofs in mathematics—which was believed at this time. Nevertheless, such formal techniques often reveal inconsistencies, ambiguities, and incompleteness, i.e., faulty design decisions that should be detected as earlier as possible. This particularly holds for concurrent systems where such faults are very hard to detect, e.g., unforeseen interleaving or mutual access to shared resources [23, Example 1.1]. We refer to the book of Peled [225, Chapter 1] and to Wing [263] for an exhaustive introduction to formal methods, but address some highlights in the next paragraphs—a historical overview is given by Baeten [21] and Clarke and Wing [67].

The history of formal specification of software goes back to the design of programming languages where, e.g., Backus [18] suggested his BNF notation for the formal syntax of a programming language in 1959. Even earlier, but not in a software related context, the field of formal language theory offers a

variety of concepts for formal specification, e.g., the string rewriting system by Thue [252] around 1914, the Post canonical system [232] published in 1943, or Chomsky's work on grammars [64] in the mid of the 1950s. Of course, Turing machines [254] and the lambda calculus [65] introduced in the 1930s can also be understood as a formal specification. These formalisms have in common that they are models for sequential computation. With respect to concurrency, where software components can be understood as concurrently cooperating entities, seminal work by Petri [227] and Dijkstra [90] in the 1960s laid the foundation for a whole research area that is nowadays known as concurrency theory. In particular, the work on process algebras such as Hoare's Communicating Sequential Processes (CSP) [141], Milner's Calculus of Communicating Systems (CCS) [200], and Bergstra and Klop's Algebra of Communicating Processes (ACP) [37] paved the way for the formal and mathematically rigorous modeling and analysis of concurrent systems. We refer to Cleaveland and Smolka [73] for a thorough historical overview of concurrency theory and to Baeten [20] for a brief history of process algebra.

A rich body of work addresses formal methods with respect to more concrete implementations. For instance, the Vienna Development Method [42] introduced at IBM around 1972 in work on compiler development, which became standardized in the 1990s by the International Organization for Standardization, provides both a high-level modeling to conduct correctness proofs due to its formal semantics and a refinement process that allows to turn the model into executable code. Comparable work was done by Abrial et al. [4] on their Z notation, which influenced a lot of further work such as the B-Method which has been used to design the software of the Paris Métro Line 14 which completely operates automatically [168] and of which it is claimed that there are no bugs found yet. Another example is the Analytical Software Design (ASD) method [48] that incorporates formal methods to traditional development of industrial control software and which has for instance been used to develop the control components of an interventional X-ray system at the company Philips Healthcare [215]. The analysis by Groote et al. [132] of the effects of applying the ASD method in this case shows that not only the quality of the software is improved, but also the development time is reduced.

The properties of a system can also be described as a model but it is more convenient and less error-prone to use a formal logic. Here, temporal logic has proven to be useful and has been put in a program verification context around 1980, e.g., the seminal work by Pnueli [230], Lamport [165], Ben-Ari et al. [33], Clarke and Emerson [66], and Emerson and Halpern [95]. Interestingly, the use of logic to assert the correctness of computer programs was already addressed by Hoare [139] at the end of the 1960—known today as Hoare logic—based on similar work by Floyd [101] on flowcharts. Here, each program step is

equipped with pre- and postconditions and by means of axioms and interference rules the correctness of the program, which in this case corresponds to a correct input-output behavior, can be deduced. Hence, this formal method is known as deductive program verification. In this direction, so-called theorem provers such as PVS [217], Isabelle/HOL [213], or ACL2 [155] can help to mechanize the deduction process or provide interactive guidance and have successfully been applied in the context of formal methods. For instance, the book of Kaufmann et al. [156] contains various industrial case studies that show how ACL2 helps to "design, build, and maintain hardware and software systems faster and more reliably" [156].

Nowadays, due to the seminal work of Clarke and Emerson [66] and Queille and Sifakis [235] from 1982, the property verification process can be carried out in an automatic way known as *model checking*, which allows for establishing correctness of systems of realistic size and has been successfully used in a number of projects [194]—especially due to the advent of symbolic model checking [57] in the 1990s using binary decision diagrams [55] as the underlying data structure. However, as we discuss in the next section, model checking can be infeasible if it is applied without further insights. We want to mention the textbooks of Clarke et al. [72] and Baier and Katoen [23] that offer a good introduction and exhaustive information on the topic of model checking.

Now, for a successful marriage of software components and formal methods, a formal component model needs to be defined, i.e., the behavior of the components and the cooperation by means of the ports needs to be mathematically specified, which then allows for verification of system properties. Various formalisms have been suggested to be used as such a model, e.g., process algebras [8, 176], channel-based methods [12], interface theories and automata [47, 81], Petri nets [1, 79], and interaction systems [121–123, 243]—which we also use as the formal component model in this thesis.

In interaction systems, each component provides a set of *actions* which are understood as ports of the associated component and which abstract from data and input/output operations. These ports are used to glue components together by so-called *interactions*, i.e., interactions model the cooperation among the components. Such an interaction consists of actions of different components and is also a step of the *global behavior*, i.e., it can be executed only if all participating components are able to execute their appropriate action. Each component's ability to execute actions, i.e., the behavior of the component, is modeled as a labeled transition system, where the set of labels equals the set of actions. The global behavior of the system is derived by executing the interactions nondeterministically according to their executability.

Here, we do not continue to compare this model with the above mentioned formalisms; we postpone this study to Chapter 2 where we also formally define interaction systems and show how properties can be verified. Next, we address the third constituent of the thesis' title: The computational viewpoint.

1.1.3 The Computational Viewpoint

A computational viewpoint is of course inherent in computer science because of its nature to approach problems algorithmically under the limitations of the underlying computing hardware. The success of a more theoretical view, which was introduced among other concepts with the definition of Turing machines [254], establishes an abstraction of current computational limitations and provides a taxonomy known as computational complexity theory that classifies problems with respect to their computational hardness [108, 220]. Moreover as argued by Karp [154], science in many fields can be seen through a so-called computational lens, i.e., systems in, say, physical or social science or in engineering can be classified by their computational requirements, and an algorithmic worldview offers new ways to understand scientific problems. Applying this computational view often changes the question of well-defined problems from "can it be solved" to "can it *efficiently* be solved" (which is, of course, only possible if the problem is decidable). We embrace this view in this thesis and focus on ideas and directions that provide an *efficient* solution from the computational perspective. Moreover, we analyze the costs of all our algorithmic approaches and classify problems with respect to their computational complexity.

In our setting where we apply formal methods to (a model of) software components, a well-known problem exists that complicates the quest for efficient techniques, viz. the state space explosion problem. Here, the combinatorial possibilities of the interleaved behavior of the components (which we call, as already mentioned above, the global behavior of an interaction system) are numerous, i.e., exponential in the number of components. Thus, property verification is a computationally hard task because a state space analysis of the global behavior can be infeasible.

We want to mention that several techniques have been developed to mitigate the explosion of the state space. For instance, partial order reduction [118] aims at identifying actions whose order of execution (in a sequence) does not affect the state that is finally reached. It is thus not necessary to analyze all intermediate states with respect to these sequences of actions since one execution sequence is sufficient if we are only interested in the final state. Clearly, the omission of all other execution sequences and the involved in-

termediate states can significantly reduce the state space. Similar ideas are known as the stubborn set method [255] or model checking using representatives [224], and the idea behind τ-confluence (together with τ-priorisation to yield a state space reduction) is also related [127, 128]. Other ideas exploit symmetries if several identical components constitute the system in question or if there are structural symmetries in the system's description in order to gain a state space reduction [70, 98, 147], or the compositionality of the system is exploited [69, 125, 126, 150, 167]. The process of abstraction is also helpful in this direction where a verification question sometimes can be answered on a small representation of the state space (the abstract model) instead of the original, huge one (the concrete model). Here, work by Cousot and Cousot [77] known as abstract interpretation provides relations between abstract and concrete models which allows for establishing properties in the abstract model that also hold in the concrete one. Such abstraction techniques have also been applied for model checking [71].

Moreover, there are efficient, tailored approaches with respect to a certain property that are comparable to a (mathematical) sufficient condition [8, 16, 39, 49, 124, 137, 146, 179, 180]: If certain preconditions hold locally, then we can conclude that the property in question holds globally, i.e., such conditions aim at providing efficiently checkable preconditions ("locally"), that ensure the property's validity without constructing the system's overall state space ("globally"). Seen from a computational viewpoint, such conditions are very attractive if the satisfiability check of all preconditions can be carried out in polynomial time (with respect to the input size of the system).

In the next section, we address the connection of the above discussed topics to this thesis.

1.2 Goals and Findings

We address the goals that we want to achieve in this thesis, i.e., we give the problem statement, motivation, and the methods we apply, and our findings, i.e., we give the contribution of this thesis, a short roadmap, and relations to other work in the literature.

1.2.1 Problem Statement and Motivation

We approach component-based development from a mathematically rigorous and formal point of view including the computational aspects. Based on work by Gössler and Sifakis [121, 123], we consider a model for software

components, called interaction systems, that takes the desired features of component-based development into account and that allows for the specification of complex software systems, but is amenable for the application of formal methods such as property verification techniques, i.e., proving that a certain model of a system satisfies a certain property. In particular, we research on how the features of software components can be exploited to make property verification an easier job. The goal of our research is that we want to derive *efficient and automatic* property verification techniques, i.e., we put an emphasis on efficiency and practicability in our approaches and establish algorithms that run in polynomial time in the size of their input, which is the system's specification. However, this efficiency cannot be guaranteed for all systems due to computational complexity results, and we thus consider assumptions under which we can check whether a certain property holds by means of a polynomial-time algorithm. These assumptions are derived from the features of component-based systems, which supports that the component-based idea allows for the development of tailored formal methods.

We want to point out that our research should be understood as being complementary to other formal techniques, i.e., we envision a toolbox of formal methods from which a software developer can choose or which are applied automatically in the background during the specification and development of a software product.

The motivation for such an approach was already mentioned above: The earlier a fault is detected in a software product, the cheaper and easier is its correction or mitigation. Since typical development strategies start with a model of the later product, we need to ensure the correctness of this early model in order to be able to rely on it in later development steps, e.g., when we substitute real software code for the components. Moreover, as Groote et al. [132] discussed convincingly, formal methods not only improve the quality of software products, but also reduce their development time, i.e., there is no good reason to refrain from using formal methods in software development.

1.2.2 Methodologies

We formalize all ideas in a mathematical notation and build our work solely on these definitions, i.e., we avoid ambiguities that are typically introduced with informal texts. This rigorous work enables us to formally prove all statements of this thesis. Since we are interested in automatic and efficient approaches, we give pseudocode implementations of all (non-obvious to implement) results and analyze these algorithms with respect to their computational costs (in

an abstract machine model) and provide upper bounds for their runtime. Moreover, we experimentally evaluate some of the results by means of a prototype implementation to support our interpretation of the theoretical bounds (in a concrete machine model). We give detailed references if we employ methods from the literature and illustrate our ideas by means of examples; in particular, we use a running example that follows us through all chapters and which we introduce in Section 1.3 of this chapter.

1.2.3 Contributions and Roadmap

We give a high-level overview of our contributions and provide a roadmap that links to the corresponding chapters. We contribute the following:

1. We extend the original model of interaction systems [121] to allow for unobservable behavior—by means of an advanced interaction model and an adapted global behavior—and multiple initial states. Furthermore, we provide a behavioral equivalence for interaction systems that, after a discussion of suitable equivalences from the literature, corresponds to branching bisimilarity with explicit divergence [115].

2. We discuss several properties of interaction systems with a focus on deadlocks and livelocks. Here, deadlocks have been immensely studied in interaction systems, and particularly the computational complexity of deadlock detection has been precisely determined, which we review in detail. We introduce the property of livelock-freedom for interaction systems, which becomes important with the introduction of our advanced interaction model, and also address its computational complexity. Moreover, we define properties by using formal logic as a means for specification, which also enables the use of model checkers in interaction systems. However, the well-known phenomenon of state space explosion can render model checking on the overall system behavior infeasible as discussed in Section 1.1.3. This opens the challenge to find verification techniques that avoid or mitigate state space explosion.

3. We introduce three operators that allow for compositionality and abstraction in interaction system in order to support the concepts of correctness by construction, hierarchical modeling, compositional reduction, and compositional verification:

   - A composition operator that allows for composing arbitrarily many interaction systems in a flexible way and that is guaranteed to yield a valid interaction system. We also derive a binary variant and show its associativity and commutativity.

- An abstraction operator that enables the modification of an interaction system's interaction model such that certain interactions become unobservable in the system's global behavior and are no longer available for further composition steps.
- A decomposition operator that projects a given interaction system to a subset of the set of components. The resulting system is again an interaction system, and we show how it can be re-composed with the remaining components to get back to the original system in an automatic and convenient way.

We discuss the connection of these operators, i.e., how they can be used together in an algebraic way, and derive various useful properties such as whether the chosen behavioral equivalence is a congruence with respect to the operators for interaction systems. Moreover, we address under which assumptions the composition operator allows for a correctness-by-construction approach.

4. In order to open up new ways to tackle the above mentioned state space explosion problem, we introduce architectural constraints for interaction systems that are based on undirected graphs and that restrict the cooperation structure, i.e., the way the components are allowed to cooperate. Here, constraints that postulate the acyclicity of the cooperation structure have been proven to be useful for the efficient verification of the property of deadlock-freedom. The intuition behind considering acyclic cooperation structures is that in such systems, the presence of circular waiting situations and hence potential deadlocks is reduced. We define two acyclic constraints, viz. *star-like* and *tree-like* architectures. Moreover, we drop the acyclicity requirement and admit certain cycles that exclude a circular waiting of three or more components such that the reasons of the single waits are unrelated. This leads to a novel architectural constraint called *disjoint circular wait free* architecture, which is particularly useful in systems with multiway cooperations when, e.g., one component i cooperates with components $j_1, \ldots, j_n$ (individually and all $n+1$ together) but no j_k cooperates with a j_l without i (for $k \neq l$).

5. We show that checking whether a given interaction system satisfies one of our architectural constraints can be carried out in polynomial time in the number of components and interactions. This time bound is essential since the check must not be more costly than the verification of a condition that is based on such a constraint. Here, the procedure is straightforward for star-like and tree-like architectures but more involved for disjoint circular wait free ones. We show that we can nevertheless answer this question in polynomial time by means of flow networks.

6. We study the classification of interaction systems with a disjoint circular wait free architecture (and discuss similar classifications for the other architectures from the literature):
   - We show that an arbitrary interaction system can be transformed in linear time into a system obeying our architectural constraint of disjoint circular wait freedom and exhibiting nearly identical behavior, viz. isomorphic up to transition relabeling. Potentially, this opens the door for more techniques that explicitly rely on our architectural constraint.
   - We give a linear-time many-to-one reduction based on the above mentioned transformation which allows us to derive various computational complexity results for architecturally constrained interaction systems from results known for arbitrary interaction systems.
7. The introduction of behavioral equivalences in interaction systems allows for a compositional reduction technique. We address how such a reduction can be employed in interaction systems in a compositional and efficient way, i.e., without constructing the global behavior and thus circumventing the state space explosion problem. In this connection, star-like and tree-like architectures are important in order to identify the components that are amenable to such a reduction. Furthermore, the approach allows for an exponential speedup of property verification in certain scenarios.
8. We show how to use architectural constraints to establish a polynomial-time checkable condition for deadlock-freedom of interaction systems that extends an earlier approach by Majster-Cederbaum and Martens [180]. The class of systems that can be tackled with our condition extends the class of systems studied by various authors [39, 49, 137, 179, 180] with respect to the architectural constraints. We evaluate this conditional approach with respect to global state space analyses.
9. We improve the approach of Majster-Cederbaum and Martens [180] by avoiding their polynomial-time preprocessing step that establishes a property of interaction systems called "strongly exclusive communication", which is important for the technique presented by the authors. But, this preprocessing possibly enlarges the behavior of the components for the verification process. We show that this step is completely unnecessary as the required information can already be extracted beforehand. We demonstrate the effect of this avoidance with an experimental evaluation of several example systems.
10. We introduce a gray-box view for interaction systems. Such a view

typically stipulates that the behavior of the components is not fully accessible—as in a white-box view—and is reasonable if the complete behavior should not be disclosed or is still in development. Now, it is interesting to ask whether we can still verify properties of the whole system if we do not have access to the full behavior of the components, or which additional assumptions we need such that property verification becomes possible. We show that small representatives of the behavior with respect to a component's cooperation partners suffice to verify deadlock-freedom if they comply with a certain equivalence. Moreover, the small size of these representatives allows for a quadratic speedup of deadlock detection in some situations.

We introduce and discuss these contributions in detail in the following chapters and in doing so keep to the following roadmap: We formally define interaction systems and their properties in Chapter 2, discuss the three operators in Chapter 3, introduce architectural constraints in Chapter 4, show how these constraints allow for compositional reduction in combination with the operators in Chapter 5, address the efficient verification of deadlock-freedom in Chapter 6, establish the gray-box view in Chapter 7, and conclude the thesis in Chapter 8.

We want to point out that part of the work enumerated above has already been published. The extensions of the original model (Chapter 2) were presented at the doctoral symposium of the FM 2009 conference [160] and at the ICE 2010 workshop [161] where the first work also introduced the operators for interaction systems (Chapter 3) and the reduction technique (Chapter 5) and the latter work the gray-box view for interaction systems (Chapter 7). Our best-paper award winning work on architectures and deadlock-freedom (Chapters 4 and 6) appeared in the proceedings of the FSEN 2011 conference [162] and an extended and more detailed version was submitted to Elsevier's SCP journal [163].

1.2.4 Related Work

We already mentioned that various authors studied formal models for software components—such that formal methods become applicable—and addressed some related work in Section 1.1. Moreover, the notion of an architecture description language that describes software architectures [110, 192] is related to the idea of software components, e.g., the separation of behavior and topology is an important guideline for software architectures [6]. In these directions, we here exemplarily mention the work on PADL by Bernardo et al. [39], Java/A by Hennicker et al. [137], SOFA by Plášil et al. [229], Darwin

by Magee et al. [176], and Wright by Allen and Garlan [8]. Since our work is based on and extends the formalism of interaction systems as introduced by Gössler and Sifakis [121], we provide a comparison with respect to these (and further) works; however, we postpone this discussion to Section 2.1.4 of the next chapter—after we formally introduced interaction systems.

In a similar vein, we deal with related work regarding our contributions that we enumerated in the previous section: Each chapter has its own related work section where we discuss and compare our results with approaches from the literature. Furthermore, we include references in the beginning of some chapters where we motivate the upcoming ideas, and sometimes directly after a definition to recognize and acknowledge related ideas.

In the next section, we introduce a running example that follows us through all subsequent chapters in order to illustrate our definitions and results.

1.3 Running Example: Merchandise Management

We consider a merchandise management system (MMS) for wholesalers which manages orders of customers and supplies in the wholesaler's storage. The MMS offers several modes of operation: A wholesaler may directly deliver after receiving an order, may demand from a customer to ask for a reservation before ordering, and may accept direct orders but request nevertheless a reservation. Additionally, any reservation is printed out for internal use and all operations can be cancelled with potential reimbursement. This versatile behavior is reasonable if we assume that the MMS is developed by a software company that wants to sell it as a software component to a variety of wholesalers with different requirements.

We now take a look at a particular wholesaler that bought this MMS as a software component and uses a storage system in which goods need to get reserved before they can be delivered. Additionally, this wholesaler requests from its customers to ask for a reservation before placing an order. We assume that the entire software of the wholesaler consists of three parts and is given as three components, one for each of the customer, merchandise management, and storage part. These components need to be glued together in order to form a functioning system that satisfies the needs of the particular wholesaler. Figure 1.1 on the facing page gives an overview of the components, their behavior, and their cooperation.

When we formally define the formal component model in the next chapter, we come back to the merchandise management example to illustrate this

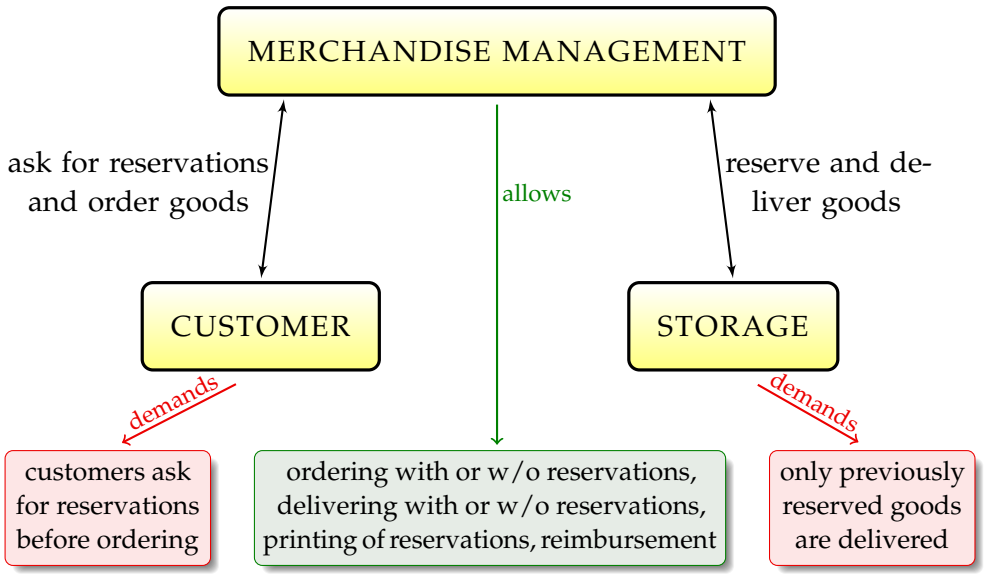


Figure 1.1: The merchandise management example

definition. We conclude the introduction with some conventions for the subsequent chapters.

1.4 Conventions

Throughout this thesis, the following conventions are used for definitions, mathematical statements, and the notation of algorithms in pseudocode, which we use as a high-level algorithmic description language because it allows to neglect unnecessary details of real programming languages such as special variable declarations and obscure syntaxes. Here, we use a similar notation as found in the book of Cormen et al. [76].

We adhere to the following rules where we also include all notations that are used ambiguously in the literature:

- We number all definitions, examples, propositions, lemmata, theorems, and corollaries in a consecutive order where we use one counter for each chapter. In our opinion, this makes it much easier to find referenced entities by browsing through the text.
- Figures, tables, and algorithm listings cannot be included in this numbering scheme because they are placed where they fit in the text. We thus provide a list of figures, tables, and algorithms at pages xxiii, xxvii, and xvii respectively. Furthermore, we provide a list of definitions on

page xix and an index on page 297, which includes all used symbols.

- The empty set is denoted by the symbol "$\emptyset$" or by "$\{\}$".
- The powerset of a set S is denoted by "2^S".
- A subset relation is denoted by "$\subseteq$" and a strict (or proper) subset relation by "$\subset$". For a superset relation, "$\supseteq$" and "$\supset$" are used.
- A Cartesian product is denoted by "$\times$" whereas multiplication is denoted by a centered dot: "$\cdot$". A sequence of products is denoted by "$\prod$" in each case, e.g., $\prod_{i \in \{1,2,3\}} S_i = S_1 \times S_2 \times S_3$ for three sets S_1, S_2, and S_3.
- In pseudocode algorithms, we use the symbol "$=$" as the assignment operator and "==" for equality comparisons.
- For a binary relation $\mathcal{R}$ over a finite set S, i.e., $\mathcal{R} \subseteq S \times S$, we denote its reflexive closure by "$\mathcal{R}^=$", its transitive closure by "$\mathcal{R}^+$", its reflexive transitive closure by "$\mathcal{R}^*$", and its symmetric closure by "$\mathcal{R}^\leftrightarrow$".
- In (predicate) logic formulae, we denote negation by "$\neg$", conjunction by "$\wedge$", disjunction by "$\vee$", and implication by "$\Longrightarrow$". A universal quantification is denoted by "$\forall$" and an existential one by "$\exists$". We ensure throughout the thesis that the interpretation of such a formula is clear from the context, and we use the following abbreviations: We write "$\forall\, x \in P\colon F(x)$" for "$\forall x\big(P(x) \implies F(x)\big)$" and "$\exists\, x \in P\colon F(x)$" for "$\exists x\big(P(x) \wedge F(x)\big)$" where P is a predicate and F is a formula, i.e., the scope of a quantification is extended after the colon to get rid of the outer parentheses.
- A superscripted "τ" denotes that the symbol τ (which is the only symbol used as such a superscript) is an element of a set, i.e., $S^\tau := S \cup \{\tau\}$ for a set S. Similarly, a superscripted slashed τ ("$\not\tau$") denotes that τ is not part of a set, i.e., $S^{\not\tau} := S \setminus \{\tau\}$ for a set S.

In the next chapter, we provide the formal foundation of this thesis.

Chapter 2

Interaction Systems

In this chapter, we formalize the setting of our goals mentioned in the introduction. We give a formal representation of components and their cooperation called *interaction system*, derive its overall observable behavior, define generic, parametrized, and specific properties of this behavior, compare various notions of behavioral equivalence from the literature in this setting, and finally select one of these notions that matches our goals and which we use in the subsequent chapters of this thesis.

2.1 From Components and Interactions to Interaction Systems

As mentioned in Section 1.1 of Chapter 1, the concept of components allows to separate the concerns of the static and the dynamic part of a system. Clearly, this separation should also be visible in a formal model for components. We start out with the static part called *component system* that models the available components and their means for cooperation with their environment. After that, we describe the way the components cooperate which constitutes the *interaction model*. These two entities are then combined and equipped with the dynamic part, which finally results in *interaction systems*.

The following definitions are based on the original work on interaction systems by Gössler and Sifakis [121, 123] with adjustments by Majster-Cederbaum and Minnameier [183, 184], Majster-Cederbaum and Martens [179, 180], and ourselves [160, 161]. The biggest change to the original model occurs for the interaction model—we discuss the differences in detail after its formal definition.

2.1.1 Component System

A component is considered as an entity with a unique name and a unique set of ports (called actions) that are offered for cooperation to its environment. Note that the uniqueness is not a restriction at all, since equal names could be distinguished by additional indices or identifiers.

Definition 2.1 (Component System): A *component system CS* is defined by means of a tuple $(Comp, \{A_i\}_{i \in Comp})$ where $Comp$ is a finite set of *components*, which are referred to as $i \in Comp$, and $\{A_i\}_{i \in Comp}$ is a family of finite sets called *action sets* (one for each component) that are pairwise disjoint, i.e., we have $\forall\, i, j \in Comp\colon i \neq j \implies A_i \cap A_j = \emptyset$. An element a_i of an action set A_i is called an *action* of component i. All actions of CS are contained in the *global action set* $Act := \bigcup_{i \in Comp} A_i$.

We now successively build up a formal representation of our running example, the merchandise management system, as introduced in Section 1.3. First, we specify the set of components:

$Comp = \{c, m, s\}$ —for **c**ustomer, **m**erchandise management, and **s**torage.

Then, we use the operations that were described in Section 1.3 to specify the action set for each component (in alphabetic order):

$A_c = \{abort_c, ask_c, buy_c, refund_c\}$,
$A_m = \{cancel_m, deliver_m, order_m, pay_m, print_m, reimburse_m, release_m, reserve_m\}$,
$A_s = \{mark_s, unmark_s, ship_s\}$.

We always subscript an action with the name of its corresponding component in the sequel. On the one hand, this guarantees that all action sets are disjoint—as required by Definition 2.1—on the other hand, it allows to quickly identify the corresponding component. Note that the component system of our running example is completely specified with $CS = (Comp, \{A_c, A_m, A_s\})$.

Throughout the thesis, we use a graphical representation to visualize component systems. Figure 2.1 on the facing page depicts this representation for our running example. Here, components are depicted as yellow boxes which contain the name of the component as an identifier. Actions are depicted as black semicircles at the border of the component boxes. Next to such a semicircle, we write the name of the corresponding action.

This completes the static part for the individual components. Next, we define how the components cooperate.

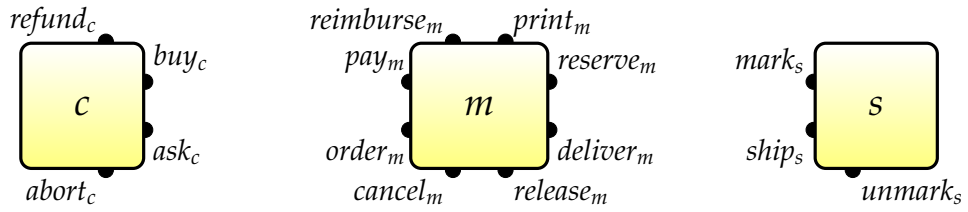


Figure 2.1: Component system of the merchandise management example

2.1.2 Interaction Model

In a component system, each component offers a set of actions for cooperation. As already mentioned in Chapter 1, we only model synchronous communication or rendezvous in this thesis, and thus cooperations among the components can be defined as sets of actions that have to happen synchronously, i.e., each element of such a set is dependent on all other elements. Such a set is called an *interaction* and defined as follows.

Definition 2.2 (Interaction): Let CS be a component system. An *interaction* of CS is a nonempty finite set $\alpha \subseteq Act$ of actions that contains at most one action of every component, i.e., we require $|\alpha \cap A_i| \leq 1$ for all $i \in Comp$. For an interaction α and a component i, we define $i(\alpha) := A_i \cap \alpha$ and say that i *participates in* α if $i(\alpha) \neq \emptyset$. By $\text{compset}(\alpha) := \{i \in Comp \mid i(\alpha) \neq \emptyset\}$ we denote the set of components participating in an interaction α.

We can now use this definition to specify any cooperation or interaction respectively that is possible among the components. This leads us to the definition of the *interaction model* that specifies the set of all allowed interactions. For consistency, we require that each action is contained in at least one interaction. If no cooperation using a particular action of a component is planned, we have to include a singleton interaction modeling that this action is independent. Furthermore, we want to distinguish which of these interactions are visible to an outside observer. Thus, for any interaction we specify its observability: If it is a so-called *closed interaction*, the corresponding synchronization of the participating components is unobservable for an outside observer, otherwise the interaction is called an *open interaction*.

Definition 2.3 (Interaction Model): Let CS be a component system. An *interaction model* IM is a tuple $(CS, Int, Int_{\text{closed}})$ where *interaction set* Int is a finite set of interactions which covers all actions, i.e., $\bigcup_{\alpha \in Int} \alpha = Act$, and *closed interaction set* Int_{closed} is a subset of Int. By $Int_{\text{open}} := Int \setminus Int_{\text{closed}}$ we denote the set of non-closed interactions which we call *open interactions*. We say that a set of components *cooperates* if an interaction exists where all those com-

ponents participate in, i.e., the components in a set $C \subseteq \mathit{Comp}$ cooperate if $\exists\, \alpha \in \mathit{Int}\colon C \subseteq \mathrm{compset}(\alpha)$ holds.

Next, we give an example for Definition 2.3 and introduce the interactions of our running example. We set:

$$\mathit{Int} = \big\{\{\mathit{abort}_c, \mathit{cancel}_m\}, \{\mathit{ask}_c, \mathit{order}_m\}, \{\mathit{buy}_c, \mathit{pay}_m\}, \{\mathit{deliver}_m, \mathit{ship}_s\},\\ \{\mathit{print}_m\}, \{\mathit{refund}_c, \mathit{reimburse}_m\}, \{\mathit{release}_m, \mathit{unmark}_s\}, \{\mathit{reserve}_m, \mathit{mark}_s\}\big\},$$

$$\mathit{Int}_{\text{closed}} = \big\{\{\mathit{print}_m\}\big\}.$$

The interaction $\{\mathit{print}_m\}$ is defined as a closed interaction, i.e., it is perceived as an internal operation by an outside observer. Observe that $\mathit{IM} = (\mathit{CS}, \mathit{Int}, \mathit{Int}_{\text{closed}})$ with CS as defined above (cf. page 18) completely specifies the interaction model of our running example.

We extend the graphical representation that we introduced for component systems to also depict the cooperation information. Figure 2.2 depicts the graphical representation of the interaction model of our running example. Interactions are depicted as black lines connecting the corresponding actions. If an interaction is a singleton, we omit any lines if no other interaction exists where the particular action is part of. Otherwise, we use a loop line connecting the corresponding action with itself. In case an interaction is closed, we use gray color (cf. the print (inter-)action of component m in Figure 2.2).

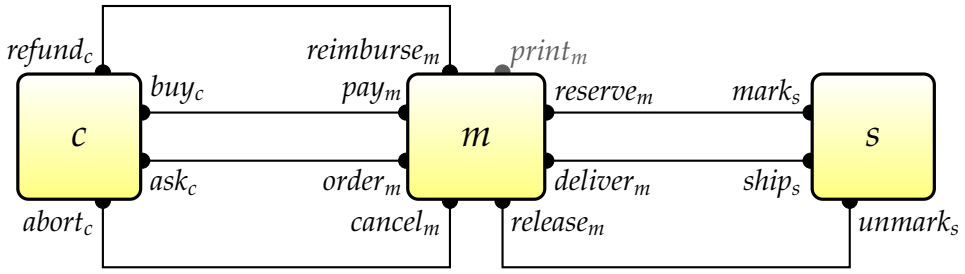


Figure 2.2: Interaction model of the merchandise management example

We have to admit that our running example does not show the flexibility of interaction models but its simplicity is more helpful for illustrating our ideas. But, we give a small example demonstrating this feature.

Example 2.4: Consider a radio station and two listeners modeled as three components rs, l_1, and l_2. The radio station has a single action $\mathit{broadcast}_{rs}$ whereas the listeners have actions listen_{l_1} and listen_{l_2} respectively. Now, an interaction model for this component system allows for a lot of flexibility in the specification because the multiway cooperation, that interaction models allow for, facilitates a very compact and convenient modeling on a certain

level of abstraction. For instance, consider the following set of interactions:

$$Int = \big\{\{broadcast_{rs}\}, \{broadcast_{rs}, listen_{l_1}\}, \{broadcast_{rs}, listen_{l_2}\}, \{broadcast_{rs}, listen_{l_1}, listen_{l_2}\}\big\}.$$

Here, we specify that the radio station is always able to broadcast its signal independently of the number of listeners but no listener can execute its action if the station's signal is unavailable. We can remove the singleton $\{broadcast_{rs}\}$ from this set to require that at least one listener is present, or only allow the interaction of all three components, i.e., $Int = \big\{\{broadcast_{rs}, listen_{l_1}, listen_{l_2}\}\big\}$, to enforce the listening of all available listeners and otherwise block the signal.

We continue with some historical remarks about the interaction model. Originally, Gössler and Sifakis [121] use the name "connector" for a set of actions (of different components) that models a synchronization of the involved components. They require that the set of all connectors covers all actions and that any connector is maximal with respect to set inclusion among all connectors. A special operator "$I(\cdot)$" is defined that returns for a given connector the set of all of its subsets, and such a subset is called an interaction. This operator is extended to also accept a set of connectors such that it returns the union of all subsets of the contained connectors. Additionally to the set of connectors, a set of "complete interactions" is specified in their interaction model that contains some of the possible interactions and any superset of such an interaction that is also an interaction. These complete interactions represent all partial synchronizations (with respect to a connector) that are possible in the system, e.g., if not all components of a connector are ready for synchronization, a subset of these can proceed if a corresponding complete interaction is defined.

This original interaction model was modified and extended several times; we shortly highlight some modifications. Majster-Cederbaum and Minnameier [183] first use the name "*Int*" for the set of all interactions of the original model of Gössler and Sifakis [121], i.e., the union of all sets $I(C)$ for all sets C of connectors. Majster-Cederbaum and Martens [179] modify the original model and only use a set of maximal connectors, i.e., without the set of complete interactions. Later, they also drop the maximality requirement of the connectors as well [180], i.e., any possible synchronization among the components is modeled as a connector. Similarly, Majster-Cederbaum and Minnameier [184] and ourselves [160] neglect the requirements of the original model and additionally drop the now obsolete name "connector" and simply call a set of actions an interaction—note that we also use this understanding in Definition 2.3. The presence of closed interactions, that should be unobservable from an outside view, is introduced to allow for abstraction by ourselves [160].

The definition of the interaction model completes the static part. We now add the behavior of the components to the interaction model, which yields interaction systems.

2.1.3 Interaction System

Several ways of modeling the behavior of an entity in a formal way can be found in the literature, e.g., flow charts, nets, logic, automata, or mathematical equations. Typically, all of these notions have in common that several behavioral states can be distinguished and that the information provided by the current state influences the future behavior, i.e., we have transitions between the states that model behavioral actions taking place.

Here, we follow this basic observation and use *labeled transition systems* for any behavioral description throughout this thesis. According to Baier and Katoen [23, Section 2.5], labeled transition systems where first used for the verification of concurrent programs by Keller [157]—under the related name "named transition systems"—and they can also be used for hardware verification [158, Chapter 3]. But, these systems have a much older and richer history, since similarities can be found with many popular computational models such as Turing machines, finite-state machines, or Petri nets, to mention just a few. For instance, Brookes and Rounds [50] trace them back to the nondeterministic automata of Rabin and Scott [236]. We refer to the book of Savage [239] for a detailed overview of computational models. Lee and Sangiovanni-Vincentelli [170] provide a comparison of such models in a concurrent setting.

Now, adding a labeled transition system for each component that uses the corresponding action set as its alphabet, fully specifies the behavior of each component. Please note that we do not formally define labeled transition systems at this point, but for convenience, included this definition in Appendix A and refer the interested reader to Definition A.1.

Definition 2.5 (Interaction System): Let IM be an interaction model. An *interaction system* Sys is defined by means of a tuple $(IM, \{LTS_i\}_{i \in Comp})$. The *behavioral model* $\{LTS_i\}_{i \in Comp}$ is a family of labeled transition systems with $LTS_i = (S_i, A_i, \{\xrightarrow{a_i}_i\}_{a_i \in A_i}, S_i^0)$, i.e., for each component exists a labeled transition system over its action set[1] which we call the *(local) behavior* of the component. For convenience, we write $[\![i]\!]$ instead of LTS_i. Further, we assume that the components' sets of states are disjoint, i.e., $\forall\, i, j \in Comp\colon i \neq j \implies S_i \cap S_j = \emptyset$, and that all sets of states are nonempty, i.e., $\forall\, i \in Comp\colon |S_i| \geq |S_i^0| > 0$.

[1] "Labeled transition system over a set" means that this set is the alphabet of the system.

We require that at least one state is present for any component, because this is the simplest behavior that can be modeled using labeled transition systems. An empty set of states does not correspond to "no-behavior", it is an underspecification of the system—no-behavior is a single initial state with no outgoing transition.

Historically, the labeled transition systems modeling the behavior of the components were only allowed to have one initial state, i.e., $|S_i^0| = 1$ for all $i \in Comp$. Since this is not an important restriction and other models, e.g., constraint automata [24], allow for more than one initial state, we here also use this more general definition, i.e., we only require $|S_i^0| \geq 1$ for all $i \in Comp$.

We now complete the specification of our running example by introducing a labeled transition system for each of the three components. Here, we directly use the informal description of the behavior of the components given in the introduction of the example in Section 1.3. For instance, the management component allows for ordering goods as well as directly paying them, which we model as transitions leading to different states. Here, it is much more convenient to use a graphical representation for labeled transition systems instead of giving the binary relations representing the transitions directly (cf. Definition A.1). For instance, Figure 2.3 depicts $[\![m]\!]$.

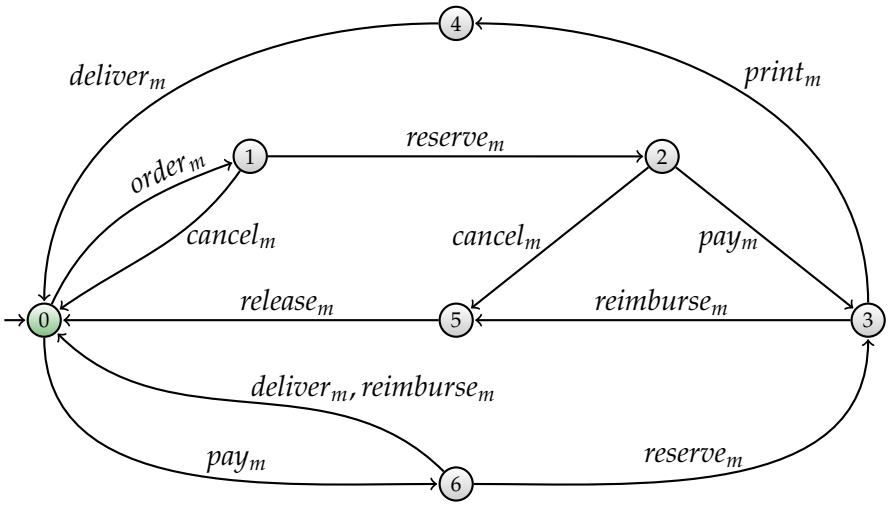


Figure 2.3: Behavior of the management component m

In the graphical representation, we identify a state such as s_m^1 with a circled, gray-colored 1: (1). We omit the name of the component since it is clear from the context in all our examples. An initial state is represented by a similar circle but with green color, e.g., state s_m^0 as (0), and with a sourceless incoming arrow. A transition is depicted in the same manner, e.g., $s_m^0 \xrightarrow{order_m}_m s_m^1$ as (0) $\xrightarrow{order_m}$ (1). Note that different labeled transitions between the same source

and destination state are depicted as one arrow with a comma-separated list of labels, e.g., consider the $deliver_m$- and $reimburse_m$-transitions between states s_m^2 and s_m^0 in Figure 2.3.

We also have to specify the behavior of the other two components and again use our graphical representation for this task: $[\![c]\!]$ is given in Figure 2.4 (a) and $[\![s]\!]$ is given in Figure 2.4 (b).

(a) Behavior of the customer component c **(b)** Behavior of the storage component s

Figure 2.4: Behavior of the customer (a) and the storage component (b)

Observe that the alphabets of the three labeled transition systems $[\![c]\!]$, $[\![m]\!]$, and $[\![s]\!]$ respectively correspond to the action sets A_c, A_m, and A_s respectively of the components as defined on page 18, i.e., all labeled transition systems are valid behaviors with respect to Definition 2.5.

We now completely specified the interaction system for our running example according to Definition 2.5. As a summary, we have (with $Comp$, A_c, A_m, A_s defined on page 18 and Int, Int_{closed} on page 20):

$$\Big(\big(\underbrace{(Comp, \{A_c, A_m, A_s\})}_{\text{component system}}, Int, Int_{\text{closed}} \big), \{ [\![c]\!], [\![m]\!], [\![s]\!] \} \Big)$$

interaction model

interaction system

We refer to this system as Sys_{MMS} in the sequel.

2.1.4 Remarks and Related Models

As demonstrated by our example interaction system Sys_{MMS}, the formalism of interaction systems allows for a very convenient and flexible way of specifying the cooperation among the components. We already discussed the origins of interaction systems in the previous sections, but want to point out that interaction systems are used as the theoretical model of the BIP framework [30, 46] and the Prometheus tool [120], and moreover, the model has been used in the context of the EU project SPEEDS [19, 45]. Next, we take a look at related work using other formalisms.

Various authors have studied ideas that are similar to interaction systems mostly under the aspect of enhancing and facilitating the specification of object-oriented, component-based, or service-oriented systems. Typically, the behavior of the components is specified in a way that distinguishes several system states in which a component offers some of its ports for cooperation. In the literature, various formalisms have been used for this task, e.g., process algebras [5, 8, 39, 176, 206, 207, 212, 228], channel-based methods [12, 13, 24], interface theories and various automata [31, 47, 81, 82, 94, 137, 237, 253], and Petri nets [1, 11, 26, 79], to mention just a few. Typically, a labeled transition system can be seen as a general model for this kind of behavior, i.e., all formalism distinguish behavioral states and transition relations between these states.

Furthermore, other formalisms can often be understood as interaction systems if we abstract from data and input/output (I/O) operations, e.g., the glue code for I/O transition systems [137] consists of binary interactions among the I/O labels of the systems and, of course, I/O transition systems are labeled transition systems. In a similar way, interface automata [80, 82] and constraint automata [24] can be translated where more complex glueing mechanisms can be realized by special glue components in interaction systems. Arnold's synchronous product of transition systems [14] corresponds to interaction systems if we interpret his synchronization vectors as interactions and ensure the disjointness of all alphabets. For 1-safe Petri nets and interaction systems, a bidirectional translation exists [182]. Approaches where the parallel operator of a process algebra is used to derive the glue code [8, 39, 207] can also be regarded as interaction systems by defining appropriate interactions among the synchronizing processes. Similarly, the multiparty synchronization operation of action systems [17], that generalize process algebras (since often, only binary synchronizations are allowed), corresponds to interactions. But, these operations are not as flexible as interactions since the synchronization happens over the same action. As a notable exception, we want to mention the work of Groote et al. [131] on the process algebra mCRL2 where multiactions similar to interactions are possible and where the parallel composition operator together with a special allow operator can be used to model interaction systems if a component's behavior is regarded as a process and only those multiactions that correspond to an interaction are allowed.

Interactions in the model of interaction systems are also similar to first-order multiparty interactions [149] since the set of interactions, that allows to glue components together, can be used in any context where the involved components offer the necessary ports. Furthermore, multiparty rendezvous [59] can be seen as interactions if we abstract statement blocks into atomic actions.

We now discuss some of the above mentioned work in more detail, which shows the similarities to interaction systems. As mentioned in Section 1.2.4 of Chapter 1, the notion of an architecture description language that describes software architectures is related to the idea of component-based development. For instance, Aldini and Bernardo [6]—where similar ideas can be found in work by Shaw and Garlan [242]—stipulate that an architecture description language has to adhere to the following guidelines: (1) separating behavior from topology, (2) specification reuse, (3) interaction elicitation, (4) communication classification, (5) graphical notation, (6) transparent use of the static operators, and (7) the support for architectural styles and system families. In our setting, we do not address all of these items but there are clearly similarities to the model of interaction systems, e.g., we also separate the behavior of the components from their cooperation, we allow for reuse, we distinguish open and closed interactions (and thus, elicit interactions architecturally), support the modeling of many communication scenarios with the flexibility of our interaction model, and provide a graphical notation. We also address the support for architectural styles by constraining the ways the components are allowed to cooperate but postpone this restriction to Chapter 4. For an overview of architecture description languages, we refer the reader to the work by Medvidović and Taylor [192]. Thus, with this connection in mind we also consider formal models for architecture description languages as related work in the following.

PADL The architecture description language PADL by Bernardo et al. [39] and Aldini and Bernardo [6] is based on a process algebra similar to CSP [141]. The separation of topology and behavior is achieved by further extending and structuring the pure usage of the process algebra. In particular, the "double use" of the parallel operator is avoided, i.e., its usage to specify the behavior of processes, which can be understood as the components, and their composition, which is the glue code of the model. The processes, that constitute the building blocks of a system, are described as so-called architectural types, that are sub-divided into a set of architectural element types, an architectural topology, and behavioral variations. Here, an architectural type fixes the name and parameters of a system. The architectural element types describe the behavior of the participating components and are thus specified as processes. The architectural topology specifies the cooperation among the available architectural elements. Behavioral variations allow for global rules that further restrict the cooperations. The reuse of architectural element types is supported by distinguishing between the definition and an instantiation of the architectural element types, i.e., the types are at first formally specified and later instantiated in the topology description of the system. Unique names are provided in the instantiation step which thus supports the reuse of the types.

The actions of the components are called interactions—not to be confused with interactions in our setting—and defined on the architectural element level. They can be distinguished as input, output, and internal actions. On the topology level, the actions can be connected by so-called architectural attachments. These attachments are directed since input and output interactions can be distinguished. In order to support a hierarchical modeling approach, the interactions can be declared to be the interface of the composite system, i.e., they are distinguished as architectural interactions and local interactions where local interactions have to be part of at least one attachment. Here, interactions with the same name cause no problems with respect to name clashing because all interactions are referenced together with the unique element name that was created in the instantiation step.

Java/A Baumeister et al. [31] introduced Java/A as an architecture description language that is closer related to a real programming language, viz. Java in this case. This relatedness should mitigate the discrepancy—also called architectural erosion by Perry and Wolf [226]—between a formal model specified in an architecture description language and its implementation that typically arises if the implementation does not adhere to the rules of the architecture description language and the load-bearing walls—to use a comparison by Perry and Wolf [226]—are removed. Since we are interested in the application of formal methods, we consider in the following only the formal semantic model on which Java/A is based upon. We use the terminology and extensions of the work by Hennicker et al. [137].

The behavior of any entity in Java/A is modeled as an I/O transition system, which is defined similar to interface automata [82]. The I/O operations are distinguished as input, output, and internal operations and all non-internal operations are understood as atomic actions that are used for cooperation. Internal operations are unobservable and only visibly as a special transition of the behavior.

A Java/A component offers ports as its interface for cooperating with its environment. A port consists of a set of required operations and a set of provided operations which are related by means of an I/O transition system. Furthermore, simple components and composite components are distinguished. A simple component has, additional to its ports, a local behavior that is specified as an I/O transition system. A composite component is an encapsulation of further simple or composite components, binary connectors among these components, and relay ports which allow for declaring unused ports of the encapsulated components as the interface of the whole composite component. If two ports are linked by a connector, then the required operations of one port must match the provided operations of the other and vice versa.

SOFA - SOFtware Appliances SOFA is a component model developed by Plášil et al. [229] and extended by Bureš et al. [58] to allow for new features such as dynamic re-configuration. Here, we only focus on the formal part of SOFA but we want to point out that also an implementational part exists and all in the following described entities are available as Java code fragments.

SOFA distinguishes between primitive and composite components where a composite component consists of several sub-components which again can be composite or primitive components. Every component is described with a so-called frame and its architecture. A frame can be understood as a black-box view of the component, i.e., it only defines the provided and required interfaces of the component. The architecture is a gray-box view: On this level, the frame is extended by defining the sub-components and the connections among these under the restriction that only the frames of the sub-components are accessible. The cooperation of the components is realized as special connectors that link the interfaces.

The behavior of a component is specified as so-called behavior protocols [228]. A behavior protocol extends a SOFA component with a regular expression like protocol, which represents traces of the component's behavior that can be observed on the interface of the component. For instance, a component that models a read-only file server which processes open, read, and close requests can ensure by the use of a behavior protocol that any read request of a client component is preceded by an open request. Such a regular expression can, of course, also be understood as a labeled transition system.

Darwin Magee et al. [176] developed Darwin as an architecture description language for distributed software architectures. Each component has a set of services which are distinguished as "required" and "provided" by the component. The name of the service is context independent, i.e., it is not important for the composition. The services are connected by so-called binds which model the cooperation among the components. Hierarchical composition is supported by encapsulation of the components where such an encapsulation is also understood as a component which indicates which of its services are offered to the outside world and which are linked internally. The authors give a model in the π-calculus [202, 203] where services are described as agents. These agents do not specify the behavior, instead they relay requests to the underlying component whose behavior is specified as a further agent or as a finite state process [175, 177]. This separates the services from the local behavior of the components.

Wright Allen and Garlan [8] proposed Wright for the description of a sys-

tem's architecture in three parts. First, they describe the available component- and connector-types where a component-type is defined as a set of ports with a so-called protocol together with the specification of the whole component-type behavior. The ports are the interface of a component-type for its cooperation with the environment and the protocol specifies the behavior of the component-type at the port. A connector-type links cooperation partners by means of a glue code where the particular links to the ports are described in the third part. At this part, the connector-type facilitates the specification of so-called roles which fix the behavior of the participating cooperation partners, i.e., the expected behavior is specified. Now, the glue code specifies how these roles should cooperate. Here, any behavior is modeled as a CSP process [141], i.e., the behavioral specification of the component-types, the protocols of the ports, the roles of the connector-types, and the glue code.

The second part describes the instances of the previously specified types. This instantiation supports the reuse of components and connectors where every instance gets a unique name. Finally, the third part specifies which ports of instantiated components are attached to which roles of instantiated connectors. This attachment ensures the independence from the used CSP actions since they are prefixed with the unique names of the instances. Furthermore, a whole architectural description can be encapsulated as a component where architectural and internal ports can be distinguished.

From the descriptions of the related models, we see that if we abstract from data and I/O operations we often can translate the specification of a given system into an interaction system. For the behavior of the components this is possible since the semantics of process algebraic expressions is usually interpreted as a labeled transition system, e.g., the transitional semantics of CCS [200, Section 2.5]. Several of the above discussed models allow for hierarchical modeling which we have not addressed yet for interaction systems—but discuss in Chapter 3.

In the next section, we show how the overall observable behavior of an interaction system can be derived.

2.2 Deriving the Behavior of Interaction Systems

From the local parts of an interaction system, viz. the behavior of the components, and its interactions we can now derive its overall behavior as a labeled transition system over the interactions. Here, the important distinction between open and closed interactions becomes clear, since we require that closed

interactions are unobservable. In the literature, several names for unobservable labels can be found, e.g., hidden, silent, invisible, or internal. Milner [195] was the first to introduce the Greek letter τ as a result of synchronization in the composition of processes in his research preceding his famous process algebra CCS [197]. Various authors followed this direction since behavior becoming unobservable is an useful way for abstraction, e.g., Bergstra and Klop [37] added silent steps to their process algebra ACP and introduced an abstraction operator that turns visible steps into silent ones. Related ideas can be found in Hoare's CSP [141, Section 3.5]. Such silent steps were also studied for nondeterministic finite automata where special ϵ-transitions correspond to a state change of an automaton without reading an input character, i.e., without an observable state change. As said in Section 1.4 of Chapter 1, we use the following abbreviation to deal with the possibility of unobservable labels: By a superscripted τ we stress the requirement that τ is an element of an alphabet, i.e., $\Sigma^\tau := \Sigma \cup \{\tau\}$ for any alphabet Σ. Similarly, by a superscripted slashed τ ("$\not\tau$") we denote that τ is not part of a set, i.e., $\Sigma^{\not\tau} := \Sigma \setminus \{\tau\}$ for any alphabet Σ.

Here, we proceed with formally defining the *global behavior* of an interaction system. As already mentioned in Chapter 1, this behavior is derived by executing the interactions nondeterministically according to their executability.

Definition 2.6 (Global Behavior): The *global behavior* of an interaction system *Sys* is a labeled transition system $[\![Sys]\!] := (S, Int^\tau_{\text{open}}, \{\xrightarrow{\alpha}\}_{\alpha \in Int^\tau_{\text{open}}}, S^0)$ where the set of *global states* $S = \prod_{i \in Comp} S_i$ is given by the Cartesian product of the components' sets of states, which we consider to be order independent. Global states are denoted by tuples $s = (s_1, \ldots, s_n)$ with $n = |Comp|$, i.e., s_i denotes for $i \in Comp$ component i's state in a global state. The set of *global initial states* is $S^0 = \prod_{i \in Comp} S^0_i$. The family of *global transition relations* $\{\xrightarrow{\alpha}\}_{\alpha \in Int^\tau_{\text{open}}}$ is defined canonically where for all $\alpha \in Int$ and all $s, t \in S$ we have

- $s \xrightarrow{\alpha} t$ if and only if $\alpha \in Int_{\text{open}}$ and for all $i \in Comp$ either $i(\alpha) = \{a_i\}$ for an $a_i \in A_i$ and $s_i \xrightarrow{a_i}_i t_i$ or $i(\alpha) = \emptyset$ and $s_i = t_i$ and
- $s \xrightarrow{\tau} t$ if and only if $\alpha \in Int_{\text{closed}}$ and for all $i \in Comp$ either $i(\alpha) = \{a_i\}$ for an $a_i \in A_i$ and $s_i \xrightarrow{a_i}_i t_i$ or $i(\alpha) = \emptyset$ and $s_i = t_i$.

Typically, we are not interested in the whole labeled transition system that models the global behavior. We are only interested in the set of global states that can be reached from a global initial state (cf. Definition A.4 in Appendix A where we give a formal definition of reachable states in a labeled transition system), which constitutes the reachable part of the global behavior—for convenience, simply called the *reachable global behavior* in the following.

From an algorithmic perspective, we thus want to avoid the construction of the Cartesian product of the components' state spaces to obtain $[\![Sys]\!]$ as specified in Definition 2.6. Instead, we use the following straightforward idea: We only compute the Cartesian product of the components' initial state spaces to obtain the set of global initial states, which are marked as unprocessed. Now, as long as there is an unprocessed global state, we check which interactions are enabled in this global state. The local execution of the corresponding actions of these enabled interactions in the components' labeled transition systems then yields the set of successor states which we add as global states. If such a successors has not been seen before, we also mark it as unprocessed. This idea is known as forward traversal and corresponds to a depth-first or breadth-first search depending on the order in which we process the global states. Here, we do not give an algorithm for this idea but refer to Algorithm B.2 given in Appendix B. In the following, we refer to such an algorithm that computes the reachable part of the global behavior as BEHAVIOR-TRAVERSAL(Sys) (for a given interaction system Sys).

Figure 2.5 depicts the labeled transition system that is computed by calling this algorithm for our running example, i.e., BEHAVIOR-TRAVERSAL(Sys_{MMS}).

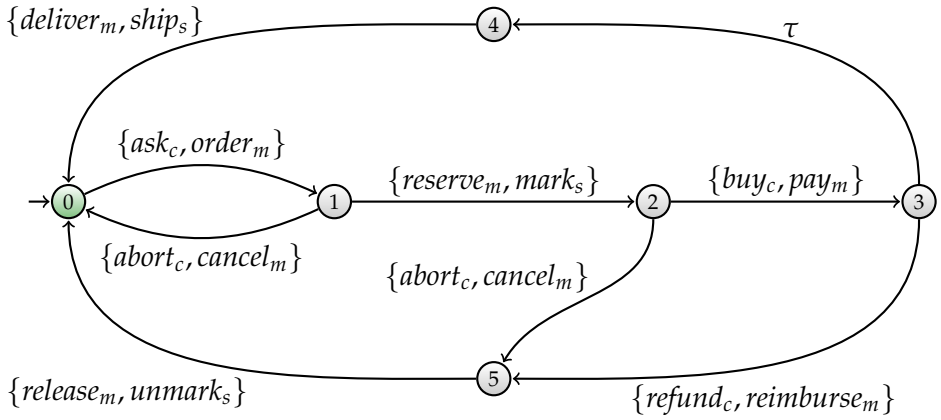


Figure 2.5: Reachable global behavior of the running example Sys_{MMS}. Note that the states represent *global states*, e.g., a circled 1 corresponds to state s^1.

Apparently, the global states convey more information than visible in Figure 2.5, i.e., we have:

$$s^0 = (s_c^0, s_m^0, s_s^0) \qquad s^2 = (s_c^1, s_m^2, s_s^1) \qquad s^4 = (s_c^0, s_m^4, s_s^1)$$
$$s^1 = (s_c^1, s_m^1, s_s^0) \qquad s^3 = (s_c^0, s_m^3, s_s^1) \qquad s^5 = (s_c^0, s_m^5, s_s^1)$$

However, as we discuss in the next section, this additional information is not needed for the properties of interaction systems that we want to study.

For an arbitrary interaction system Sys, the labeled transition system yielded by BEHAVIOR-TRAVERSAL(Sys) is typically different from $[\![Sys]\!]$ as given by Definition 2.6 because the former contains only the global states that are reachable from an global initial state. For instance, the number of reachable global states, as depicted in Figure 2.5, is six, whereas the number of possible global states of the labeled transition system $[\![Sys_{\text{MMS}}]\!]$ is 28 ($|S_c| \cdot |S_m| \cdot |S_s| = 2 \cdot 7 \cdot 2$). However in general, every possible combination of the components' local states could be reachable. Thus, if we estimate the size of the global state space S, we get:

$$|S| = |\prod_{i \in Comp} S_i| = \prod_{i \in Comp} |S_i| \leq \prod_{i \in Comp} |S_{\max}| = |S_{\max}|^{|Comp|}$$

where $|S_{\max}|$ denotes the size of the largest local state space, i.e., $|S_{\max}| = \max\{|S_i| \mid i \in Comp\}$. Here, the reachable global behavior can be exponential in the number of components, and its computation is thus only feasible for small parameters such as our running example Sys_{MMS}.

Note that this phenomenon is, as already mentioned in Chapter 1, well-known as the state space explosion problem [23, 72, 256] since the number of global states is exponential in the number of components. This explosion typically arises in settings where combinatorial many behavioral possibilities exist. We demonstrate this effect for interaction systems by means of an example.

Example 2.7: We model a binary counter as an interaction system in order to demonstrate the effect of state space explosion. Consider the interaction system $Sys_{\text{bin}}(n)$ where for $n \in \mathbb{N}$, the system consists of n components each modeling a binary variable called "bit" in the following. Each bit is initially set to zero, can be increased to one, and can then be decreased to zero. From this behavioral description, we derive for each component a labeled transition system $[\![i]\!]$ with $1 \leq i \leq n$ that is depicted in Figure 2.6.

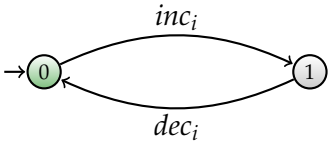


Figure 2.6: Behavior $[\![i]\!]$ of component i representing the ith bit for $1 \leq i \leq n$

We glue the components together such that the interactions model the steps of a binary counter. Figure 2.7 on the facing page depicts the resulting interaction model. For the first bit, a singleton interaction exists, viz. $\{inc_1\}$, because in a binary counter, the first bit can always be increased (if it is zero) to get to the next value of the counter. Observe that any other bit increment depends

on the possibility of being able to decrease all lower bits, e.g., consider the interaction $\{inc_3, dec_2, dec_1\}$ where bit 3 can only be increased if bit 2 and bit 1 are decreased. Finally, if the counter reaches its maximum value, i.e., all bits are set to one, we model the next value, as usual, as an overflow, i.e., the next value is again the lowest one.

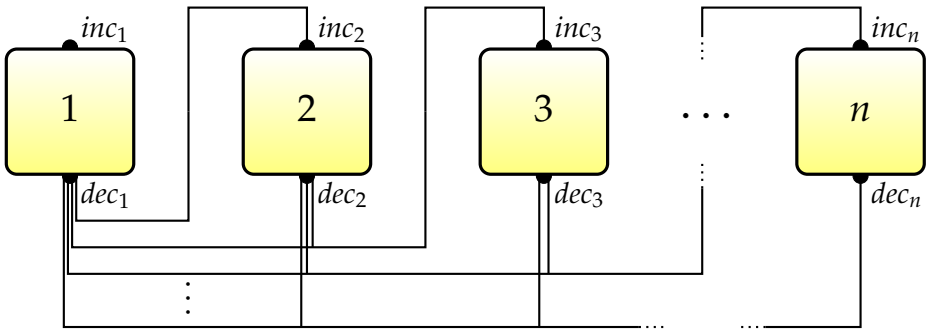


Figure 2.7: Interaction model of the binary counter

Summarizing, we build the following interaction system (where $n \in \mathbb{N}$):

$$Sys_{\text{bin}}(n) = (((Comp, \{A_i\}_{i \in Comp}), Int, Int_{\text{closed}}), \{[\![i]\!]\}_{i \in Comp})$$

with $Comp = \{1, \ldots, n\}$, $A_i = \{inc_i, dec_i\}$ for $i \in Comp$, $Int_{\text{closed}} = \emptyset$, and

$$Int = \bigcup_{2 \leq i \leq n} \{\{inc_i, dec_{i-1}, \ldots, dec_1\}\} \cup \{\{inc_1\}\} \cup \{\{dec_n, \ldots, dec_1\}\}.$$

In order to illustrate the effect of state space explosion, we now consider the global behavior of $Sys_{\text{bin}}(n)$. Since a binary counter with n bits can represent 2^n different numbers, viz. 0 to $2^n - 1$, the global state space S of $[\![Sys_{\text{bin}}(n)]\!]$ consists of 2^n reachable global states. Figure 2.8 depicts (a part of) the global behavior of the binary counter system.

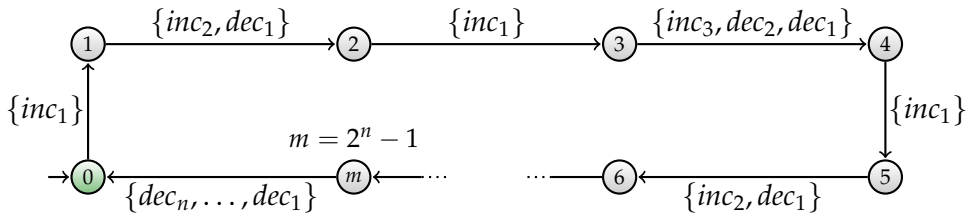


Figure 2.8: Global behavior of the binary counter. Observe that the decimal number of each global state corresponds to a bit string where each bit represents the state of the corresponding bit component, e.g., for $n = 3$ we have $s^6 = (s_3^1, s_2^1, s_1^0)$ and $6_{10} = 110_2$.

If we now want to compare the size of the global behavior with the size of the input, we first have to compute the size of $Sys_{\text{bin}}(n)$. For each $i \in Comp$ we have $|A_i| = 2$, $|S_i| = 2$, and $\sum_{a \in A_i} |\xrightarrow{a}| = 2$, i.e., each component requires constant space. We have $|Comp| = n$ and $|Int| = \sum_{\alpha \in Int} |\alpha| = (\sum_{2 \leq k \leq n} k) + 1 + n = \frac{n(n+3)}{2}$. Thus, the size of $Sys_{\text{bin}}(n)$ is bounded by $O(n^2)$ whereas $|[\![Sys_{\text{bin}}(n)]\!]| = |S| + \sum_{\alpha \in Int} |\xrightarrow{\alpha}| = 2^n + 2^n$ is exponentially large in n, i.e., bounded by $O(2^n)$.

The binary counter example shows that even simple interaction systems suffer from state space explosion because the reachable global state space may correspond to the whole (combinatorial) global state space. Since we are interested in properties of interaction system that address the global behavior, we have to use more sophisticated ways if we want to guarantee polynomial bounds for property verification.

But before that, we have to formally define what properties we have in mind, i.e., we next consider properties of interaction systems more precisely.

2.3 Properties of Interaction Systems

We distinguish three kinds of properties for interaction systems. The first kind, *generic properties*, refers to interaction systems as a whole, i.e., there is no dependence on any particular element of the system (such as a particular component or interaction): The whole system either satisfies the property or not. We consider the properties of deadlock- and livelock-freedom as examples for generic properties.

The second kind, *parametrized properties*, allows for addressing parts of an interaction system for which the property should hold. For instance, we consider the property of progress of a component, i.e., whether the component in question executes transitions locally in any possible execution of the whole system. Characterizing for this kind of properties is that they do not refer to specific elements of an interaction system such as, e.g., the presence of the customer component of our running example Sys_{MMS}. Such properties are only allowed to refer to parameters; hence their name parametrized properties.

The third and last kind, *specific properties*, addresses this issue. These properties are tailored for a specific interaction system and are allowed to refer to its elements. An example for this kind of properties is the question in our running example Sys_{MMS}, whether an interaction modeling a deliver/ship operation is always preceded by a reserve/mark operation.

As we can see from the paragraphs above, the specification of properties in plain text such as the English language is often cumbersome, error-prone, and imprecise, e.g., what is meant by "always preceded"? Therefore, this specification problem has been addressed by many authors and several ways for a formal specification can be found in the literature on program verification such as, e.g., logic, automata, and transition systems. Here, we use a temporal logic called Computation Tree Logic that was introduced by Emerson and Halpern [95] after work by Pnueli [230], Lamport [165], Ben-Ari et al. [33], and Clarke and Emerson [66] that put the original work on temporal logic by Prior [233, 234] in a program verification context. Vardi [258] gives an exhaustive overview on the development of temporal logic for program verification. The use of temporal logic is also interesting for automatic verification since so-called model checking algorithms, which were independently discovered by Clarke and Emerson [66] and Queille and Sifakis [235], are known that algorithmically check whether a given logical formula holds in a certain model of the system under analysis.

Before we formally introduce properties for interaction systems, we want to mention the seminal work of distinguishing system properties as safety and liveness properties originally by Lamport [164] and extended by Alpern and Schneider [9], and also the theorem of the latter authors that every property can be understood as an intersection of a safety and a liveness property [9]. Roughly speaking, a safety property stipulates that something bad never happens, and a liveness property that something good should eventually happen. This distinction has also been adopted for temporal logic [245], and Büchi automata [56] have been used to recognize safety and liveness properties [10]. Pnueli et al. [231] present a compositional proof system addressing safety and liveness properties of synchronous specifications. Often, one can benefit from this distinction and use different approaches for safety and liveness properties, e.g., proof lattices for liveness properties [216], the different strategies for safety and liveness property verification using compositional reachability analysis [62, 63], or the techniques in the model checker SPIN [142]. Biere et al. [41] consider how liveness checking can be understood as safety checking.

However, as pointed out by Naumovich and Clarke [210], this distinction is a tough task that can require deep mathematical insight. Similar observations lead to work on extensions [133] and on new characterizations [87, 210] of the original one by Alpern and Schneider [9]. Here, we do not go further into the details of distinguishing properties but we refer to this characterization when we deal with properties of interaction systems in the following. We want to mention the following alternative (rough but intuitive) characterization: Safety properties are satisfied in infinite time and violated in finite time whereas for liveness properties it is the other way round [23, Sections 3.3 and 3.4].

Next, we introduce the properties *freedom from deadlock* and *freedom from livelock* of interaction systems. Afterwards, we take a closer look at Computation Tree Logic and some of its weaker variants, formally define the logic in our setting, and show how properties of interaction systems can be expressed and verified. Here, livelock- and deadlock-freedom are of special interest for some of the logic's variants, and hence they are introduced beforehand.

2.3.1 Freedom from Deadlock

Deadlocks are well-known throughout all areas of computer science. For instance in the area of operating systems, the two classic introductory books, the one by Tanenbaum [250, Chapter 6] and by Silberschatz et al. [244, Chapter 7], both contain own chapters about deadlocks. Recall the classic example for a deadlock: Processes are waiting for each other because each process holds a resource that another one needs to continue and no process releases its resource.

Here, we consider a similar situation but in the context of interaction systems. Of course, the components can be understood as the processes. However, the components do not hold resources that others need, instead they potentially do not provide needed "resources", viz. actions that are needed to execute interactions, which other components potentially are waiting for. For instance, if a component i wants to execute an action a_i in its current state that is contained in an interaction $\alpha = \{a_i, b_j\}$ where b_j is an action of a component j (distinct from i), but component j is unable to execute b_j in its current state, then j does not provide the "resource" that is needed by i, i.e., interaction α is not enabled in the corresponding global state (cf. Definition A.1). Thus, if each component has to wait for other components because all actions that it wants to execute are contained in interactions that are not enabled (with respect to a global state), we have a deadlock. Next, we formally define this situation where, as formally introduced in Definition A.2 in Appendix A, $\mathrm{Suc}(s)$ denotes the set of successors of a state s.

Definition 2.8 (Deadlock): A *deadlock* in an interaction system *Sys* is a global state $s \in S$ of the global behavior $[\![Sys]\!]$ such that no interaction is enabled in s, i.e., $\mathrm{Suc}(s) = \emptyset$. If no such state is reachable in $[\![Sys]\!]$, we call *Sys deadlock-free*.

The property of deadlock-freedom is an important system property in itself, since deadlocks clearly correspond to faulty design decisions. Moreover, if a previously undetected deadlock occurs in a later software product, the consequences can range from annoyance of users, over financial losses, to catastrophes and disasters.

Note that deadlock-freedom itself is a safety property [216] because it stipulates that a state without a successor must not occur, i.e., something bad never happens, which clearly can be violated in finite time. Furthermore, the verification of safety properties can be reduced to deadlock detection [119] which emphasizes the importance of this property.

Now, we turn to the question of how we can detect a reachable deadlock in an interaction system in an automatic (but naive) way. We use the following idea: While we compute the reachable global behavior of a given interaction system *Sys* with a call to BEHAVIOR-TRAVERSAL(*Sys*) (cf. Algorithm B.2), we can mark the currently considered global state as a deadlock (or output a deadlock warning) if we find no successors. However, the runtime of such an algorithm can be exponential in the number of components because, as we discussed in Section 2.2, the reachable global state space may contain an exponential number of global states. Remember that we already constructed the global behavior of our running example Sys_{MMS} (cf. Figure 2.5), i.e., we already know that Sys_{MMS} is deadlock-free since all reachable global states have at least one successor.

An interesting question that arises now is whether we can do (asymptotically) any better than this naive approach. This research direction typically leads to the question of the computational complexity of a (decision) problem which has been addressed for interaction systems as well. We want to mention the books of Garey and Johnson [108] and Papadimitriou [220] as excellent references for computational complexity, and we use their definitions of complexity classes here. As discussed above, we can assume that the problem of deciding whether an interaction system is deadlock-free (called *deadlock detection* for short in the following paragraphs) is contained in the class EXPTIME.

The first result regarding interaction system and deadlock-freedom was given by Minnameier [204] where the author presents a polynomial-time reduction of the well-known 3-satisfiability problem, which is NP-complete [74], to deadlock detection. This establishes the NP-hardness of deadlock detection but also opens the possibility that we can find an algorithm which only needs nondeterministic polynomial time. However, this complexity gap was later closed by Majster-Cederbaum and Minnameier [182] which show that deadlock detection is PSPACE-hard by a polynomial-time reduction of the decision problem of deadlock-freedom in 1-safe Petri nets to deadlock detection. Since the former problem was shown to be PSPACE-hard by Cheng et al. [60], the PSPACE-hardness of deadlock detection follows. Later, Majster-Cederbaum and Minnameier [183] addressed the question whether deadlock detection is also contained in the complexity class PSPACE, and since they answer this question affirmative (via a sequence of a polynomial-time reductions to a

problem that is shown to be in PSPACE), we can finally state that deadlock detection is PSPACE-complete. A more detailed discussion can be found in the dissertation of Minnameier [205].

For our naive deadlock detection approach, the PSPACE-completeness result shows that its strategy is not space-optimal, since it uses the whole representation of the reachable part of the corresponding labeled transition system for investigation. Clearly, this representation possibly uses more than polynomial space. However, as pointed out by Valmari [256, Section 5.4] in the context of Petri nets, the result that a problem can be solved theoretically in polynomial space often sacrifices a significant amount of time for this space bound, and in practice, the exponential-time algorithms are much faster. The reason for this much worse time bound lies in the usual way to show that the problem is contained in PSPACE: To give an algorithm that decides the problem in polynomial space, authors typically give a nondeterministic algorithm that occupies only a polynomial amount of space, which proves the problem is in NPSPACE, and apply the famous result of Savitch [240] that PSPACE = NPSPACE. This way is also used by Majster-Cederbaum and Minnameier [183] to show that a certain problem, to which deadlock detection is reducible, can be decided in polynomial space. However, Savitch's translation does not yield an algorithm that is useful in practice.

We do not want to continue here with a space-optimal algorithm for deadlock-freedom in interaction systems, since its runtime would still not be polynomial in the size of the input—which is just the representation of the interaction system—unless the complexity classes mentioned above collapse. However, in order to demonstrate the above mentioned issues, we give a polynomial-space algorithm for deadlock-freedom in Section C.1 of Appendix C and refer the interested reader to this section. Next, we take a look at a similar problem, viz. freedom from livelocks, that was—to the best of our knowledge—not considered in the context of interaction systems before.

2.3.2 Freedom from Livelock

Livelocks are conceptually related to deadlocks but typically describe a different situation: A livelocked system is not stuck at a specific point (or global state in interaction systems); however, there is some internal activity that keeps the system going but an outside observer cannot perceive a change in behavior. According to Kwong [159], the term "livelock" was coined by Ashcroft [15] in work about an airline reservation system where a booking is never finished although the system is proven to be deadlock-free. In the literature, there are many definitions of a livelock, even inconsistent ones [138].

For instance, Tanenbaum [250, Section 6.7.3] gives an example where two processes use busy-waiting to acquire two resources, and although no one succeeds in getting both resources (the first process always precedes the second one in requesting one of the resources and vice versa), the processes do not block each other since their restart their polling requests over and over again. Hoare [141, Section 4.4.3] considers livelocks as situations where processes communicate with each other but never with the external world. Here, we define a *livelock* and the notion of *livelock-freedom* as follows.

Definition 2.9 (Livelock): A *livelock* in an interaction system Sys is a global state $s \in S$ such that no open interaction is enabled in s and in any global state $t \in S$ that is reachable from s by the execution of closed interactions and such that s itself is reachable (again) by the execution of closed interactions, i.e., $s \xrightarrow{\tau}{}^{+} s$ and $\forall\, t \in S \colon s \xrightarrow{\tau}{}^{*} t \implies \mathrm{Suc}(t, Int_{\mathrm{open}}) = \emptyset$. If no such state is reachable in $[\![Sys]\!]$, we call Sys *livelock-free*.

Contrary to deadlocks, livelock-freedom is a liveness property [216] because in a livelock-free system it is always possible that something good happens eventually which here is the occurrence of an open interaction, i.e., the property can be satisfied in finite time. We want to point out that in an interaction system's reachable global state which has a τ-self-loop and an outgoing non-τ-transition, a path or global execution fragment is observable that merely consists of τ-transitions. However, it is always possible to leave this path and continue with an open interaction, i.e., the system is not prevented from performing particular actions [171], which here are the open interactions. Manna and Pnueli [189, pages 325–326] describe this property as follows: The system does not stay constrained forever within a given range of locations. Thus, in Definition 2.9 we took a similar view, i.e., the (always given) possibility of the occurrence of an open interaction renders a system livelock-free.

For our running example Sys_{MMS}, we again take a look at the reachable global behavior depicted in Figure 2.5 (cf. page 31). Since state number 3 is the only one with an outgoing τ-transition but its successor, state number 4, has an outgoing non-τ-transition, the livelock-freedom follows. However, for more complex systems an automatic investigation procedure is needed. We can use the following idea: We compute the reachable global behavior, remove all states in which a non-τ-transition is enabled, and check for a (directed) cycle among the remaining states. Here, we do not give a detailed algorithm but included one in Appendix B as Algorithm B.3. Clearly, the runtime of this algorithm is not polynomial because we compute the global behavior.

As for the question of deadlock-freedom in Section 2.3.1, we are interested in the optimality of algorithms for livelock-freedom, i.e., whether we can do

better than computing the global behavior beforehand. This again leads us to the computational complexity of the associated decision problem. Next, we show that we can reduce the question of deadlock-freedom in interaction systems to livelock-freedom, which implies the PSPACE-hardness of the problem.

Lemma 2.10 (Deadlock-Livelock Mapping): Let Sys be an interaction system. Sys is deadlock-free if and only if Sys' is livelock-free where Sys' is constructed in constant time as follows. Let x be a fresh component (not contained in Sys) with a single action $live_x$ and a local behavior that consists of one initial state with a self-loop labeled by the action. We set $Comp' = Comp \cup \{x\}$, and we keep all action sets and local behaviors of the components for Sys'. We also keep the set of interactions and add x's action as a singleton, i.e., $Int' = Int \cup \big\{\{live_x\}\big\}$, but set $Int'_{\text{closed}} = \big\{\{live_x\}\big\}$—only the new interaction is closed in Sys'.

A formal proof of Lemma 2.10 can be found in Appendix F on page 237.

Lemma 2.10 shows that if we were able to answer the question of livelock-freedom efficiently, we can use this technique to decide the question of deadlock-freedom for a given interaction system in an efficient way. Since we learned in the previous section that the latter question already is PSPACE-hard (as proven by Majster-Cederbaum and Minnameier [183]), this result carries over to livelock-freedom, i.e., the reduction implied by Lemma 2.10 establishes the PSPACE-hardness of livelock detection.

Moreover, we know from Algorithm B.3, which we shortly discussed above, that the problem of livelock-freedom is contained in the class EXPTIME. So the next natural question is—as we discussed for the property of deadlock-freedom in Section 2.3.1—whether we can also check for livelocks in polynomial space, which would establish the PSPACE-completeness of the problem. We answer this question affirmative but refer the interested reader to Section C.2 of Appendix C for the details and the corresponding algorithm.

Now, after discussing a safety and a liveness property, we turn to a better way of specifying properties by means of logical formulae.

2.3.3 Using Logical Formulae for Property Specification

As mentioned in the introduction to Section 2.3, we want to use Computation Tree Logic (CTL) to specify properties of interaction systems. For convenience, we skip here a formal definition of the logic and refer the reader to Appendix D where we define the syntax (cf. Definition D.1) and semantics (cf.

Definition D.4) of CTL*. Note that the superscripted star denotes the extended version of Computation Tree Logic as defined in Appendix D. For a more detailed introduction to CTL*, we refer the reader to the books of Baier and Katoen [23, Chapter 6] and Clarke et al. [72, Chapter 3].

One important aspect of CTL* is its interpretation in a model called Kripke structure, which is similar to a labeled transition system but instead of labeling the transitions, states are equipped with labels. We refer the reader to Definition D.3 where we formally define Kripke structures.

Since we want to use the logic CTL* in our setting, we now need a way to interpret CTL* formulae in labeled transition systems. De Nicola and Vaandrager [84] show such a way that allows for this interpretation: A translation between Kripke structures and labeled transition systems. Here, we use their linear version [85] and "condense" the translation which originally introduced an intermediate step to allow a translation in both directions. Historically, the translation was first proposed by Emerson and Lei [96], modified to deal with unobservable actions by De Nicola and Vaandrager [84], and later refined by the latter authors to be linear in the number of states and transitions [85]—to be precise, the translation originally only allows for the interpretation of a fragment of CTL* but results by Browne et al. [53] (that equivalent[2] Kripke structures satisfy the same CTL* formulae) and by Reniers and Willemse [238] (that this equivalence is preserved by the translation) show that the translation is also useful for the interpretation of all CTL* formulae.

Next, we define the one-way translation as discussed above (cf. Definition A.1 for labeled transition systems and Definition D.3 for Kripke structures).

Definition 2.11 (Labeled Transition System to Kripke Struct. Translation): A given labeled transition system $LTS = (S, \Sigma, \{\xrightarrow{a}\}_{a\in\Sigma}, S^0)$ is translated via the function $\mathfrak{K}(\cdot)$ into a Kripke structure $\mathfrak{K}(LTS) := (S', \longrightarrow, S^0, \Sigma^\tau, \mathcal{L})$ with:

$$
\begin{aligned}
S' &= S \cup \{(s,a,t) \in S \times \Sigma \times S \mid s \xrightarrow{a} t \ \wedge \ a \neq \tau\},\\
\longrightarrow &= \{(s,t) \in S \times S \mid \tau \in \Sigma \ \wedge \ s \xrightarrow{\tau} t\}\\
&\qquad \cup \{(s,(s,a,t)) \in S \times (S \times \Sigma \times S) \mid s \xrightarrow{a} t \ \wedge \ a \neq \tau\}\\
&\qquad \cup \{((s,a,t),t) \in (S \times \Sigma \times S) \times S \mid s \xrightarrow{a} t \ \wedge \ a \neq \tau\},\\
\mathcal{L}(s) &= \{\tau\} \text{ for all } s \in S, \text{ and}\\
\mathcal{L}((s,a,t)) &= \{a\} \text{ for all } s,t \in S \text{ and all } a \in \Sigma \text{ with } s \xrightarrow{a} t \text{ and } a \neq \tau.
\end{aligned}
$$

We refer the reader to the above mentioned work [84, 85, 96, 238] for a more detailed discussion of the translation. We want to point out that also an action based version of CTL* exist—discussed in the work of De Nicola and

[2] This equivalence is known as bisimilarity which we introduce in Section 2.4.

Vaandrager [83]—but here stick to the state-based interpretation as it is more common, used in references that we cite in the following, and can efficiently be applied to interaction systems with the translation of Definition 2.11 as we address now by an extension of the satisfaction relation for CTL* (cf. Definition D.4).

Definition 2.12 (LTS Satisfaction Relation for CTL*): For a labeled transition system LTS and a CTL* state formula Φ, we write $LTS \overset{\forall}{\models} \Phi$ if and only if $\forall\, s^0 \in S^0 \colon \mathfrak{K}(LTS), s^0 \models \Phi$, and we write $LTS \overset{\exists}{\models} \Phi$ if and only if $\exists\, s^0 \in S^0 \colon \mathfrak{K}(LTS), s^0 \models \Phi$. In the case of only (exactly) one initial state, i.e., $|S^0| = 1$, the two relations coincide and we simply write $LTS \models \Phi$.

As an example for the translation, we consider our running example Sys_{MMS}. Figure 2.9 depicts the transformation applied to Sys_{MMS}'s reachable global behavior (cf. Figure 2.5 on page 31).

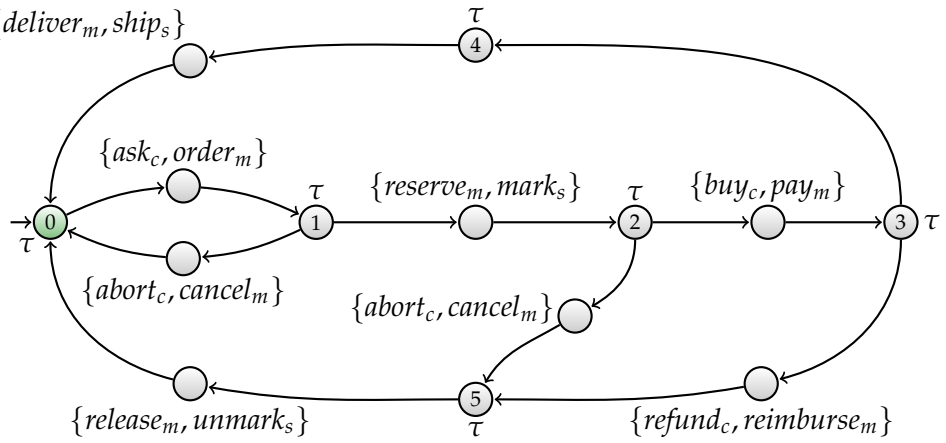


Figure 2.9: The labeled transition system to Kripke structure transformation applied to our running example Sys_{MMS}. Note that the original states kept their naming and the new (intermediate) states are numbered with their source, label, and destination information. Every state has exactly one label (actually this should be a singleton at every state, but we omitted the set braces).

Now, we take a look at how to encode freedom from deadlock and freedom from livelock as CTL* formulae. We define:

$$\Psi_{\text{dlf}} := \mathsf{AG}\,\mathsf{EX}\,\top \quad \text{and} \quad \Psi_{\text{llf}} := \mathsf{AG}\big(\mathsf{EX}\,\top \Rightarrow \mathsf{EF}\big(\textstyle\bigvee_{\alpha \in Int_{\text{open}}} \alpha\big)\big).$$

Note that we omit the small space (cf. Definition D.1) between the quantifiers and the temporal operators for better readability in the sequel.[3] The formula

[3] This omission becomes particularly relevant for an important fragment of CTL*, called CTL, that we discuss in the next paragraph.

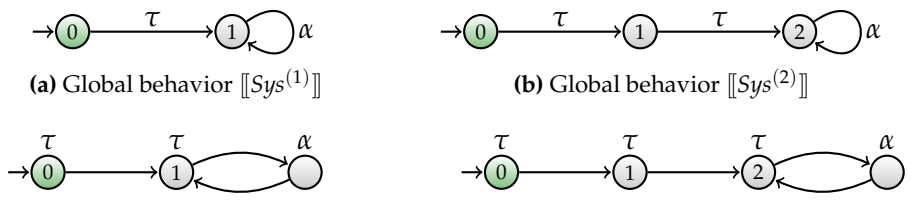
Ψ_{dlf} states that all reachable states on all maximal paths have a direct successors. Similarly, formula Ψ_{llf} states that all reachable states on all maximal paths that have a direct successor also have a (not necessarily direct) successor that is labeled with an open interaction. Observe that these formulae correspond to deadlock- and livelock-freedom as given by Definitions 2.8 and 2.9 respectively. We already showed that this is the case for our running example, we can thus conclude that $[\![Sys_{\text{MMS}}]\!] \models \Psi_{\text{dlf}}$ and $[\![Sys_{\text{MMS}}]\!] \models \Psi_{\text{llf}}$ holds (cf. Definition 2.12).

In the next paragraph, we deal with the question of how to algorithmically check whether a formula holds in a system—this procedure is known as *model checking*. We also introduce important fragments of the logic from the literature that are interesting in our setting.

Important Fragments of CTL* and Model Checking

The fragment CTL*–X of CTL* is defined by omitting the (next) temporal operator "X" in Definition D.1. This fragment is important for abstraction because the "X"-operator is incompatible with abstraction from internal activity [116, Section 1]. Lamport [166, Section 2.3] discussed this issue in detail; the following quote pinpoints his critique: "When one talks about the next state, one really means the next state in which a significant change occurs—where significant means visible at the level of detail of the specification." We illustrate this issue by a small example.

Example 2.13: We consider the two labeled transition systems depicted in Figure 2.10 (a) and Figure 2.10 (b) that represent the global behaviors of two interaction systems $Sys^{(1)}$ and $Sys^{(2)}$ with closed interactions where α denotes the single open interaction in each case. We do not specify these interaction systems here, since the point we want to illustrate only affects the global behaviors and properties specified in CTL*.



(a) Global behavior $[\![Sys^{(1)}]\!]$ **(b)** Global behavior $[\![Sys^{(2)}]\!]$

(c) Kripke structure $\mathfrak{K}([\![Sys^{(1)}]\!])$ **(d)** Kripke structure $\mathfrak{K}([\![Sys^{(2)}]\!])$

Figure 2.10: Global behaviors and corresponding Kripke structures

Now, we consider the CTL* formula $\Psi = \mathsf{EX}\,\mathsf{X}\,\alpha$ stating that a maximal path

exists where α holds for the second element (cf. Definition D.4). We find out that $Sys^{(1)} \models \Psi$ and $Sys^{(2)} \not\models \Psi$ holds—the corresponding Kripke structures (cf. Definition 2.11) are depicted in Figure 2.10 (c) and Figure 2.10 (d) respectively. Clearly, both systems satisfy the CTL* formula $\mathsf{EF}\,\alpha$ stating that a maximal path exists where α holds eventually. However from a property specification point of view, the exact point or the number of steps when/before α holds should be irrelevant since we cannot foresee how the system is constructed such that it satisfies relevant or meaningful properties, e.g., both systems $Sys^{(1)}$ and $Sys^{(2)}$ satisfy the same set of CTL*–X properties, i.e., properties specified in Extended Computational Tree Logic without the next temporal operator. This relevance is also the main critique of the next temporal operator by Lamport [166].

A further advantage identified by Lamport [166, Section 4.2] of omitting a next temporal operator (interpreted as the next point in time) is that it does not matter for property specification by means of temporal logic whether time is considered as discrete or continuous in the underlying models. However, since we only consider discrete event systems in this thesis—note that steps of a labeled transition system, e.g., states on execution paths, are discrete events—we do not further discuss this issue here. But, we also omit the next temporal operator of CTL* for the reasons mentioned in Example 2.13.

Another important fragment of CTL* is simply called CTL (without the superscripted star) and can be defined such that every temporal operator is quantified by exactly one path quantifier, e.g., $\mathsf{AF}\,\Phi$ is a valid CTL formula whereas $\mathsf{AFG}\,\Phi$ is not—for an arbitrary CTL formula Φ. Similarly, the fragment CTL–X of CTL*–X can be defined. Observe that we said "can be defined" since this fragment was invented by Clarke and Emerson [66] previously to CTL*, which was introduced by Emerson and Halpern [95]. The fragment CTL is of special interest because the question wether a given Kripke structure KS and a state $s \in S$ of this structure satisfy a given CTL formula Φ, i.e., whether $KS, s \stackrel{?}{\models} \Phi$ holds (cf. Definition D.4), is decidable with an algorithm by Clarke et al. [68] in linear time in the product of the sizes of the parameters, i.e., in $\mathrm{O}(|KS| \cdot |\Phi|)$ where $|KS|$ is the sum of the number of states and transitions of the Kripke structure KS (cf. Definition D.3) and $|\Phi|$ is the number of operators in the formula Φ (cf. Definition D.1). Note that if we are interested in the asymptotic length $\mathrm{O}(|\Phi|)$ of a CTL* formula Φ, the occurrence of the additional CTL* operators of Definition D.2 does not play a role because they add only constant length, i.e., we can count them as one instead of considering their expansion [23, Remark 5.5, page 235].

The above discussed satisfiability question and the way to answer it algorithmically was independently invented by Clarke and Emerson [66] and

Queille and Sifakis [235] and has become famous as *model checking*, which now generally refers to the automatic verification technique that checks whether a certain model, which is the specification of a system, satisfies a certain logical formula, which is a property of the system. A model checking algorithm for CTL* was derived by Emerson and Lei [97] with running time $O(|KS| \cdot 2^{|\Phi|})$ and, together with a complexity result by Sistla and Clarke [246] that shows (as a consequence of a weaker logic) that CTL* model checking is PSPACE-complete, no efficient algorithm (as in the case for CTL) can be expected. However from a practical perspective and in our setting, the formulae are not the problem for the efficiency of model checking algorithms: The problem is the size of the Kripke structure.

This problem is directly connected to the state space explosion problem that also occurs in interaction system as we illustrated in Example 2.7. Thus in the following, we pay attention to the direct application of model checking, i.e., if we want to verify a CTL*–X property for an interaction system, we search for ways that avoid constructing the global behavior.

When we introduced logical formulae for deadlock- and livelock-freedom above (cf. page 42), we used the next temporal operator for both formulae. As we stated in this section, we want to avoid its usage and thus have to take a look at an alternative characterization.

Freedom from Deadlock and Livelock as CTL*–X Formulae

Consider the following CTL–X formula:

$$\Psi_{\text{lock}} := \mathsf{AG}\,\mathsf{EF}\big(\bigvee_{\alpha \in Int_{\text{open}}} \alpha\big).$$

The formula stipulates that in all reachable states there is a maximal path starting in this state where eventually an open interaction occurs—more precisely where a state can be reached that is labeled with an open interaction. What does this formula mean? Clearly, a system whose global initial state is a deadlock does not satisfy Ψ_{lock}. However, we know that starting from a livelock only τ-transitions are observable and thus no open interaction occurs, i.e., a system whose global initial state is a livelock also does not satisfy Ψ_{lock}. Thus, the formula cannot distinguish deadlocks and livelocks in interaction systems.

Interestingly as pointed out by Van Glabbeek et al. [117], there is no CTL*–X formula that is able to provide this distinction. A similar observation was made by Kaivola and Valmari [151, Definition 3.3] for another temporal logic that in the absence of a next temporal operator one needs to define a new operator for the task of distinguishing a finite sequence from an infinite one,

which boils down to distinguishing deadlocks and livelocks. Note that the fact that deadlock-freedom cannot be distinguished from livelock-freedom in CTL*–X is not caused by the fact that τ is not an atomic proposition. The problem is caused by the absence of the next operator "X".

Here, we avoid this drawback by stressing the importance of deadlock-freedom in interaction systems. This means that we always establish the deadlock-freedom of an interaction system before we apply model checking of CTL*–X formulae. This avoids the drawback and moreover, we can then use the formula Ψ_{lock} to express livelock-freedom in CTL*–X for deadlock-free interaction systems since we have (cf. Definition D.5):

$$\begin{aligned}\Psi_{\text{dlf}} \wedge \Psi_{\text{llf}} &\equiv \mathsf{AG}\,\mathsf{EX}\,\top \wedge \mathsf{AG}\big(\mathsf{EX}\,\top \Rightarrow \mathsf{EF}\big(\textstyle\bigvee_{\alpha \in Int_{\text{open}}} \alpha\big)\big)\\ &\equiv \mathsf{AG}\,\mathsf{EX}\,\top \wedge \mathsf{AG}\,\mathsf{EF}\big(\textstyle\bigvee_{\alpha \in Int_{\text{open}}} \alpha\big)\\ &\equiv \Psi_{\text{dlf}} \wedge \Psi_{\text{lock}}.\end{aligned}$$

Next, we continue with parametrized properties of interaction systems.

2.3.4 Various Known Parametrized Properties

In this section, we consider properties of interaction systems that refer to certain parts of a system such as components or interactions. However, these properties have in common that they do not refer to a specific component such as the presence of the management component of our running example Sys_{MMS}. Instead, they offer a parameter that can be set for any valid interaction system.

Majster-Cederbaum and Minnameier [183, Section 2.2] introduce various such properties for interaction systems and study the computational complexity of their decision procedures. For instance, the property of *progress* of a certain component in an interaction system stipulates that actions of the component occur infinitely often in every run starting from a global initial state where a run is an infinite transition sequence. Martens [190, Definition 2.2.2] defines this property as: Actions of the component in question participate in every run starting from any reachable state where participation means that an interaction occurs where one of the actions is contained in [190, Definition 2.1.2]. A similar definition is given by Majster-Cederbaum et al. [188, Definition 3.4] under the name "liveness" (not to be confused with the more general liveness in liveness properties by Alpern and Schneider [9]). Gössler et al. [124, Definition 7] define that a component may progress if any run can be continued in a way such that the component participates in an occurring interaction, although as admitted later [190, Section 2.3], the name "may progress" should not be confused with

the other definitions of progress in interaction systems. All authors require that the interaction system in question is deadlock-free. Since systems with unobservable transitions are not considered in the referenced work, we now could adjust these definitions and give extended definitions that are compliant with our setting and derive a notion of observable progress.

However here, as already mentioned at the beginning of Section 2.3, we want to get rid of informal text descriptions serving as property specifications, but also want to avoid several definitions that are needed to precisely fix the meaning of a property. Moreover, it is interesting to ask why we need to explicitly require the deadlock-freedom in the definition of progress, although, as we pointed out at the end of the previous section, we should establish the deadlock-freedom of a system before we apply model checking. We express (observable) progress as the following CTL–X formula where i is a component parameter:

$$\Psi_{\text{progress}}(i) := \mathsf{AG}\,\mathsf{AF}\big(\bigvee_{\alpha \in Int_{\text{open}} \wedge\, i(\alpha) \neq \emptyset} \alpha\big).$$

Now, a component i of an interaction system Sys makes (observable) progress if $[\![Sys]\!] \models^{\forall} \Psi_{\text{progress}}(i)$ (cf. Definition 2.12). Note that this completes the definition of progress with respect to the work referenced above, which shows the power of using a temporal logic. Nevertheless, we explain the idea behind the property. The formula stipulates that in all reachable global states it holds for all maximal paths starting in this state that eventually an open interaction occurs where component i participates in. Note that a system whose global initial state is a deadlock (or similarly, if a deadlock is reachable in an interaction system) does not satisfy this property because on the single maximal path of the corresponding Kripke structure, that merely consists of the deadlocked state, no state is reached that is labeled with an open interaction. Moreover, if all interactions in which component i participates in are closed interactions, the disjunction in the formula $\Psi_{\text{progress}}(i)$ is empty which corresponds to $\bot$ (cf. Definition D.2), i.e., in this case the whole formula is not satisfied by any interaction system since we required that at least one global (initial) state is present.

We want to point out that the formulae for properties that refer to a component of an interaction system, e.g., progress, are first completely generated, i.e., the disjunction of the corresponding interactions is first carried out before the property is checked. This is important since the property may only consist of atomic propositions that are open interactions of the underlying interaction system.

But, this is not a restriction since we can also add special atomic propositions that refer to, e.g., the participation of a certain component. For instance

regarding the progress property above, we can modify the translation to a Kripke structure (cf. Definition 2.11) and add for each global state the set of components that participate in the corresponding interaction, i.e., the label function becomes $\mathcal{L}((s, \alpha, t)) = \{\alpha\} \cup \text{compset}(\alpha)$ (for a global state s it stays the same). Then, the progress property simply becomes $\Psi'_{\text{progress}}(i) := \mathsf{AG}\,\mathsf{AF}\,i$ for $i \in \mathit{Comp}$. However, this syntactical sugar only shortens formulae and we thus stick to our original definition/translation of CTL*.

We continue with two further known parametrized properties. First, we consider the *availability* of a component. Majster-Cederbaum and Minnameier [183, Section 2.2] define it as: On every run starting in a global initial state of a deadlock-free interaction system—where a run is, as mentioned above, an infinite sequence of transitions—actions of the component in question are contained in an enabled interaction infinitely often, which means that the next global state could be a state that is reached via an interaction where the component participates in. Actually, here we have to say the next *observable* global state since we cannot directly address the next global state if we use CTL*–X. We thus express (observable) availability as the following CTL–X formula where i is a component parameter:

$$\Psi_{\text{available}}(i) := \mathsf{AG}\,\mathsf{AF}\,\mathsf{E}\big((\bigwedge_{\alpha \in \mathit{Int}_{\text{open}}} \neg\alpha)\;\mathsf{U}\;(\bigvee_{\alpha \in \mathit{Int}_{\text{open}} \wedge\, i(\alpha) \neq \emptyset} \alpha)\big).$$

Now, a component i of an interaction system Sys is (observable) available if $[\![\mathit{Sys}]\!] \overset{\forall}{\models} \Psi_{\text{available}}(i)$. The formula stipulates that in all reachable global states it holds for all maximal paths starting in this state that eventually a global state is reached in which a maximal path starts on which only closed interactions, i.e., τ-labels, occur until an open interaction occurs where component i participates in.

Second, we consider whether a component is *live* where we use the definition of Minnameier [205, Definition 2.25]. Note that this property is also specified by Gössler et al. [124, Definition 7] under the misleading name "progress". The property states for a component of an interaction system that we can always execute a sequence of interactions from every reachable global state such that eventually an interaction is executed where the component participates in. Again, we have to adjust this idea to be conform with unobservable behavior, and thus express whether is a component is (observable) live as the as the following CTL–X formula:

$$\Psi_{\text{live}}(i) := \mathsf{AG}\,\mathsf{EF}(\bigvee_{\alpha \in \mathit{Int}_{\text{open}} \wedge\, i(\alpha) \neq \emptyset} \alpha).$$

Now, a component i of an interaction system Sys is (observable) live if $[\![\mathit{Sys}]\!] \overset{\forall}{\models} \Psi_{\text{live}}(i)$. The formula stipulates that in all reachable global states there is a maximal path starting in this state where eventually an open interaction occurs where component i participates in.

We want to mention that all three formulae $\Psi_{\text{progress}}(i)$, $\Psi_{\text{available}}(i)$, and $\Psi_{\text{live}}(i)$ (where $i \in \mathit{Comp}$) are CTL–X formulae, i.e., we did not need the whole expressive power of Extended Computational Tree Logic without the next temporal operator. This means for model checking these properties, that the check can be carried out, as mentioned in the previous section, in $O(|KS| \cdot |\Phi|)$ [68]. Since the translation of the global behavior into a Kripke structure is linear (cf. Definition 2.11) and we have $|\Psi_{\text{progress}}(i)| \in O(|Int_{\text{open}}|)$ for a component $i \in \mathit{Comp}$ (the same holds for $\Psi_{\text{available}}(i)$ and $\Psi_{\text{live}}(i)$), the driving parameter for checking these properties is the size of the global behavior. We already know that the construction of this global behavior is exponential in the size of the input, which means that we cannot hope for a naive verification approach in polynomial time. This is supported by the results of Majster-Cederbaum and Minnameier [183] and Minnameier [205] who show that the corresponding decision problems for the three above discussed properties are PSPACE-hard.

Next, we take a look at specific properties, i.e., properties that refer to a specific interaction system.

2.3.5 Specific Properties

Specific properties are tailored for a particular interaction system, i.e., the atomic propositions are the set of open interactions of this particular system. For instance, we can ask for our running example Sys_{MMS}: Is it possible that the management component m executes its action $\mathit{deliver}_m$ before its action $\mathit{reserve}_m$ as corresponding interactions seen from the global initial state? In other words, is the interaction $\{\mathit{deliver}_m, \mathit{ship}_s\}$ preceded by the interaction $\{\mathit{reserve}_m, \mathit{mark}_s\}$? For this question, we can verify the following CTL*–X property in $[\![Sys_{\text{MMS}}]\!]$:

$$\Psi = \mathsf{A}(\mathsf{F}\,\{\mathit{deliver}_m, \mathit{ship}_s\} \Rightarrow \neg\{\mathit{deliver}_m, \mathit{ship}_s\}\,\mathsf{U}\,\{\mathit{reserve}_m, \mathit{mark}_s\}).$$

The formula stipulates that in all maximal paths if eventually the interaction $\{\mathit{deliver}_m, \mathit{ship}_s\}$ occurs, then $\{\mathit{deliver}_m, \mathit{ship}_s\}$ does not occur on the path before $\{\mathit{reserve}_m, \mathit{mark}_s\}$ occurs. Now, we can use a model checker which yields $[\![Sys_{\text{MMS}}]\!] \models \Psi$, i.e., interaction $\{\mathit{deliver}_m, \mathit{ship}_s\}$ is preceded by interaction $\{\mathit{reserve}_m, \mathit{mark}_s\}$ as seen from the global initial state of the system. However, as mentioned above, the model checker needs to construct the global behavior of the interaction systems and this construction potentially exceeds the available memory because of the state space explosion problem.

We come back to this issue in later chapters and show how to avoid this construction. But before, we introduce equivalences for interaction systems.

2.4 Behavioral Equivalence in Interaction Systems

We already mentioned several times that we need a notion to compare the behavior of interaction systems, i.e., something like a behavioral equivalence. In this section, we address this issue.

The identification of equivalent behavior is beneficial in many aspects. For instance, once a set of properties is verified for a certain system, we expect that this set also holds for systems that exhibit an equivalent behavior. A similar aspect is the question of abstraction, i.e., whether we can safely abstract from system parts that are not of interest in our current verification step and prove that properties that hold in the abstracted version also hold in the original system—since the abstraction and the original are behavioral equivalent. The key question here is: What is exactly meant by equivalent behavior?

Typically, by equivalence we mean an equivalence relation on the behavior. Here, this boils down to which states of given labeled transition systems—which we use to describe behavior—can be related such that they cannot be distinguished by observation. These kind of relations were addressed in many fields of research, e.g., equivalence of regular languages in automata theory [145, Section 4.4] or bisimilarity in Milner's process algebra CCS [197, Chapter 4]—in fact, Milner was the first to use equivalences to compare models of concurrency at different levels of abstraction [141, Section 7.4.1].

In the context of sequential systems and concurrency semantics (which also applies here), Van Glabbeek [112, 113] studied suitable relations in an exhaustive manner. In his work, one can find several different semantics and dozens of behavioral equivalences. Here, we focus on five of these equivalences that are important for our considerations and are compliant with our following goals: We want to pick a behavioral equivalence that preserves an interesting set of properties, abstracts from negligible parts of the behavior (especially unobservable parts, i.e., τ-transitions), and can be computed efficiently, i.e., in polynomial time. The last point excludes all trace equivalences, which typically correspond to language equivalence in automata theory, e.g., the one by Hoare [140], since they are PSPACE-complete as shown by Kanellakis and Smolka [152]—following the corresponding PSPACE-completeness result for language equivalence by Meyer and Stockmeyer [193]. As pointed out by Van Glabbeek [113, Section 5], it is a good strategy to start the selection process for an equivalence with the finest one available and later step down to coarser ones since previous verification steps usually remain valid if a coarser equivalence is used later. We thus begin our selection process with the finest one available [112], viz. strong bisimilarity, and descend to four coarser ones since strong bisimilarity does not distinguish between observable and

unobservable actions, i.e., all actions are observable, and one of our goals is to neglect unobservable parts as much as possible.

We consider the following well-known equivalences in a selection process for a behavioral equivalence of interaction systems, ordered by their appearance in the literature. The formal definitions in the setting of this thesis can be found in Appendix E (cf. the pointers in the following list).

1. Strong bisimilarity: $\approx_s$ (cf. Definition E.1)
2. Weak bisimilarity: $\approx_w$ (cf. Definition E.2)
3. Branching bisimilarity: $\approx_b$ (cf. Definition E.3)
4. Divergence sensitive branching bisimilarity: $\approx_b^\lambda$ (cf. Definition E.4)
5. Branching bisimilarity with explicit divergence: $\approx_b^\Delta$ (cf. Definition E.5)

We want to point out that weak bisimilarity has been used heavily in models comparable to interaction systems (cf. Section 2.1.4), e.g., in the approaches by Hennicker et al. [137] and by Bernardo et al. [39]. We have used branching bisimilarity in the context of interaction systems [160]. It is thus natural to include these equivalences in our selection process here. The other two equivalences are finer versions of branching bisimilarity. Note that all five equivalences can be computed in polynomial time (cf. Appendix E).

We want to shortly mention the origins of these bisimilarities, we refer the reader to Appendix E and the work of Van Glabbeek [113] for a more detailed discussion. In particular, we focus on logical characterizations since these provide a measure for the kind of properties that are preserved. The origins of strong bisimilarity go back to Park [222] and Milner [197] where the notation of Park is the one that is typically used today (and which was also adopted by Milner [198]). Browne et al. [53] provide a logical characterization of strong bisimilarity where satisfaction of the same CTL* formulae coincides with strong bisimilarity. Hennessy-Milner logic [135] also logically characterizes strong bisimilarity [136]. Milner was also involved in the introduction of the first variant that deals with unobservable behavior, i.e., τ-transitions, leading to the notion of weak bisimilarity by Hennessy and Milner [135, 136]—back then, they used the name observational equivalence (in the presence of unobservable actions) although Milner [198] also used the name weak bisimilarity. Bergstra and Klop [37] use the name τ-bisimilarity for a similar notion, and later Milner [201] uses the name bisimilarity for the coarsest relation satisfying his observational equivalence—to avoid confusion, we here only use the names strong and weak bisimilarity. In his book on CCS [200, Chapter 10], Milner introduces a weak variant of Hennessy-Milner logic, which is called $\mathcal{PL}^{\approx}$ (Process logic restricted to weak possibility), that logically

characterizes weak bisimilarity.

Van Glabbeek and Weijland [114] observe that weak bisimilarity does not preserve the branching structure of behavior, which led to the introduction of branching bisimilarity. Logical characterizations of branching bisimilarity are provided by De Nicola and Vaandrager [85], where the authors prove that CTL*–X interpreted over all paths instead of maximal ones—as originally defined by Emerson and Halpern [95]—logically characterizes branching bisimilarity (based on stuttering equivalence of Kripke structures [53]). In order to overcome this "all path interpretation", De Nicola and Vaandrager [85] introduce divergence sensitive branching bisimilarity to establish an equivalence that is logically characterized by CTL*–X (as defined here, cf. Appendix D). Van Glabbeek et al. [117] note that divergence sensitive branching bisimilarity is not suitable in compositional settings since it fails to be a congruence for a process algebraic parallel operator. The reason for this behavior is that CTL*–X cannot distinguish between deadlocks and livelocks as we discussed on page 45, and indeed, deadlocks and livelocks are equivalent under divergence sensitive branching bisimilarity. In the spirit of Bergstra et al. [38], a notion called branching bisimilarity with explicit divergence is introduced (which was already mentioned in earlier work by Van Glabbeek and Weijland [115]) that overcomes this problem. However, Van Glabbeek et al. [117] also need to adjust CTL*–X such that it characterizes this new bisimilarity, or alternatively, apply the characterization only for deadlock-free systems.

In the next section, we compare the five equivalences by means of a small example, summarize the pros and cons, and finish the selection process.

2.4.1 Choosing an Appropriate Behavioral Equivalence

The following example illustrates the differences of the equivalences with respect to the global behaviors of interaction systems.

Example 2.14: We compare five labeled transition systems that correspond to the global behaviors of interaction systems. We introduce five interaction systems $Sys^{(k)} = ((\{\{i\}, \{\{a_i, b_i, c_i\}\}), \{\{a_i\}, \{b_i\}, \{c_i\}\}, \{\{c_i\}\}), \{[\![i]\!]\})$ for $1 \leq k \leq 5$ with different local behavior of component i in each case (which therefore results in a different global behavior since i is the only component). Observe that action c_i is only contained in a closed interaction, thus becomes unobservable in the global behavior, i.e., each τ corresponds to c_i in the local behavior of the component. Figure 2.11 on the facing page shows the global behaviors of the five systems. We do not specify the local behavior of component i in each case since it is the same as the global behavior if we exchange the interactions with the corresponding actions.

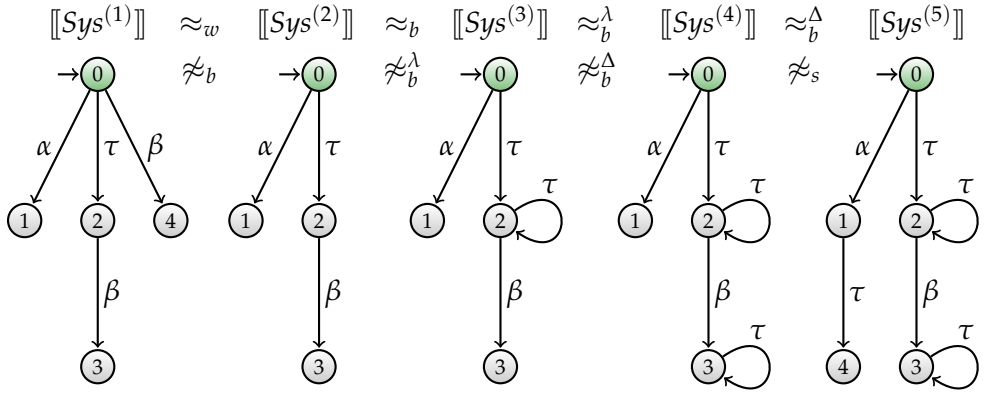


Figure 2.11: Five different behaviors of component i result in the five depicted global behaviors of the corresponding interaction systems, that can be distinguished by the introduced equivalences. The equivalences that identify and distinguish respectively each system are depicted between the initial states, where in each case this holds for all following systems from the left to the right, e.g., the first one is also weak bisimilar to the last one. We use the abbreviations $\alpha = \{a_i\}$ and $\beta = \{b_i\}$ for the open interactions.

These labeled transition systems are based on an example of Van Glabbeek and Weijland [115, Figure 7] where more examples that distinguish weak and branching bisimilarity can be found. Van Glabbeek et al. [117] have other examples that distinguish the different kinds branching bisimilarity.

We shortly demonstrate how the equivalences in Figure 2.11 are established. For instance, in order to show why the systems are weak bisimilar, we have to give an appropriate relation $\mathcal{R}_{\approx_w}$ on the states (cf. Definition E.2): We set $\mathcal{R}_{\approx_w} = \{(s^0, t^0), (s^1, t^1), (s^2, t^2), (s^3, t^3), (s^4, t^3)\}^{\leftrightarrow}$ where s denotes the states of $[\![Sys^{(1)}]\!]$ and t of $[\![Sys^{(2)}]\!]$, $[\![Sys^{(3)}]\!]$, or $[\![Sys^{(4)}]\!]$. Clearly, this shows the weak bisimilarity of the systems. If we want to establish the weak bisimilarity between $[\![Sys^{(1)}]\!]$ and $[\![Sys^{(5)}]\!]$, we have to add the tuples (s^1, t^4) and (t^4, s^1) to $\mathcal{R}_{\approx_w}$—again, if s denotes the states of $[\![Sys^{(1)}]\!]$ and t of $[\![Sys^{(5)}]\!]$. We do not explicitly address all combinations here, we just point out that, e.g., $[\![Sys^{(4)}]\!]$ and $[\![Sys^{(5)}]\!]$ can be distinguished by the CTL* formula $\Phi = \mathsf{AG}\big(\alpha \Rightarrow \mathsf{X}\,\mathsf{X}\,\top\big)$, i.e., $[\![Sys^{(4)}]\!] \not\models \Phi$ and $[\![Sys^{(5)}]\!] \models \Phi$, and thus they cannot be strongly bisimilar. The other cases can be settled in an analogous way by either giving an appropriate relation to establish the equivalence or a formula in the corresponding characterization to refute it.

Example 2.14 shows the differences in the five equivalence relations with respect to which behavior is considered as equivalent. Next, we provide an overview of these differences in Table 2.1 on the following page.

Equiv.	Logical Char.	Pros and Cons	Runtime
$\approx_s$	CTL*	compresses identical branches, treats unobservable behavior as observable	$O(m \cdot \log n)$
$\approx_w$	$\mathcal{PL}^{\approx}$	compresses branches with the same observable behavior, discards divergence, cannot distinguish dead- and livelock	$O(n^{2.3727})$
$\approx_b$	CTL*–X interpreted over all paths instead of max. ones	compresses unobservable sequences & identical branches, discards divergence, cannot distinguish dead- and livelock	$O(m \cdot n)$
$\approx_b^{\lambda}$	CTL*–X	compresses unobservable sequences & identical branches, preserves divergence in non-deadlocks, cannot distinguish dead- and livelock	$O(m \cdot n)$
$\approx_b^{\Delta}$	CTL*–X if free of deadlocks	compresses unobservable sequences & identical branches, preserves divergence	$O(m \cdot n)$

Table 2.1: Table summarizing the aspects for choosing an equivalence

We compare the logical characterizations, several pros and cons under the aspect of which behavior is considered equivalent, and the runtimes to check whether two systems are equivalent. For the runtimes, we use the abbreviations $n = |S_1| + |S_2|$ and $m = \sum_{a \in \Sigma}(|\xrightarrow{a}_1| + |\xrightarrow{a}_2|)$ for the sake of better readability—where $LTS_i = (S_i, \Sigma, \{\xrightarrow{a}_i\}_{a \in \Sigma}, S_i^0)$, $i = 1, 2$ are the two labeled transition systems under consideration (cf. Appendix E). Moreover, we assume that $m > n$ holds and consider the set of labels to be fixed, i.e., the number of labels is a constant, which yields the usual bounds found in the literature. For a more detailed discussion of the runtimes, we refer the reader to Appendix E.

This brings us back to our selection process: Which of the five equivalences is best suited as an equivalence for interaction systems based on the aspects in Table 2.1? We want to point out that at different occasions, different equivalences work well and that, if in doubt, starting off with a finer equivalence is a better choice since one can "fall back" to a coarser one if a certain verification step does not succeed—and all previous steps remain valid.

Clearly, we want to neglect some of the unobservable parts of a behavior, which means that strong bisimilarity is not suited. Although weak bisimilarity has a better runtime in Table 2.1 than the remaining bisimilarities, we know that the bound is dominated by the runtime of a sub-cubic fast matrix multiplication algorithm (cf. Appendix E) which is only reasonable for very large systems. As pointed out by Groote and Vaandrager [130, Section 6.2], this bound is only valid if the number of labels is a constant and otherwise, the runtime for branching bisimilarity is faster than for weak bisimilarity (again, we refer the interested reader to Appendix E for a more detailed discussion). Thus, we conclude that from a runtime point of view, there is no difference (since we aim for polynomial costs). However, there is a difference in the preservation of properties since CTL*–X is more expressive than $\mathcal{PL}^{\approx}$. Since we already introduced properties with respect to CTL*–X in Section 2.3, we narrow our selection process to one of the branching bisimilarities. Now, since the "all path interpretation" of CTL*–X is only natural in a setting where deadlocks are excluded beforehand [85, Example 3.2.8], divergence sensitive branching bisimilarity fails to be a congruence for a process algebraic parallel operator [117], and, as mentioned in Section 2.3, the distinction between deadlocks and livelocks is important, we choose branching bisimilarity with explicit divergence as our "working" equivalence for interaction systems. Again, we want to mention that a more detailed discussion can be found in Appendix E.

Next, we repeat the definition of branching bisimilarity with explicit divergence but extend it to interaction systems. This allows us to use the notion interchangeable for interaction systems and labeled transition systems, where the former case uses the global behavior and thus, the two coincide.

Definition 2.15 (Behavioral Equivalent Interaction Systems): Let $Sys^{(1)}$ and $Sys^{(2)}$ be two interaction systems with the same set of open interactions, i.e., $Int^{(1)}_{\text{open}} = Int^{(2)}_{\text{open}} := \Sigma$. Let $[\![Sys^{(i)}]\!] = (S_i, \Sigma^\tau, \{\xrightarrow{\alpha}_i\}_{\alpha\in\Sigma^\tau}, S^0_i)$, $i = 1, 2$, denote their global behaviors. The systems are *behavioral equivalent* or *branching bisimilar with explicit divergence*, denoted by $Sys^{(1)} \approx^{\Delta}_b Sys^{(2)}$, if $[\![Sys^{(1)}]\!] \approx^{\Delta}_b [\![Sys^{(2)}]\!]$ holds, i.e., if a symmetric relation $\mathcal{R} \subseteq S_1 \times S_2 \cup S_2 \times S_1$ exists such that for $i = 1, 2$ and $j = (i \bmod 2) + 1$ holds

1. for all states $s^0_i \in S^0_i$ exists a state $s^0_j \in S^0_j$ such that $(s^0_i, s^0_j) \in \mathcal{R}$ and
2. for all $(s_i, s_j) \in \mathcal{R}$ with $s_i \in S_i$ and $s_j \in S_j$ it holds that
   2.1. for all $t_i \in S_i$ and all $\alpha \in \Sigma^\tau$, if $s_i \xrightarrow{\alpha}_i t_i$ then
      (a) either $\alpha = \tau$ and $(t_i, s_j) \in \mathcal{R}$
      (b) or there are states $r_j, t_j \in S_j$ with $s_j \xrightarrow{\tau}^*_j r_j$, $r_j \xrightarrow{\alpha}_j t_j$, $(s_i, r_j) \in \mathcal{R}$, and $(t_i, t_j) \in \mathcal{R}$ and

2.2. if there is an infinite path π over $(S_i, \xrightarrow{\tau}_i)$ such that $\pi[0] = s_i$, then there exists a state $t_j \in S_j$ such that $s_j \xrightarrow{\tau}_j t_j$ and $(t_i, t_j) \in \mathcal{R}$ where $t_i = \pi[k]$ for some $k \in \mathbb{N}$.

We want to point out one important aspect that goes along with the choice of branching bisimilarity with explicit divergence. As mentioned in Table 2.1, two interaction systems that are behavioral equivalent with respect to Definition 2.15, i.e., their global behaviors are branching bisimilar with explicit divergence, satisfy the same set of properties only if they are deadlock-free. This emphasizes the importance of this property and we thus address this issue in an own chapter (cf. Chapter 6).

It is important to realize that this does not mean that branching bisimilarity with explicit divergence cannot distinguish between dead- and livelocks. Clearly, a deadlock cannot be related to a livelock since it has no infinite path of τ-transitions—as opposed to divergence sensitive branching bisimilarity where this is the case (cf. Definition E.4). Thus, the requirement of deadlock-freedom only refers to the logical characterization in CTL*–X, although freedom from deadlocks is in general a desired property of any system.

This ends the discussion and search for a behavioral equivalence for interaction systems that matches our goals. In the following sections, we address some further notions and useful ideas.

2.4.2 Reducing Behavior: Quotients

Since a relation that establishes branching bisimilarity with explicit divergence can be turned into an equivalence relation (see the work by Van Glabbeek et al. [116, Section 3] for a proof idea that adapts the proof of Basten [29] that branching bisimilarity (cf. Definition E.3) is an equivalence relation), a natural question is whether a labeled transition system can be reduced with the help of the equivalence classes that are induced by the relation. Similar approaches are well known in automata theory where deterministic finite automata can be minimized by computing the equivalence classes of the famous relation by Myhill [209] and Nerode [211] (see the book of Hopcroft et al. [145, Section 4.4] for an overview).

In the case of labeled transition systems, this reduced version (with respect to bisimilarity) is usually called the *quotient labeled transition system*—we simply call it quotient in the following—but works along the lines of automata minimization. In order to obtain the smallest quotient, one has to consider the coarsest equivalence relation, i.e., the one that relates as many states as possible, that still guarantees to establish bisimilarity. We refer the reader to

Section 7.1.1 in the book of Baier and Katoen [23] for an introduction to the concept of quotients and coarsest relations.

Now, we give a formal definition of the quotient with respect to branching bisimilarity with explicit divergence. Admittedly, such a definition is straightforward along the lines of the definition of the equivalence itself; however, we use this quotient in subsequent chapters and thus define it here for the sake of self-containedness and for introducing our notation.

Please note that we define the quotient with respect to labeled transition system and thus use Definition E.5 as the underlying definition of branching bisimilarity with explicit divergence and not Definition 2.15 that introduced this notion for interaction systems.

Definition 2.16 ($\approx_b^\Delta$-Quotient): Let $LTS = (S, \Sigma^\tau, \{\xrightarrow{a}\}_{a\in\Sigma^\tau}, S^0)$ be a labeled transition system. Let $\mathcal{R} \subseteq S \times S$ be a symmetric relation that establishes the branching bisimilarity with explicit divergence of LTS with itself, i.e., $LTS \approx_b^\Delta LTS$, that is the coarsest among all suitable relations, i.e., for all $\mathcal{R}' \subseteq S \times S$ it holds that if $\mathcal{R}'$ establishes $LTS \approx_b^\Delta LTS$, then $\mathcal{R}' \subseteq \mathcal{R}$ holds. The $\approx_b^\Delta$*-quotient* of LTS with respect to $\mathcal{R}$ is the labeled transition system $LTS_{\approx_b^\Delta} = (S', \Sigma^\tau, \{\xrightarrow{a}'\}_{a\in\Sigma^\tau}, S^{0\prime})$ with $S' = \{[s]_\mathcal{R} \mid s \in S\}$ and $S^{0\prime} = \{[s^0]_\mathcal{R} \mid s^0 \in S^0\}$ where $[s]_\mathcal{R} = \{t \in S \mid (s,t) \in \mathcal{R}\}$ denotes the equivalence class of a state $s \in S$ with respect to $\mathcal{R}$. The family of transition relations $\{\xrightarrow{a}'\}_{a\in\Sigma^\tau}$ is defined according to the cases in Definition E.5: For all $s, t \in S$ and all $a \in \Sigma^\tau$: If $a \neq \tau$, we have $[s]_\mathcal{R} \xrightarrow{a}' [t]_\mathcal{R}$ if and only if $s \xrightarrow{a} t$ holds, and if $a = \tau$, we have $[s]_\mathcal{R} \xrightarrow{\tau}' [t]_\mathcal{R}$ if and only if $[s]_\mathcal{R} \neq [t]_\mathcal{R}$ and $s \xrightarrow{\tau} t$ or $[s]_\mathcal{R} = [t]_\mathcal{R}$ and there is a $r \in [s]_\mathcal{R}$ with $r \xrightarrow{\tau}{}^+ r$.

The quotient of a labeled transition system introduced in Definition 2.16 is only useful if we can safely consider the quotient instead of the original system. In the following lemma, we formally establish this usefulness.

Lemma 2.17 (Validity of the $\approx_b^\Delta$-Quotient): Let $LTS = (S, \Sigma^\tau, \{\xrightarrow{a}\}_{a\in\Sigma^\tau}, S^0)$ be a labeled transition system. We have:

$$LTS_{\approx_b^\Delta} \approx_b^\Delta LTS.$$

A formal proof of Lemma 2.17 can be found in Appendix F on page 237.

Lemma 2.17 ensures the validity of our definition of the quotient with respect to branching bisimilarity with explicit divergence. Here, we do not address how the quotient can be computed, but give a detailed discussion in Appendix E. Instead, we introduce a further, very useful equivalence that we need in the following chapters.

2.4.3 A Further Equivalence: Isomorphism

We introduce a further equivalence that is stronger than the variants presented above, viz. isomorphism. In our context, isomorphism is understood similarly as in graph theory [89, Chapter 1]. Interestingly, graph isomorphism is one of the problems that can be solved in nondeterministic polynomial time, i.e., the corresponding decision problem belongs to NP, but it is unknown whether is is NP-complete [108, Section 7.1]. However, this does not guarantee that isomorphism can be established in polynomial time and we thus use this notion not for computations, but rather in situations where we want to establish that two systems are "nearly identical". Since in our setting we have labeled edges, we adjust the usual definition of isomorphism and derive a notion called isomorphism up to transition relabeling. Next, we define what we mean by this notion.

Definition 2.18 (Isomorphism up to Transition Relabeling): Let $LTS_i = (S_i, \Sigma_i, \{\xrightarrow{a}_i\}_{a\in\Sigma_i}, S_i^0)$, $i = 1, 2$, be labeled transition systems. The systems are *isomorphic up to transition relabeling* if two bijective functions $f\colon S_1 \to S_2$ and $g\colon \Sigma_1 \to \Sigma_2$ exist such that for all $s_1 \in S_1$ it holds that $s_1 \in S_1^0$ holds if and only if $f(s_1) \in S_2^0$ holds and for all $s_1, t_1 \in S_1$ and $a \in \Sigma_1$ it holds that $s_1 \xrightarrow{a}_1 t_1$ holds if and only if $f(s_1) \xrightarrow{b}_2 f(t_1)$ with $b = g(a)$ holds.

This ends our discussion of equivalences. Next, we summarize the chapter.

2.5 Summary

We introduced the model of interaction systems as a means for the specification of component-based systems. We discussed a variety of properties and showed how temporal logic can be used to specify properties precisely. The combinatorial explosion, that can arise when constructing the global behavior of an interaction system, opens the challenge of verification techniques that are guaranteed to run in polynomial time in the size of the original input, viz. the specification of the interaction system in question. We address this challenge in the subsequent chapters of this thesis where one way uses the behavioral equivalences that we introduced in this chapter.

In the next chapter, we study how the concepts of compositionality and abstraction can be employed in interaction systems.

Chapter 3

Compositionality & Abstraction

In the previous chapter, we considered *one* interaction system and its properties. An important extension to this consideration is the treatment of several interaction systems and their composition, e.g., the building blocks of a software product under development are not exclusively given as components but also as systems. This compositional aspect is very natural and often called the key concept of successful software engineering. It can thus be found in many formalisms such as the composition operators of process algebras. Moreover, the approach of component-based development is not only enriched by dividing functionality among the components but also by combining some of these components and using a resulting *composite component* in future systems. As we saw in the previous chapter (cf. Section 2.1.4), many related models allow for such composite components. Here, we treat such composite components as interaction systems.

The above described compositional way is known as *hierarchical modeling* and allows to organize the development of software products on multiple levels and fosters the reuse of system parts that have been verified as correct. The ultimate goal for software engineers following this way in combination with formal methods is to compose only verified system parts in order to automatically derive a correct system by construction. This approach, known as *correctness by construction*, enables software engineers to create complex and correct systems right from the beginning. One important aspect of this approach is the abstraction of information or functionality that is not needed on higher levels. For instance, consider two components that should be treated as a (composed) interaction system but only one of them should be used in further composition steps, i.e., the other component is only needed for internal cooperation in this interaction system. From a hierarchical modeling perspective, we now have to make sure that only the relevant information is

visible for composition, i.e., there has to be a way of hiding the components and interactions of a system that should not be visible. For this purpose, we already introduced closed interactions in Chapter 2 that provide this functionality in interaction systems, but we need an operator that allows us to declare interactions as closed.

Thus in the sequel, we define several operators for interaction systems that enable us to realize compositionality and abstraction in interaction systems. The key compositional concepts developed in this chapter become important in later chapters, where we make additional assumptions about the architecture of interaction systems. We start out with combining interaction systems in the next section.

3.1 Composition of Interaction Systems

We want to mention that there already is a composition operator for interaction systems, viz. the one defined in the work by Gössler and Sifakis [121]. However as we pointed out in Chapter 2, the concept of complete interactions that have to be closed with respect to set inclusion in the set of all possible interactions render the operator incompatible with our definition of interaction systems. Moreover, the operator is defined with a rule that prioritizes the execution of the largest interaction, and it does not allow to remove interactions that should not occur in the composite system. The advantage of these rules is that they automatically imply deadlock-freedom of a composite system if its constituting parts are deadlock-free but under strong restrictions such as the preserved executability of all interactions of the single parts. Our concept of closed interactions, that naturally should not be available for composition, cannot directly be added to this operator. Thus, we define a new operator that is compatible with our definition and allows for a more expressive composition at the price of not implying deadlock-freedom automatically.

But before we introduce our composition operator, we need three auxiliary definitions that ease the definition of our operator.

3.1.1 Preliminaries

We have to make one assumption before we start combining interaction systems, namely that their provided information does not overlap, i.e., we have to make sure that we can identify the origin of every component or action. For this purpose, we define the notion of *disjoint* interaction systems.

Definition 3.1 (Disjoint Interaction Systems): Let $Sys^{(1)}$ and $Sys^{(2)}$ be two interaction systems. $Sys^{(1)}$ and $Sys^{(2)}$ are called *disjoint*, if their component systems are disjoint, i.e., $Comp^{(1)} \cap Comp^{(2)} = \emptyset$ and $Act^{(1)} \cap Act^{(2)} = \emptyset$ holds, and all components' sets of states are disjoint, i.e., $S_i \cap S_j = \emptyset$ holds for all $i \in Comp^{(1)}$ and $j \in Comp^{(2)}$.

Please note that the disjointness of the sets of states within a single interaction system is already guaranteed by Definition 2.5 (cf. page 22).

The next definition introduces a notation that we need for our composition operator. This notation is not tied to interaction systems, we thus define it over arbitrary sets.

Definition 3.2 (Powerset Interjoin): Let $\{\mathcal{X}_i\}_{i\in\{1,\ldots,n\}}$ be a family of sets of sets with $n \geq 2$ and each $\mathcal{X}_i = \{X_{i_1}, \ldots, X_{i_m}\}$ with $i_m \geq 1$ for $1 \leq i \leq n$. The *powerset interjoin* of these sets is defined by $\mathcal{X}_1 \bowtie \mathcal{X}_2 \bowtie \ldots \bowtie \mathcal{X}_n := \{\xi_1 \cup \xi_2 \cup \ldots \cup \xi_n \mid \xi_1 \in \bigcup_{X\in\mathcal{X}_1} 2^X \ \wedge \ \xi_2 \in \bigcup_{X\in\mathcal{X}_2} 2^X \ \wedge \ \ldots \ \wedge \ \xi_n \in \bigcup_{X\in\mathcal{X}_n} 2^X\} \setminus (\bigcup_{1\leq i\leq n} \bigcup_{X\in\mathcal{X}_i} 2^X)$, i.e., the powerset interjoin contains only new (nonempty) sets that are not contained in any $\bigcup_{X\in\mathcal{X}_i} 2^X$ for $1 \leq i \leq n$.

We demonstrate the powerset interjoin by a small example. Consider the two sets of sets $\mathcal{X}_1 = \{\{a_1, a_2\}\}$ and $\mathcal{X}_2 = \{\{a_3\}, \{a_4\}\}$. We build their powerset interjoin which yields:

$$\begin{aligned}
\mathcal{X}_1 \bowtie \mathcal{X}_2 &= \{\xi_1 \cup \xi_2 \mid \xi_1 \in \bigcup_{X\in\mathcal{X}_1} 2^X \ \wedge \ \xi_2 \in \bigcup_{X\in\mathcal{X}_2} 2^X\} \setminus (\bigcup_{1\leq i\leq 2} \bigcup_{X\in\mathcal{X}_i} 2^X) \\
&= \big\{\xi_1 \cup \xi_2 \mid \xi_1 \in \{\emptyset, \{a_1\}, \{a_2\}, \{a_1, a_2\}\} \ \wedge \ \xi_2 \in \{\emptyset, \{a_3\}, \{a_4\}\}\big\} \setminus \\
&\qquad\qquad \big\{\emptyset, \{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}, \{a_1, a_2\}\big\} \\
&= \big\{\{a_1, a_3\}, \{a_1, a_4\}, \{a_2, a_3\}, \{a_2, a_4\}, \{a_1, a_2, a_3\}, \{a_1, a_2, a_4\}\big\}.
\end{aligned}$$

The last preliminary definition also introduces a notation that we need in the following. Again, the notation is not tied to interaction systems and thus defined with respect to arbitrary sets.

Definition 3.3 (Coverage): A set of sets $\mathcal{X}$ is said to *be covered by* a set of sets $\mathcal{Y}$, denoted by $\mathcal{X} \sqsubseteq \mathcal{Y}$, if $\forall X \in \mathcal{X} \ \exists Y \in \mathcal{Y}\colon X \subseteq Y$.

As an example, take a look at the sets $\mathcal{X} = \{\{1,2\}, \{2,3\}\}$, $\mathcal{Y} = \{\{1,2,3\}\}$, and $\mathcal{Z} = \{\{1,3\}\}$. Both $\mathcal{X}$ and $\mathcal{Z}$ are covered by $\mathcal{Y}$, i.e., $\mathcal{X} \sqsubseteq \mathcal{Y}$ and $\mathcal{Z} \sqsubseteq \mathcal{Y}$ holds, but the sets $\mathcal{X}$ and $\mathcal{Z}$ are not covered by each other, i.e., $\mathcal{X} \not\sqsubseteq \mathcal{Z}$ and $\mathcal{Z} \not\sqsubseteq \mathcal{X}$ holds.

Next, we introduce our composition operator for interaction systems.

3.1.2 The Composition Operator

We define the composition operator as follows.

Definition 3.4 (Composition): Let $Sys^{(1)}, \ldots, Sys^{(n)}$ be n pairwise disjoint interaction systems with $n \geq 2$, and let (I^+, I^-) be their *composition information* with $I^+ \subseteq Int^{(1)}_{\text{open}} \bowtie \ldots \bowtie Int^{(n)}_{\text{open}}$ being a set of *new interactions* that are added to the composite system and $I^- \subseteq Int^{(1)}_{\text{open}} \cup \ldots \cup Int^{(n)}_{\text{open}}$ being a set of *old interactions* that are removed from the composite system. Let $I^- \sqsubseteq I^+$ hold, i.e., any old interaction is contained in at least one new interaction. The *composition* of $Sys^{(1)}, \ldots, Sys^{(n)}$ with respect to (I^+, I^-) is the tuple

$$\bigotimes_{(I^+,I^-)} \{Sys^{(1)}, \ldots, Sys^{(n)}\} := \Big(\bigotimes_{(I^+,I^-)} \{IM^{(1)}, \ldots, IM^{(n)}\}, \{[\![i]\!]\}_{i \in Comp^{(1)} \cup \ldots \cup Comp^{(n)}} \Big)$$

with

- $\bigotimes_{(I^+,I^-)}\{IM^{(1)}, \ldots, IM^{(n)}\} := \Big(\bigotimes\{CS^{(1)}, \ldots, CS^{(n)}\}, \big(I^+ \cup Int^{(1)} \cup \ldots \cup Int^{(n)}\big) \setminus I^-, Int^{(1)}_{\text{closed}} \cup \ldots \cup Int^{(n)}_{\text{closed}} \Big)$ and
- $\bigotimes\{CS^{(1)}, \ldots, CS^{(n)}\} := \big(Comp^{(1)} \cup \ldots \cup Comp^{(n)}, \{A_i\}_{i \in Comp^{(1)} \cup \ldots \cup Comp^{(n)}}\big)$.

For the case $n = 2$, called *binary composition*, we also write $Sys^{(1)} \otimes_{(I^+,I^-)} Sys^{(2)}$. Finally, we write $\bigotimes_{\substack{(I^+,I^-) \\ 1 \leq i \leq n}} Sys^{(i)}$ if the numbering of the systems is unambiguous.

Please note that the sets I^+ and I^- in Definition 3.4 allow for modeling arbitrary composition situations as the following example shows.

Example 3.5: Consider two disjoint interaction systems $Sys^{(1)}$ and $Sys^{(2)}$, each with two components, the interaction sets $Int^{(1)} = \big\{\{a_1, a_2\}\big\}$ and $Int^{(2)} = \big\{\{a_3, a_4\}\big\}$, and empty sets of closed interactions. We want to compose these systems, i.e., we have to provide their composition information (I^+, I^-). If we now put $I^+ = \big\{\{a_1, a_2, a_3, a_4\}\big\}$, the set I^- allows us to control whether we still allow the interactions $\{a_1, a_2\}$ and $\{a_3, a_4\}$ to occur in the composite system, i.e., we put $I^- = \emptyset$, or we only want the interaction $\{a_1, a_2, a_3, a_4\}$ to be possible, i.e., we put $I^- = \big\{\{a_1, a_2\}, \{a_3, a_4\}\big\}$. Note that in either case $I^- \sqsubseteq I^+$ holds because $\{a_1, a_2\} \subseteq \{a_1, a_2, a_3, a_4\}$ and $\{a_3, a_4\} \subseteq \{a_1, a_2, a_3, a_4\}$.

In the definition of the composition operator, we only stated that the composite system is a tuple that has the same build-up as an interaction system, i.e., we can identify a component system and an interaction model. Next, we show that this tuple is indeed a valid interaction system with respect to its formal definition (cf. Definition 2.5).

Proposition 3.6 (Composition Operator Validity): The tuple constructed by the composition operator of Definition 3.4 is an interaction system.

A formal proof of Proposition 3.6 can be found in Appendix F on page 238.

We use our running example Sys_{MMS} to demonstrate the usage of the composition operator. We introduce a second customer and a second management component that build a second merchandise management system which we compose with our running example in order to access its storage component, i.e., the two systems share the storage component. This situation can occur when a wholesaler owns a retail shop and also runs an online shop where the prices differ in the two merchandise management systems and the wholesaler uses its storage for both shops.

We assume that the new merchandise management system is already modeled, i.e., given is the following interaction system:

$$Sys^{(2)}_{\text{MMS}} = (((Comp^{(2)}, \{A_{c_2}, A_{m_2}\}), Int^{(2)}, Int^{(2)}_{\text{closed}}), \{[\![c_2]\!], [\![m_2]\!]\})$$

where the elements of the component system and interaction model respectively are given by:

$$Comp^{(2)} = \{c_2, m_2\},$$

$$A_{c_2} = \{abort_{c_2}, ask_{c_2}, buy_{c_2}, refund_{c_2}\},$$

$$A_{m_2} = \{cancel_{m_2}, deliver_{m_2}, order_{m_2}, pay_{m_2}, print_{m_2}, reimburse_{m_2}, release_{m_2}, reserve_{m_2}\},$$

$$Int^{(2)} = \big\{\{abort_{c_2}, cancel_{m_2}\}, \{ask_{c_2}, order_{m_2}\}, \{buy_{c_2}, pay_{m_2}\}, \{deliver_{m_2}\}, \{print_{m_2}\}, \{refund_{c_2}, reimburse_{m_2}\}, \{release_{m_2}\}, \{reserve_{m_2}\}\big\}, \text{and}$$

$$Int^{(2)}_{\text{closed}} = \big\{\{print_{m_2}\}\big\}.$$

Figure 3.1 depicts the interaction model of $Sys^{(2)}$.

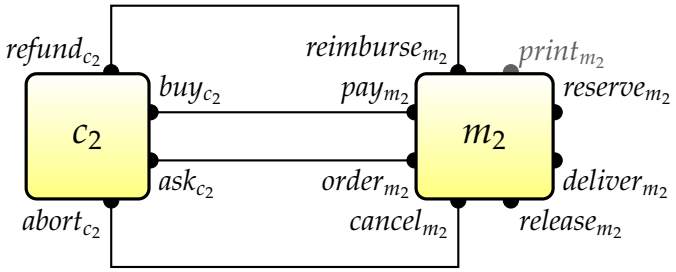


Figure 3.1: Interaction model of the second merchandise management system

For the behavior of the components in $Comp^{(2)}$, we assume that a labeled transition system equal to $[\![m]\!]$ up to label renaming is given for the second management component m_2, i.e., every label of $[\![m]\!]$ is replaced by the corresponding label in A_{m_2} to obtain $[\![m_2]\!]$ (cf. Figure 2.3 on page 23 for $[\![m]\!]$). The behavior of the second customer component c_2 is given as:

$$[\![c_2]\!] = (\{s^0_{c_2}\}, A_{c_2}, \{\{\}_{abort_{c_2}}, \{\}_{ask_{c_2}}, \{(s^0_{c_2}, s^0_{c_2})\}_{buy_{c_2}}, \{(s^0_{c_2}, s^0_{c_2})\}_{refund_{c_2}}\}, \{s^0_{c_2}\}).$$

Thus, this labeled transition system only consists of one initial state and two self-loops labeled by buy_{c_2} and $refund_{c_2}$ respectively, i.e., this customer can only directly buy items which is the case for a typical retail store.

Next, we compose the original merchandise management system Sys_{MMS} with the second system $Sys^{(2)}_{\text{MMS}}$. We combine the actions of the second management component m_2 that are used to access a storage component with the storage component s, i.e., we get the following set of new interactions:

$$I^+ = \{\{deliver_{m_2}, ship_s\}, \{release_{m_2}, unmark_s\}, \{reserve_{m_2}, mark_s\}\}.$$

Observe that $I^+ \subseteq Int_{\text{open}} \bowtie Int^{(2)}_{\text{open}}$ holds, i.e., the set is valid for the composition operator according to Definition 3.4. Since component m_2 should use the storage component s for any request, we have to remove the storage-cooperation singletons of $Int^{(2)}$ in the composite system, i.e., we get the following set of old interactions:

$$I^- = \{\{deliver_{m_2}\}, \{release_{m_2}\}, \{reserve_{m_2}\}\}.$$

Observe that $I^- \subseteq Int_{\text{open}} \cup Int^{(2)}_{\text{open}}$ holds, i.e., the set is valid for the composition operator according to Definition 3.4. Finally, observe that $I^- \sqsubseteq I^+$ holds, since any old interaction is contained in a new interaction.

We now set $Sys^{(3)}_{\text{MMS}} := \bigotimes_{(I^+, I^-)}\{Sys_{\text{MMS}}, Sys^{(2)}_{\text{MMS}}\}$, which results in:

$$Sys^{(3)}_{\text{MMS}} = (((Comp^{(3)}, \{A_i\}_{i \in Comp^{(3)}}), Int^{(3)}, Int^{(3)}_{\text{closed}}), \{[\![i]\!]\}_{i \in Comp^{(3)}})$$

where the new elements—the action sets and labeled transition systems were already specified above—are given by:

$$Comp^{(3)} = Comp \cup Comp^{(2)} = \{c, c_2, m, m_2, s\},$$

$$\begin{aligned} Int^{(3)} = \left(I^+ \cup Int \cup Int^{(2)}\right) \setminus I^- = \{&\{abort_c, cancel_m\}, \{abort_{c_2}, cancel_{m_2}\},\\ &\{ask_c, order_m\}, \{ask_{c_2}, order_{m_2}\}, \{buy_c, pay_m\}, \{buy_{c_2}, pay_{m_2}\},\\ &\{deliver_m, ship_s\}, \{deliver_{m_2}, ship_s\}, \{print_m\}, \{print_{m_2}\},\\ &\{refund_c, reimburse_m\}, \{refund_{c_2}, reimburse_{m_2}\}, \{release_m, unmark_s\},\\ &\{release_{m_2}, unmark_s\}, \{reserve_m, mark_s\}, \{reserve_{m_2}, mark_s\}\}, \text{ and} \end{aligned}$$

$$Int^{(3)}_{\text{closed}} = Int_{\text{closed}} \cup Int^{(2)}_{\text{closed}} = \{\{print_m\}, \{print_{m_2}\}\}.$$

We use our graphical representation to illustrate this composition process. Figure 3.2 depicts the composite interaction model, where we colored the new and old interactions (cf. the caption of the figure).

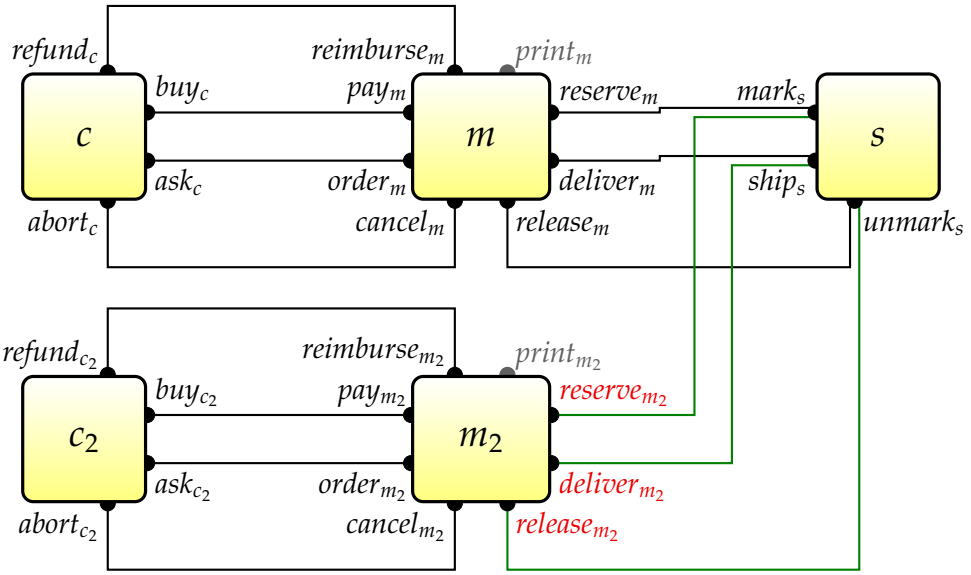


Figure 3.2: The interaction model of the composed system $Sys^{(3)}_{\text{MMS}}$. New interactions (I^+) are colored in green and old interactions (I^-) are colored in red where a completely red colored action illustrates that the corresponding singleton interaction is contained in I^-.

Now, it is interesting to verify properties of our composition, e.g., does the storage component answer any reservation request from the second management component before the first management component interacts with the storage, i.e., we want to verify $[\![Sys^{(3)}]\!] \stackrel{?}{\models} \Psi$ with:

$$\Psi = \mathsf{AG}\big(\{reserve_{m_2}, mark_s\} \Rightarrow \mathsf{A}(\neg \bigvee\nolimits_{\alpha \in Int^{(3)}_{\text{open}} \wedge \{m,s\} \subseteq \text{compset}(\alpha)} \alpha)\, \mathsf{U}\, (\{deliver_{m_2}, ship_s\} \vee \{release_{m_2}, unmark_s\})\big).$$

Indeed, this property holds but for its verification we have to construct the global behavior $[\![Sys^{(3)}_{\text{MMS}}]\!]$, i.e., execute BEHAVIOR-TRAVERSAL($Sys^{(3)}_{\text{MMS}}$) (cf. Algorithm B.2), and run a model checker, which is very efficient for this small system—the reachable part of $[\![Sys^{(3)}_{\text{MMS}}]\!]$ returned by Algorithm B.2 consists of 44 states and 117 transitions—but not for a large composition, e.g., imagine more management and customer components attached to the storage component. However, we do not depict the global behavior here, because the number of states and transitions is already too large.

Thus, we first establish properties of our composition operator that together with a means of abstraction allows us to verify properties on a reduced version

of a system and imply the property validity for the original system. However, we postpone this reduction approach to Chapter 5.

In the next section, we establish important properties of our composition operator such as its commutativity and associativity.

3.1.3 Properties of Composition

It is a useful property of composition operators that they are associative and commutative. Note that associativity is important for compositionality since it allows to compose several systems incrementally without paying attention to the order in which the systems are composed. However, we introduced our composition operator as a multi-ary variant, and associativity and commutativity are usually defined for binary operators. In order to overcome this "arity problem", we next show how we can expand a multi-ary composition to a sequence of binary composition steps.

Proposition 3.7 (Expansion to Binary Composition): Let $Sys^{(1)}, \ldots, Sys^{(n)}$ be n pairwise disjoint interaction systems with $n \geq 2$, and let (I^+, I^-) be their composition information with $I^+ \subseteq Int^{(1)}_{\text{open}} \bowtie \ldots \bowtie Int^{(n)}_{\text{open}}$, $I^- \subseteq Int^{(1)}_{\text{open}} \cup \ldots \cup Int^{(n)}_{\text{open}}$, and $I^- \sqsubseteq I^+$. The composition of these systems can be *expanded* to $n-1$ binary compositions as follows:

$$\bigotimes_{\substack{(I^+,I^-)\\1\leq i\leq n}} Sys^{(i)} \;=\; \underbrace{\big(\ldots\big(}_{n-2 \text{ times}} Sys^{(1)} \underset{\mathcal{I}_{1,2}}{\otimes} Sys^{(2)}\big) \underset{\mathcal{I}_{1,2,3}}{\otimes} Sys^{(3)}\big)\ldots\big) \underset{\mathcal{I}_{1,\ldots,n}}{\otimes} Sys^{(n)}$$

with $\mathcal{I}_{1,\ldots,i} := (I^+_{1,\ldots,i}, I^-_{1,\ldots,i})$ for $2 \leq i \leq n$ and

$$\begin{aligned}
I^+_{1,\ldots,i} := \big\{ & \alpha \in 2^{Act^{(1)} \cup \ldots \cup Act^{(i)}} \mid \exists\, \beta \in I^+ \colon \alpha = \beta \cap \big(Act^{(1)} \cup \ldots \cup Act^{(i)}\big) \\
& \quad \wedge\ \alpha \cap Act^{(i)} \neq \emptyset \ \wedge\ \alpha \not\subseteq Act^{(i)} \\
& \big\} \text{ and} \\
I^-_{1,\ldots,i} := \big\{ & \alpha \in 2^{Act^{(1)} \cup \ldots \cup Act^{(i)}} \mid \exists\, \beta \in I^+_{1,\ldots,i} \colon \\
& \quad \big(\alpha = \beta \setminus Act^{(i)} \ \wedge\ \alpha \notin I^+ \\
& \qquad \wedge\ \big(\forall\, \gamma \in I^+ \colon \alpha \subseteq \gamma \implies \beta \subseteq \gamma\big) \\
& \qquad \wedge\ \big(\forall\, j \in \{1, \ldots, i-1\} \colon \alpha \subseteq Act^{(j)} \implies \alpha \in I^-\big) \\
& \quad \big) \\
& \quad \vee \big(\alpha = \beta \cap Act^{(i)} \ \wedge\ \alpha \in I^-\big) \\
& \big\}.
\end{aligned}$$

A formal proof of Proposition 3.7 can be found in Appendix F on page 239.

We demonstrate the use of Proposition 3.7 by means of an example. Here, we do not use our running example because the system gets too big to be reasonable depicted.

Example 3.8: We consider the interaction systems $Sys^{(1)}$, $Sys^{(2)}$, and $Sys^{(3)}$ where for $k = 1, 2, 3$ the system $Sys^{(k)}$ consists of a component k with a single action a_k, i.e., $A_k = \{a_k\}$. Observe that the systems are pairwise disjoint. The behavior of each component is a labeled transition system $[\![k]\!]$ consisting of a single initial state and a self-loop labeled with the single action, and there is only one interaction consisting of the action as a singleton in each system, i.e., $Int^{(k)} = \big\{\{a_k\}\big\}$ for $k = 1, 2, 3$. No interaction is closed. Thus, we have $Sys^{(k)} = (((\{k\}, \{A_k\}), Int^{(k)}, \emptyset), \{[\![k]\!]\})$ for $k = 1, 2, 3$.

We want to compose these systems into one interaction system with a single interaction build from the individual singleton interactions, i.e., we set

$$Sys = \bigotimes_{(I^+, I^-)} \{Sys^{(1)}, Sys^{(2)}, Sys^{(3)}\}$$

with $I^+ = \big\{\{a_1, a_2, a_3\}\big\}$ and $I^- = \big\{\{a_1\}, \{a_2\}, \{a_3\}\big\}$. Observe that $I^+ \subseteq Int^{(1)}_{\text{open}} \bowtie Int^{(2)}_{\text{open}} \bowtie Int^{(3)}_{\text{open}}$, $I^- \subseteq Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}} \cup Int^{(3)}_{\text{open}}$, and $I^- \sqsubseteq I^+$ holds, i.e., the composition is valid (cf. Definition 3.4).

Since we want to demonstrate the expansion of the composition to a sequence of binary compositions as stated in Proposition 3.7, we apply this proposition and get

$$Sys = \bigotimes_{\substack{(I^+, I^-) \\ 1 \le i \le 3}} Sys^{(i)} = \big(Sys^{(1)} \underset{\mathcal{I}_{1,2}}{\otimes} Sys^{(2)}\big) \underset{\mathcal{I}_{1,2,3}}{\otimes} Sys^{(3)}$$

where the composition information is given by

$$\mathcal{I}_{1,2} = (I^+_{1,2}, I^-_{1,2}) \text{ with } I^+_{1,2} = \big\{\{a_1, a_2\}\big\} \text{ and } I^-_{1,2} = \big\{\{a_1\}, \{a_2\}\big\},$$
$$\mathcal{I}_{1,2,3} = (I^+_{1,2,3}, I^-_{1,2,3}) \text{ with } I^+_{1,2,3} = \big\{\{a_1, a_2, a_3\}\big\} \text{ and } I^-_{1,2,3} = \big\{\{a_1, a_2\}, \{a_3\}\big\}.$$

Observe that any new interaction occurring in a binary composition step corresponds to a set union of an interaction of the system of the left-hand side of the operator with an interaction of the system of the right-hand side with respect to a superset interaction contained in I^+. Some of these new interactions are temporary, i.e., they are needed for a subsequent composition in the sequence of steps. For instance, the interaction $\{a_1, a_2\}$ is needed in the composition step with $Sys^{(3)}$ in order to obtain the interaction $\{a_1, a_2, a_3\}$. But, these interactions need to be removed after they fulfilled their purpose which is achieved by an appropriate set of old interactions in each composition step, e.g., $\{a_1, a_2\}$ is contained in $I^-_{1,2,3}$.

As we already mentioned above, important features for a binary operator are its commutativity and associativity. The commutativity of our operator directly follows from the commutativity of the powerset interjoin and set union operators that are used to build the composition information. We formalize this by the following proposition.

Proposition 3.9 (Commutativity of the Binary Composition Operator): Let $Sys^{(1)}$ and $Sys^{(2)}$ be two disjoint interaction systems, and let (I^+, I^-) be their composition information with $I^+ \subseteq Int^{(1)}_{\text{open}} \bowtie Int^{(2)}_{\text{open}}$, $I^- \subseteq Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}}$, and $I^- \sqsubseteq I^+$. The composition is *commutative*, i.e., it holds that

$$Sys^{(1)} \underset{(I^+,I^-)}{\otimes} Sys^{(2)} = Sys^{(2)} \underset{(I^+,I^-)}{\otimes} Sys^{(1)}.$$

A formal proof of Proposition 3.9 can be found in Appendix F on page 242.

However, if we now take a look at the associativity, we cannot simply rely on the associativity of the underlying operators as for commutativity because our composition operator is parametrized. Thus, we have to adjust this parameter and reorganize the composition information. Fortunately, for this reorganization we can use the expansion of a composition into binary steps. We formalize this idea and the associativity in the following proposition.

Proposition 3.10 (Associativity of the Binary Composition Operator): Let $Sys^{(1)}$, $Sys^{(2)}$, and $Sys^{(3)}$ be three pairwise disjoint interaction systems, and let $(I^+_{1,2}, I^-_{1,2})$ be the composition information of $Sys^{(1)}$ and $Sys^{(2)}$ with $I^+_{1,2} \subseteq Int^{(1)}_{\text{open}} \bowtie Int^{(2)}_{\text{open}}$, $I^-_{1,2} \subseteq Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}}$, and $I^-_{1,2} \sqsubseteq I^+_{1,2}$. Furthermore, let $(I^+_{1,2,3}, I^-_{1,2,3})$ be the composition information of the system resulting from the first composition and $Sys^{(3)}$ with $I^+_{1,2,3} \subseteq Int^{(1,2)}_{\text{open}} \bowtie Int^{(3)}_{\text{open}}$, $I^-_{1,2,3} \subseteq Int^{(1,2)}_{\text{open}} \cup Int^{(3)}_{\text{open}}$, and $I^-_{1,2,3} \sqsubseteq I^+_{1,2,3}$ where $Int^{(1,2)}_{\text{open}} = (I^+_{1,2} \cup Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}}) \setminus I^-_{1,2}$ (cf. Definition 3.4).

The composition of these systems is *associative*, i.e., it holds that

$$\left(Sys^{(1)} \underset{(I^+_{1,2},I^-_{1,2})}{\otimes} Sys^{(2)}\right) \underset{(I^+_{1,2,3},I^-_{1,2,3})}{\otimes} Sys^{(3)} = Sys^{(1)} \underset{\mathcal{I}_{1',2',3'}}{\otimes} \left(Sys^{(2)} \underset{\mathcal{I}_{1',2'}}{\otimes} Sys^{(3)}\right)$$

where $1' = 2$, $2' = 3$, and $3' = 1$ and the composition information $\mathcal{I}_{1',2'}$ and $\mathcal{I}_{1',2',3'}$ are defined as in Proposition 3.7 with respect to the sets $I^+ = (I^+_{1,2} \cup I^+_{1,2,3}) \setminus I^-_{1,2,3}$ and $I^- = (I^-_{1,2} \cup I^-_{1,2,3}) \setminus I^+_{1,2}$ and with primed indices in the definition of the parametrized composition information.

A formal proof of Proposition 3.10 can be found in Appendix F on page 242.

The key point of this proposition is that the order of the interaction systems that is induced on the single composition by the parentheses of the expansion

in Proposition 3.7 is not important. In fact, which system is numbered as system one, i.e., $Sys^{(1)}$, can be changed with an index transformation in an automatic way as shown by Proposition 3.10. We demonstrate the associativity by the following example.

Example 3.11: We use the same interaction systems as introduced in Example 3.8 on page 67, i.e., we consider for $k = 1, 2, 3$ the interaction systems $Sys^{(k)}$, each with a component k, one action a_k, and one interaction $\{a_k\}$. We compose $Sys^{(1)}$ and $Sys^{(2)}$ with $(I^+_{1,2}, I^-_{1,2})$ where $I^+_{1,2} = \{\{a_1, a_2\}\}$ and $I^-_{1,2} = \{\{a_1\}, \{a_2\}\}$. Afterwards, we compose the resulting system and $Sys^{(3)}$ with $(I^+_{1,2,3}, I^-_{1,2,3})$ where $I^+_{1,2,3} = \{\{a_1, a_2, a_3\}\}$ and $I^-_{1,2,3} = \{\{a_1, a_2\}, \{a_3\}\}$. Thus, we have:

$$\left(Sys^{(1)} \underset{(\{\{a_1,a_2\}\},\{\{a_1\},\{a_2\}\})}{\otimes} Sys^{(2)}\right) \underset{(\{\{a_1,a_2,a_3\}\},\{\{a_1,a_2\},\{a_3\}\})}{\otimes} Sys^{(3)}.$$

Now, according to Proposition 3.10, the composition information can be rearranged as the operator is associative, i.e., we get

$$Sys^{(1)} \underset{(\{\{a_1,a_2,a_3\}\},\{\{a_1\},\{a_2,a_3\}\})}{\otimes} \left(Sys^{(2)} \underset{(\{\{a_2,a_3\}\},\{\{a_2\},\{a_3\}\})}{\otimes} Sys^{(3)}\right)$$

because for the composition information

$$\begin{aligned} I^+ &= (I^+_{1,2} \cup I^+_{1,2,3}) \setminus I^-_{1,2,3} = \left(\{\{a_1, a_2\}\} \cup \{\{a_1, a_2, a_3\}\}\right) \setminus \{\{a_1, a_2\}, \{a_3\}\} \\ &= \{\{a_1, a_2, a_3\}\} \text{ and} \\ I^- &= (I^-_{1,2} \cup I^-_{1,2,3}) \setminus I^+_{1,2} = \left(\{\{a_1\}, \{a_2\}\} \cup \{\{a_1, a_2\}, \{a_3\}\}\right) \setminus \{\{a_1, a_2\}\} \\ &= \{\{a_1\}, \{a_2\}, \{a_3\}\} \end{aligned}$$

the sets $\mathcal{I}_{1',2'}$ and $\mathcal{I}_{1',2',3'}$ with $1' = 2$, $2' = 3$, and $3' = 1$ are defined as $\mathcal{I}_{1',2'} = (I^+_{1',2'}, I^-_{1',2'})$ with $I^+_{1',2'} = \{\{a_2, a_3\}\}$ and $I^-_{1',2'} = \{\{a_2\}, \{a_3\}\}$ and $\mathcal{I}_{1',2',3'} = (I^+_{1',2',3'}, I^-_{1',2',3'})$ with $I^+_{1',2',3'} = \{\{a_1, a_2, a_3\}\}$ and $I^-_{1',2',3'} = \{\{a_2, a_3\}, \{a_1\}\}$.

We established the important properties of commutativity and associativity for the binary version of our composition operator. Furthermore, the expansion property allows us to roll out a composition step into a sequence of binary ones. In Chapter 2, we examined several equivalence relations for interaction systems. There, we chose branching bisimilarity with explicit divergence, denoted by $\approx^{\Delta}_b$, as a suitable equivalence for interaction systems (cf. Section 2.4.1). Now, an important question for any operator on interaction systems is whether the chosen equivalence relation is a congruence with respect to the operator. The property of being a congruence is essential in order to use the operator in algebraic reasoning. For instance, if we have a composition of several interaction systems and want to replace one of these system with an additional one whose global behavior is branching bisimilar

(with explicit divergence) to the one to be replaced, then we expect that the global behaviors of the overall composition before and after the replacement are branching bisimilar (with explicit divergence). However, such a property is only valid if the equivalence is a congruence with respect to the composition operator.

We want to mention that such a claim is not obvious and does not easily follow from the definition. For instance, Van Glabbeek et al. [117, Section 5] show that divergence sensitive branching bisimilarity (cf. Section 2.4) fails to be a congruence for a basic parallel composition operator for labeled transition systems. Here, we answer the congruence question for our composition operator affirmative in the following proposition.

Proposition 3.12 ($\approx_b^\Delta$ is a Congruence w.r.t. Composition): Let $Sys^{(1)}$, $Sys^{(2)}$, and $Sys^{(3)}$ be three interaction systems, where $Sys^{(3)}$ is disjoint from both $Sys^{(1)}$ and $Sys^{(2)}$. Let either $I^+ \subseteq Int^{(1)}_{\text{open}} \bowtie Int^{(3)}_{\text{open}}$ and $I^- \subseteq Int^{(1)}_{\text{open}} \cup Int^{(3)}_{\text{open}}$, or $I^+ \subseteq Int^{(2)}_{\text{open}} \bowtie Int^{(3)}_{\text{open}}$ and $I^- \subseteq Int^{(2)}_{\text{open}} \cup Int^{(3)}_{\text{open}}$ with $I^- \sqsubseteq I^+$ in both cases.

$$\text{If } Sys^{(1)} \approx_b^\Delta Sys^{(2)} \text{ then } Sys^{(1)} \underset{(I^+,I^-)}{\otimes} Sys^{(3)} \approx_b^\Delta Sys^{(2)} \underset{(I^+,I^-)}{\otimes} Sys^{(3)}.$$

A formal proof of Proposition 3.12 can be found in Appendix F on page 243.

We summarize the properties of the composition operator for interaction systems as given by Definition 3.4. We showed that the operator

- yields a valid interaction system (cf. Proposition 3.6),
- can be expanded to binary composition steps (cf. Proposition 3.7),
- is commutative in its binary variant (cf. Proposition 3.9),
- is associative in its binary variant (cf. Proposition 3.10), and
- ensures that branching bisimilarity with explicit divergence is a congruence (cf. Proposition 3.12).

In the following section, we introduce an operator that allows for abstraction in interaction systems.

3.2 Abstraction in Interaction Systems

As we mentioned in the introduction to this chapter, from a hierarchical modeling perspective, an important aspect is the abstraction of information or functionality that is not needed on higher levels. Here, we exploit the closed

interactions (cf. Definition 2.3) for this purpose as such interactions become unobservable in the global behavior and are not available for composition because we required that only open interactions can be used as new and old interactions (cf. Definition 3.4).

Thus, we use, similarly to a process algebra such as CSP [141] or CCS [200], the concealment or hiding of interactions as a means of abstraction.

3.2.1 The Closing Operator

We define the closing operator as follows.

Definition 3.13 (Closing of Interactions): Let Sys be an interaction system and $\hat{I}$ be an arbitrary set. The *closing* of the interactions over Sys's component system contained in $\hat{I}$ in Sys, denoted by $Sys \setminus\setminus \hat{I}$, is the tuple $Sys \setminus\setminus \hat{I} := \big(IM \setminus\setminus \hat{I}, \{[\![i]\!]\}_{i \in Comp}\big)$ with $IM \setminus\setminus \hat{I} := \big(CS, Int, Int_{\text{closed}} \cup (\hat{I} \cap Int)\big)$.

Observe that only those elements of $\hat{I}$ that are also interactions over Sys's component system are relevant for the closing operation. In Definition 3.13, we stated that the new operator yields a tuple. Next, we show that this tuple is an interaction system.

Proposition 3.14 (Closing Operator Validity): The tuple constructed by the closing operator of Definition 3.13 is an interaction system.

A formal proof of Proposition 3.14 can be found in Appendix F on page 245.

As an example for the closing operator, we close the interactions that are used between the customer and the management component of our running example Sys_{MMS}. We set:

$$\hat{I} = \big\{\{abort_c, cancel_m\}, \{ask_c, order_m\}, \{buy_c, pay_m\}, \{refund_c, reimburse_m\}\big\}$$

and $Sys_{\text{MMS}} \setminus\setminus \hat{I} =: Sys^{(4)}_{\text{MMS}}$. Figure 3.3 depicts the new interaction model.

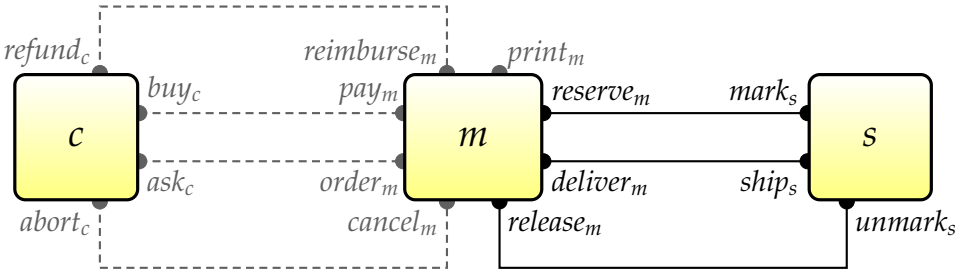


Figure 3.3: The interaction model of interaction system $Sys^{(4)}_{\text{MMS}}$

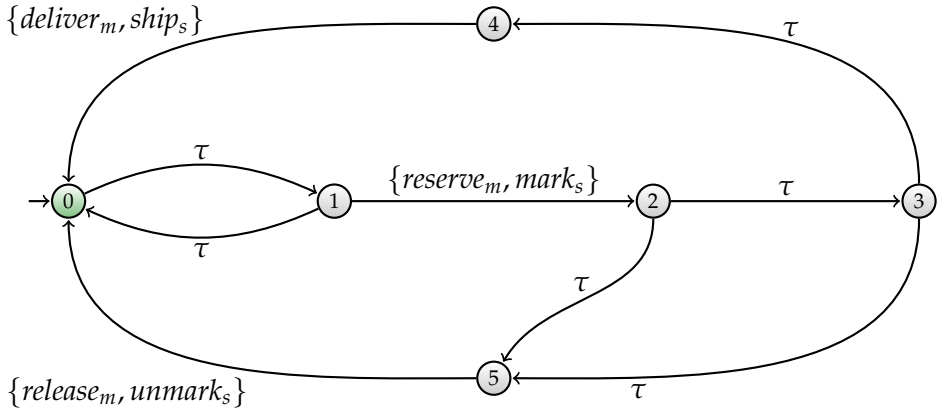


Figure 3.4: The reachable global behavior of interaction system $Sys^{(4)}_{\text{MMS}}$

From $Sys^{(4)}_{\text{MMS}}$'s global behavior, which is depicted in Figure 3.4, we see why the closing operator is a means for abstraction: An outside observer cannot distinguish the (now) internal cooperation of the customer and management component anymore. Observe that the composition with $Sys^{(2)}_{\text{MMS}}$ (cf. page 63) is still possible since the interactions needed for this composition are open.

Next, we establish important properties of our closing operator.

3.2.2 Properties of Closing

First, we consider the interplay of the composition and closing operator, i.e., whether they can be used in combination. This question typically arises when we have a composition of two interaction system and afterwards close some of the interactions. Now, we want to know which of these to be closed interactions can be handed over as a closing operation to the systems before their composition, and how these interactions can be identified.

Proposition 3.15 (Localization of Closed Interactions): Let $Sys^{(1)}$ and $Sys^{(2)}$ be two disjoint interaction systems, and let (I^+, I^-) be their composition information with $I^+ \subseteq Int^{(1)}_{\text{open}} \bowtie Int^{(2)}_{\text{open}}$, $I^- \subseteq Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}}$, and $I^- \sqsubseteq I^+$. Further let $\hat{I}$ be an arbitrary set whose elements, that are interactions over the composed component system, should become closed after the composition. For all subsets $\hat{I}^{(1)}, \hat{I}^{(2)}$ of $\hat{I}$ with $\hat{I}^{(i)} \cap \{\alpha \in 2^{Act^{(i)}} \mid \alpha \in I^- \ \vee \ (\exists\, \beta \in I^+ \colon \alpha = \beta \cap Act^{(i)} \ \wedge \ \alpha \neq \emptyset)\} = \emptyset$ for $i = 1, 2$ holds

$$\left(Sys^{(1)} \underset{(I^+,I^-)}{\otimes} Sys^{(2)}\right) \setminus\setminus \hat{I} = \left(Sys^{(1)} \setminus\setminus \hat{I}^{(1)} \underset{(I^+,I^-)}{\otimes} Sys^{(2)} \setminus\setminus \hat{I}^{(2)}\right) \setminus\setminus \hat{I}.$$

A formal proof of Proposition 3.15 can be found in Appendix F on page 245.

We demonstrate the use of Proposition 3.15 by means of an example.

Example 3.16: Consider two disjoint interaction systems $Sys^{(1)}$ and $Sys^{(2)}$, each with two components, the interaction sets $Int^{(1)} = \{\{a_1\}, \{a_1, a_2\}\}$ and $Int^{(2)} = \{\{a_3\}, \{a_3, a_4\}\}$, and empty sets of closed interactions. The composition information of these systems is given by $I^+ = \{\{a_1, a_2, a_3, a_4\}\}$ and $I^- = \emptyset$, i.e., (I^+, I^-) is a valid composition information as required by Definition 3.4. Suppose that we now want to close all singleton interactions, i.e., we set $\hat{I} = \{\{a_1\}, \{a_3\}\}$. According to Proposition 3.15, we now have for the two subsets $\hat{I}^{(1)} = \{\{a_1\}\}$ and $\hat{I}^{(2)} = \{\{a_3\}\}$ of $\hat{I}$ that

$$\hat{I}^{(i)} \cap \{\alpha \in 2^{Act^{(i)}} \mid \alpha \in I^- \ \vee \ (\exists\, \beta \in I^+ \colon \alpha = \beta \cap Act^{(i)} \ \wedge \ \alpha \neq \emptyset)\} = \emptyset$$

holds for $i = 1, 2$ since the latter set of the intersection equals $\{\{a_1, a_2\}\}$ for $Sys^{(1)}$ and $\{\{a_3, a_4\}\}$ for $Sys^{(2)}$. Thus, we can already close these interactions in the respective subsystems before the composition step, i.e., we have (as stated in Proposition 3.15):

$$\big(Sys^{(1)} \underset{(I^+,I^-)}{\otimes} Sys^{(2)}\big) \setminus\!\setminus \hat{I} = \big(Sys^{(1)} \setminus\!\setminus \hat{I}^{(1)} \underset{(I^+,I^-)}{\otimes} Sys^{(2)} \setminus\!\setminus \hat{I}^{(2)}\big) \setminus\!\setminus \hat{I}.$$

Next, we address the question whether branching bisimilarity with explicit divergence is a congruence with respect to the closing operator. As mentioned above for the similar question in case of the composition operator, such a property is essential if we want to use the operator in reasoning about systems, e.g., substitute behavioral equivalent system parts.

Proposition 3.17 ($\approx_b^\Delta$ is a Congruence w.r.t. Closing): Let $Sys^{(1)}$ and $Sys^{(2)}$ be two interaction systems, and let $\hat{I}$ be an arbitrary set whose elements, that are interactions over one of the component systems, should become closed.

$$\text{If } \ Sys^{(1)} \approx_b^\Delta Sys^{(2)} \ \text{ then } \ Sys^{(1)} \setminus\!\setminus \hat{I} \approx_b^\Delta Sys^{(2)} \setminus\!\setminus \hat{I}.$$

A formal proof of Proposition 3.17 can be found in Appendix F on page 245.

We summarize the properties of the closing operator for interaction systems as given by Definition 3.13. We showed about the closing operator that it

- yields a valid interaction system (cf. Proposition 3.14),
- integrates well with the composition operator (cf. Proposition 3.15), and
- ensures that branching bisimilarity with explicit divergence is a congruence (cf. Proposition 3.17).

In the next section, we introduce an operator that allows to decompose an interaction systems.

3.3 Decomposition of Interaction Systems

The complement of a composition operator is a disassembling or decomposition operator. For interaction systems, such an operator has been defined by several authors in order to consider a partial system or subsystem of a given interaction system.

Historically, Gössler and Sifakis [121] define a so-called set of interacting components in their original work that introduced interaction systems, which restricts the interaction model to the components in the set by intersecting all interactions with the action sets of the chosen components (and some additional technical requirements). We find similar definitions for interaction systems under a variety of notions such as an induced system [124, Definition 13], a projection operator for the global behavior [187, Section 2.2], a projection operator for interaction systems [179, Definition 2], subsystems [184, Definition 6], subsystem construction [160, Definition 4], or partial behavior [162, Definition 3].

Here, we also introduce such an operator that allows for decomposing an interaction system with respect to a set of components, which yields a subsystem of the original system, hence the name *subsystem construction operator*. It is important that such a subsystem is also a valid interaction system (with respect to Definition 2.5). Furthermore, the operator should be usable in combination with our composition and closing operator, i.e., the decomposition of a system into two interaction systems should be re-composable to the original system. We start out with formally defining our operator.

3.3.1 The Subsystem Construction Operator

We define the subsystem construction operator as follows.

Definition 3.18 (Subsystem Construction): Let Sys be an interaction system and let $C \subseteq Comp$ be a nonempty set of components. The *subsystem* of Sys obtained by only considering the components in C, denoted by $Sys[C]$, is the tuple $Sys[C] := (IM[C], \{[\![i]\!]\}_{i \in C})$ with

- $IM[C] := (CS[C], Int[C], Int_{\text{closed}}[C])$,
- $CS[C] := (C, \{A_i\}_{i \in C})$ and $Act[C] := \bigcup_{i \in C} A_i$,
- $Int[C] := \{\alpha \in 2^{Act[C]} \mid \exists\, \beta \in Int\colon \alpha = \beta \cap Act[C] \ \wedge\ \alpha \neq \emptyset\}$, and
- $Int_{\text{closed}}[C] := \{\alpha \in Int_{\text{closed}} \mid \forall\, \beta \in Int\colon \alpha \subseteq \beta \implies \text{compset}(\beta) \subseteq C\}$.

The set of open interactions is defined as $Int_{\text{open}}[C] := Int[C] \setminus Int_{\text{closed}}[C]$.

In Definition 3.18, we stated that the new operator yields a tuple. Next, we show that this tuple is also an interaction system.

Proposition 3.19 (Subsystem Construction Operator Validity): The subsystem construction operator of Definition 3.18 yields a tuple that is an interaction system.

A formal proof of Proposition 3.19 can be found in Appendix F on page 246.

We demonstrate the subsystem construction operator by an application to our running example. Consider again $Sys^{(4)}_{\text{MMS}}$ (cf. page 71), which we obtained from the original merchandise management system Sys_{MMS} by closing all interactions between the management and the customer component. We now apply our new operator with respect to the set $C = \{m, s\}$, i.e., we want to construct the subsystem consisting of the management and storage component, which we call $Sys^{(5)}_{\text{MMS}}$. According to Definition 3.18, we get:

$$Sys^{(5)}_{\text{MMS}} := Sys^{(4)}_{\text{MMS}}[\{m,s\}] = \left(IM^{(4)}[\{m,s\}], \{[\![i]\!]\}_{i \in \{m,s\}}\right)$$
$$\text{with } IM^{(4)}[\{m,s\}] = \left(CS^{(4)}[\{m,s\}], Int^{(4)}[\{m,s\}], Int^{(4)}_{\text{closed}}[\{m,s\}]\right).$$

Here, the interaction model of the subsystem consists of the following items and is depicted in the red box in Figure 3.5:

$$CS^{(4)}[\{m,s\}] = \left(\{m,s\}, \{A_m, A_s\}\right),$$
$$Int^{(4)}[\{m,s\}] = \{\{cancel_m\}, \{order_m\}, \{pay_m\}, \{print_m\}, \{reimburse_m\},$$
$$\{deliver_m, ship_s\}, \{release_m, unmark_s\}, \{reserve_m, mark_s\}\},$$
$$Int^{(4)}_{\text{closed}}[\{m,s\}] = \{\{print_m\}\}.$$

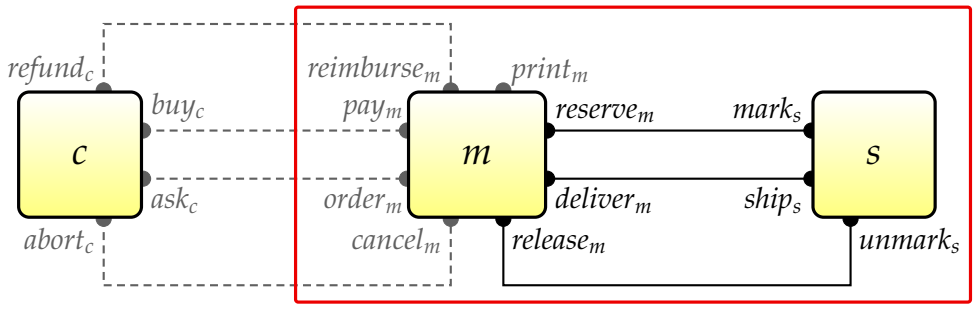


Figure 3.5: The red box surrounds the components of the subsystem that corresponds to the interaction system $Sys^{(5)}_{\text{MMS}}$. Observe that an interaction crossing the border of the box needs to be modified for the subsystem. Particularly, if such an interaction is closed (depicted as a dashed line in the figure), the modified version needs to be open in the subsystem.

The reachable global behavior of $Sys^{(5)}_{\text{MMS}}$ is depicted in Figure 3.6. Interestingly, the behavior of this interaction system, which is yielded by the subsystem construction operator, seems to "contain" the behavior of the original system up to relabeling (cf. Figure 2.5 on page 31), i.e., the behavior that is observable in the original system is also observable in the subsystem.

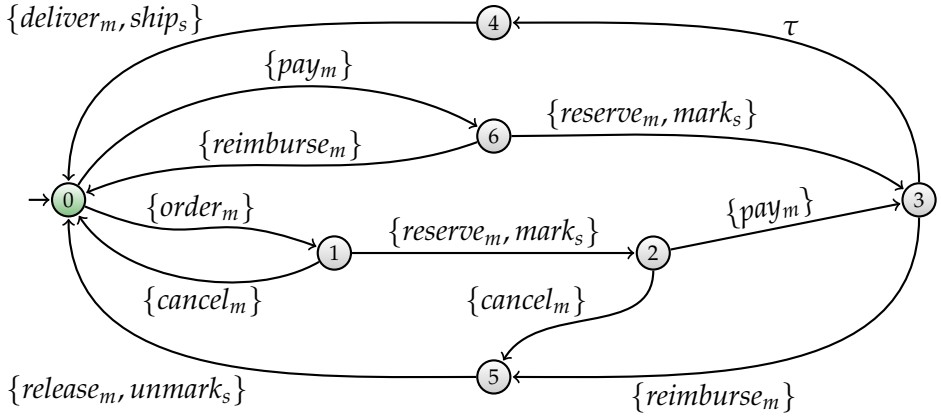


Figure 3.6: The reachable global behavior of interaction system $Sys^{(5)}_{\text{MMS}}$

This "containedness" is sometimes referred to as being an over-approximation, i.e., the behavior of the original system is approximated but additional behavioral branches are possible hence the name over-approximation. This observation paved the way for several verification approaches that aim for efficiency, i.e., avoid the construction of the global behavior of the whole system. As we already mentioned, we address such approaches in later chapters and thus do not discuss the observation further at this point.

As for the previous operators, we continue with useful properties of subsystem construction in the following section.

3.3.2 Properties of Subsystem Construction

First, we show that an interaction system is equal to the interaction system yielded by the subsystem construction operator with respect to the set of components contained in its component system. This proposition allows to introduce subsystems when reasoning about a given interaction system.

Proposition 3.20 (Identity of Subsystems): Let Sys be an interaction system. It holds that

$$Sys = Sys[Comp].$$

A formal proof of Proposition 3.20 can be found in Appendix F on page 247.

We now want to relate the subsystem construction operator to the composition operator. Interestingly, we only used the fact that all components participating in a closed interaction are contained in the components of the subsystem for the proof of Proposition 3.19. But, in the definition of the subsystem construction operator (cf. Definition 3.18) we required that no interaction is closed in a subsystem that is a subset of an interaction where components not contained in the subsystem, i.e., they are contained in the original interaction system, participate in. This requirement becomes important if we want to compose subsystems of an interaction system and ensure that the composed system is equal to an appropriate subsystem consisting of the union of the set of components in the subsystems to be composed. The following proposition establishes this idea.

Proposition 3.21 (Expansion of Subsystems): Let Sys be an interaction system and let $C_1, C_2 \subseteq Comp$ be nonempty sets of components with $C_1 \cap C_2 = \emptyset$. Further let $\mathcal{I}_{C_1,C_2} := (I^+_{C_1,C_2}, I^-_{C_1,C_2})$ be the composition information between the subsystems with $I^+_{C_1,C_2} := \{\alpha \in Int[C_1 \cup C_2] \mid \exists\, i \in C_1 \;\; \exists\, j \in C_2 \colon \{i,j\} \subseteq \text{compset}(\alpha)\}$ and $I^-_{C_1,C_2} := \{\alpha \in Int[C_1] \cup Int[C_2] \mid \alpha \notin Int[C_1 \cup C_2]\}$. It holds that

$$Sys[C_1 \cup C_2] = \big(Sys[C_1] \underset{\mathcal{I}_{C_1,C_2}}{\otimes} Sys[C_2]\big) \setminus\!\setminus Int_{\text{closed}}[C_1 \cup C_2].$$

A formal proof of Proposition 3.21 can be found in Appendix F on page 247.

We demonstrate the use of Proposition 3.21 by means of an example.

Example 3.22: Consider an interaction system Sys consisting of four components each with a single action, i.e., $Comp = \{1,2,3,4\}$ and $A_i = \{a_i\}$ for all $i \in Comp$, and the interaction set $Int = \big\{\{a_1,a_3\},\{a_2,a_4\}\big\}$ and the closed interaction set $Int_{\text{closed}} = \big\{\{a_2,a_4\}\big\}$. Suppose we set $C_1 = \{1,2\}$ and $C_2 = \{3,4\}$, i.e., we consider the case $Sys = Sys[C_1 \cup C_2]$. According to Proposition 3.21, we now have $I^+_{C_1,C_2} = \big\{\{a_1,a_3\},\{a_2,a_4\}\big\}$ and $I^-_{C_1,C_2} = \big\{\{a_1\},\{a_2\},\{a_3\},\{a_4\}\big\}$. Observe that this composition information allows us to reconstruct the original system but the set of closed interactions needs to be adjusted since the sets of closed interactions of the subsystems $Sys[C_1]$ and $Sys[C_2]$ are empty (cf. Definition 3.18). Here, we have $Int_{\text{closed}}[C_1 \cup C_2] = \big\{\{a_2,a_4\}\big\}$ which illustrates the correctness of the equation $Sys[C_1 \cup C_2] = \big(Sys[C_1] \otimes_{\mathcal{I}_{C_1,C_2}} Sys[C_2]\big) \setminus\!\setminus Int_{\text{closed}}[C_1 \cup C_2]$ as in Proposition 3.21.

We further refine Proposition 3.21 such that the composition information between the two subsystems after expansion is as small as possible, i.e., that only components that cooperate with a counterpart in the other subsystem are considered in the composition information.

Proposition 3.23 (Identification of Cooperating Components): Let Sys be an interaction system and let $C_1, C_2 \subseteq Comp$ be nonempty sets of components with $C_1 \cap C_2 = \emptyset$. According to Proposition 3.21, the system can be expanded as follows:

$$Sys[C_1 \cup C_2] = \big(Sys[C_1] \underset{\mathcal{I}_{C_1,C_2}}{\otimes} Sys[C_2]\big) \setminus\!\setminus Int_{\text{closed}}[C_1 \cup C_2].$$

For any subsets $C_1' \subseteq C_1$ and $C_2' \subseteq C_2$ of components that are only cooperating with their corresponding superset—i.e., for the sets holds $\forall\, i, j \in Comp \;\; \forall\, \alpha \in Int\colon \big(i \in C_1' \;\wedge\; j \in C_2 \implies \{i,j\} \not\subseteq \text{compset}(\alpha)\big) \;\wedge\; \big(i \in C_2' \;\wedge\; j \in C_1 \implies \{i,j\} \not\subseteq \text{compset}(\alpha)\big)$—holds

$$Sys[C_1 \cup C_2] = \big(Sys[C_1] \underset{\mathcal{I}_{C_1 \setminus C_1', C_2 \setminus C_2'}}{\otimes} Sys[C_2]\big) \setminus\!\setminus Int_{\text{closed}}[C_1 \cup C_2].$$

where $\mathcal{I}_{C_1 \setminus C_1', C_2 \setminus C_2'}$ is defined as in Proposition 3.21.

A formal proof of Proposition 3.23 can be found in Appendix F on page 248.

For the previous two operators, we established that branching bisimilarity with explicit divergence is a congruence with respect to them. As we already mentioned, this is important in order to allow for the usage of these operators in algebraic reasoning. However, for the subsystem construction operator we cannot state a similar result, i.e., in general, branching bisimilarity with explicit divergence is not a congruence with respect to the operator. The following counterexample illustrates this issue.

Example 3.24: Consider the interaction systems $Sys^{(1)}$ and $Sys^{(2)}$, both consisting of the following component system: $CS = (Comp, \{A_i\}_{i \in Comp})$ with $Comp = \{1,2\}$, $A_1 = \{a_1\}$, and $A_2 = \{a_2\}$. The behavioral models of the systems are also equal and consist for each component of a single initial state and a self-loop labeled with the single action in each case, i.e., we have:

$$\{[\![i]\!]\}_{i \in Comp} \text{ where } [\![i]\!] = (\{s_i^0\}, A_i, \{\{(s_i^0, s_i^0)\}_{a_i}\}, \{s_i^0\}) \text{ for } i = 1,2.$$

The interaction models of the two systems are different. Here, we have:

$$IM^{(1)} = (CS, Int^{(1)}, Int^{(1)}_{\text{closed}}) \text{ with } Int^{(1)} = \big\{\{a_1, a_2\}\big\} \text{ and } Int^{(1)}_{\text{closed}} = Int^{(1)},$$
$$IM^{(2)} = (CS, Int^{(2)}, Int^{(2)}_{\text{closed}}) \text{ with } Int^{(2)} = \big\{\{a_1\}, \{a_2\}\big\} \text{ and } Int^{(2)}_{\text{closed}} = Int^{(2)}.$$

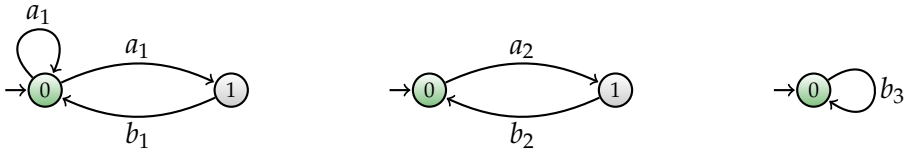
Thus, we have $Sys^{(1)} = (IM^{(1)}, \{[\![i]\!]\}_{i \in Comp})$ and $Sys^{(2)} = (IM^{(2)}, \{[\![i]\!]\}_{i \in Comp})$.

Now, consider the global behaviors of the systems. Clearly, we have $Sys^{(1)} \approx_b^{\Delta} Sys^{(2)}$ since all interactions are closed and thus only a τ-self-loop is observable. However, if we now construct the subsystems with respect to component 1, i.e., $Sys^{(1)}[\{1\}]$ and $Sys^{(2)}[\{1\}]$, we get $Sys^{(1)}[\{1\}] \not\approx_b^{\Delta} Sys^{(2)}[\{1\}]$ because in the former system we can observe an $\{a_1\}$-self-loop and in the latter we still

observe only a τ-self-loop. The reason for this behavioral difference clearly lies in the different interaction models, where in $Sys^{(1)}$ the closed interaction where both components participate in cannot be closed in $Sys^{(1)}[\{1\}]$ since the corresponding subsystem interaction $\{a_1\}$ is needed as an open interaction if we want to compose the subsystems $Sys^{(1)}[\{1\}]$ and $Sys^{(1)}[\{2\}]$ to get $Sys^{(1)}[\{1\} \cup \{2\}] = Sys^{(1)}$ (cf. Proposition 3.20).

Example 3.24 shows that in general, we cannot construct subsystems of arbitrary but branching bisimilar (with explicit divergence) interaction systems and hope for equivalent subsystems. However, an interesting question is whether additional requirements on the decomposition process exist such that the equivalence becomes a congruence with respect to the decomposition operator. From Example 3.24 we learned that we have to require at least equal alphabets of the global behaviors of the corresponding subsystems, i.e., $Int^{(1)}_{\text{open}}[\{1\}] = Int^{(2)}_{\text{open}}[\{1\}]$ in Example 3.24. Is this additional requirement sufficient to show that branching bisimilar with explicit divergence is a congruence with respect to the subsystem construction operator? Unfortunately, the answer is negative as the following example shows.

Example 3.25: Consider the interaction systems $Sys^{(1)}$ and $Sys^{(2)}$, both consisting of the following component system: $CS = (Comp, \{A_i\}_{i \in Comp})$ with $Comp = \{1, 2, 3\}$, $A_1 = \{a_1, b_1\}$, $A_2 = \{a_2, b_2\}$, and $A_3 = \{b_3\}$. The behavioral models of the systems are equal and the labeled transition systems of the components are depicted in Figure 3.7: $[\![1]\!]$ is given in Figure 3.7 (a), $[\![2]\!]$ is given in Figure 3.7 (b), and $[\![3]\!]$ is given in Figure 3.7 (c).



(a) Behavior of component 1 **(b)** Behavior of component 2 **(c)** Beh. of comp. 3

Figure 3.7: Behavior of the components 1, 2, and 3

The interaction models of the two systems are different:

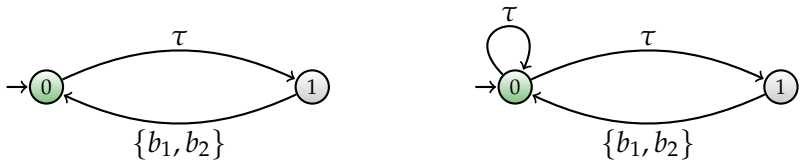
$$IM^{(1)} = (CS, Int^{(1)}, Int^{(1)}_{\text{closed}}) \text{ with } Int^{(1)} = \big\{\{a_1, a_2\}, \{b_1, b_2, b_3\}\big\} \text{ and}$$
$$IM^{(2)} = (CS, Int^{(2)}, Int^{(2)}_{\text{closed}}) \text{ with } Int^{(2)} = \big\{\{a_1\}, \{a_2\}, \{b_1, b_2, b_3\}\big\}.$$

In both cases, all interactions are closed, i.e., we have $Int^{(1)}_{\text{closed}} = Int^{(1)}$ and $Int^{(2)}_{\text{closed}} = Int^{(2)}$. Thus, we have $Sys^{(1)} = (IM^{(1)}, \{[\![i]\!]\}_{i \in Comp})$ and $Sys^{(2)} = (IM^{(2)}, \{[\![i]\!]\}_{i \in Comp})$.

Now, consider the global behaviors of the systems. Clearly, we have $Sys^{(1)} \approx_b^\Delta Sys^{(2)}$ since all interactions are closed and thus only a τ-self-loop is observable if we compare the quotients. But, if we now construct subsystems with respect to the set $C = \{1, 2\}$, we get the following sets of interactions (cf. Definition 3.18):

$$Int^{(1)}[C] = \{\{a_1, a_2\}, \{b_1, b_2\}\} \text{ and } Int^{(1)}_{\text{closed}}[C] = \{\{a_1, a_2\}\} \text{ and}$$
$$Int^{(2)}[C] = \{\{a_1\}, \{a_2\}, \{b_1, b_2\}\} \text{ and } Int^{(2)}_{\text{closed}}[C] = \{\{a_1\}, \{a_2\}\}.$$

Observe that in both subsystems, the interaction $\{b_1, b_2\}$ is not closed because its openness is needed for the re-composition with respect to component 3. Now, although the labeled transition systems that correspond to the global behaviors $[\![Sys^{(1)}[C]]\!]$ and $[\![Sys^{(2)}[C]]\!]$ are over the same alphabet, i.e., $Int^{(1)}_{\text{open}}[C] = Int^{(2)}_{\text{open}}[C]$ holds, the two subsystems are not branching bisimilar with explicit divergence. Figure 3.8 depicts the corresponding labeled transition systems.



(a) Global behavior of $Sys^{(1)}[C]$ **(b)** Global behavior of $Sys^{(2)}[C]$

Figure 3.8: Global behaviors of $Sys^{(1)}[C]$ and $Sys^{(2)}[C]$

Clearly, there is an infinite path of τ-transitions—formally, an infinite path over $(S^{(2)}, \xrightarrow{\tau}{}^{(2)})$—in $[\![Sys^{(2)}[C]]\!]$ (cf. Figure 3.8 (b)) which does not exist in $[\![Sys^{(1)}[C]]\!]$ (cf. Figure 3.8 (a)), i.e., no relation over the state spaces can satisfy Part 2.2 of Definition 2.15. Thus, we have $Sys^{(1)}[C] \not\approx_b^\Delta Sys^{(2)}[C]$.

Thus, it is not sufficient to require that $Int^{(1)}_{\text{open}}[C] = Int^{(2)}_{\text{open}}[C]$ holds as an additional requirement in order to show that branching bisimilarity with explicit divergence is a congruence with respect to the subsystem construction operator. In order to establish this property, we have to strengthen the additional requirements even more. But, Example 3.24 shows that requiring $Int^{(1)}[C] = Int^{(2)}[C]$ is also not sufficient. Thus, we have to use the strongest requirement, i.e., that the two subsystems are in fact equal, which we do not formally state here.

We summarize the properties of the subsystem construction operator for interaction systems as given by Definition 3.18. We showed about subsystem construction that it

- yields a valid interaction system (cf. Proposition 3.19),
- yields an identical interaction system with respect to the set of all components (cf. Proposition 3.20),
- integrates well with the composition and closing operator (cf. Propositions 3.21 and 3.23), and
- fails to ensure that branching bisimilarity with explicit divergence is a congruence (cf. Examples 3.24 and 3.25).

This ends the introduction of our operators for interaction systems. In the next section, we take a first look at correctness by construction.

3.4 Correctness by Construction

We already stressed the importance of approaches that support correctness by construction. An interesting question for our composition operator with respect to this approach is: Under which assumptions is the composition of two interactions systems automatically correct if the two systems are correct? Here, we derive a first result regarding deadlock-freedom.

3.4.1 Deadlock-Freedom Preserving Composition

We show that the composition of two deadlock-free interaction systems yields a deadlock-free (composite) interaction system if we restrict the composition information in a certain way. However, a simple relaxation of this result already fails as we discuss in the following.

But first, we motivate the idea behind this result. If we have two interaction systems that are both deadlock-free, we know that in all reachable states of these systems at least one interaction is enabled. Thus, if we compose the systems and want to preserve the property of deadlock-freedom, we have to ensure that the original interactions are still enabled in the corresponding composite states. The following theorem formalizes this idea.

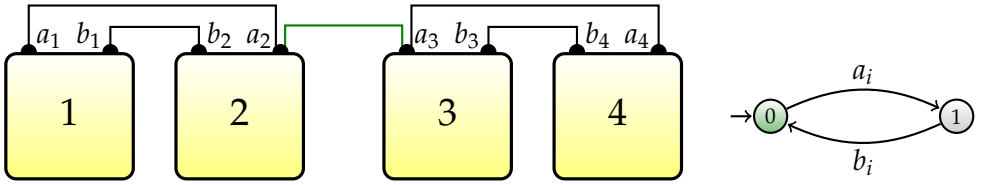
Theorem 3.26: Let $Sys^{(1)}$ and $Sys^{(2)}$ be two disjoint interaction systems, and let (I^+, I^-) be their composition information with $I^+ \subseteq Int^{(1)}_{\text{open}} \bowtie Int^{(2)}_{\text{open}}$ and $I^- = \emptyset$. If both $Sys^{(1)}$ and $Sys^{(2)}$ are deadlock-free and $\forall\, \alpha \in I^+ \;\exists\, \beta \in Int^{(1)} \;\exists\, \gamma \in Int^{(2)} \colon \alpha = \beta \cup \gamma$, then $Sys^{(1)} \otimes_{(I^+,I^-)} Sys^{(2)}$ is deadlock-free.

A formal proof of Theorem 3.26 can be found in Appendix F on page 249.

An important consequence of the above result is that if we can decompose a system in such a way that the recomposition only contains new interactions (with the additional assumption of Theorem 3.26), then we can also verify only the subsystems with respect to deadlock-freedom and the deadlock-freedom of the composed system follows. For instance, if there are interactions $\{a_i\}$, $\{b_j\}$, and $\{a_i, b_j\}$ of components i and j, then we can consider the subsystems $Sys[\{i\}]$ and $Sys[\{j\}]$ since for their composition, the set I^- is empty and all new interactions in the composition can be split into two interactions contained the subsystems.

However, a simple relaxation of the assumptions already fails, e.g., if we allow that interactions are removed or the new interactions are not restricted as in Theorem 3.26. We show the failure of the latter idea by a small example.

Example 3.27: We consider two interaction systems $Sys^{(1)}$ and $Sys^{(2)}$ that both consist of two components, i.e., we have $Comp^{(1)} = \{1, 2\}$ and $Comp^{(2)} = \{3, 4\}$. For all components i, where $1 \leq i \leq 4$, we have $A_i = \{a_i, b_i\}$. The interaction models of the systems are depicted in Figure 3.9 (a) (without the green-colored new interaction between components 2 and 3), i.e., we have $Int^{(1)} = \{\{a_1, a_2\}, \{b_1, b_2\}\}$ and $Int^{(2)} = \{\{a_3, a_4\}, \{b_3, b_4\}\}$. The local behavior of all components is depicted in Figure 3.9 (b): Each component first wants to execute its a-action and then its b-action.



(a) Interaction models and the composition information

(b) Beh. of i, $1 \leq i \leq 4$

Figure 3.9: Example illustrating a false relaxation of Theorem 3.26

Obviously, both systems are deadlock-free. Now, we compose the systems with the following composition information: We set $I^+ = \{\{a_2, a_3\}\}$ and $I^- = \emptyset$. The new interaction is also depicted in Figure 3.9 (a) as a green line. Observe that the tuple (I^+, I^-) is a valid composition information with respect to Definition 3.4.

However, if the composed interaction system now executes the new interaction $\{a_2, a_3\}$ in its global initial state $(s_1^0, s_2^0, s_3^0, s_4^0)$—observe that the new interaction is enabled—the next global state is $(s_1^0, s_2^1, s_3^1, s_4^0)$. But now, we reached a deadlock because each component wants to execute a different interaction in this global state.

Thus, the composition introduced a deadlock although the two systems were

originally deadlock-free. This explains the additional requirement about the new interactions that is included in Theorem 3.26. Note that this requirement does not hold for this example since the new interaction cannot be split into two interactions that were originally present in the systems.

Example 3.27 shows that more sophisticated methods are needed if the cooperation of the components is complex. We already mentioned that we consider such methods in following chapters of the thesis.

In the next section, we discuss how the operators can be used in algorithms and whether their application can be carried out in polynomial time.

3.5 Algorithmic Treatment of the Operators

Algorithms that construct new interaction systems with respect to an application of one of our three operators can be implemented straightforwardly: We only need to adjust the sets that define an interaction system (cf. Definition 2.5). Clearly, these operations can be carried out in polynomial time in the size of the input interaction system or the sum of the systems' sizes in case of the composition operator.

However, there is one task that we need to address before a composition, viz. the question whether the provided composition information is valid. Remember that for n given pairwise disjoint interaction systems $Sys^{(i)}$ where $1 \leq i \leq n$ a tuple (I^+, I^-) is provided and we need to check, as required by Definition 3.4, whether $I^+ \subseteq Int^{(1)}_{\text{open}} \bowtie \ldots \bowtie Int^{(n)}_{\text{open}}$, $I^- \subseteq Int^{(1)}_{\text{open}} \cup \ldots \cup Int^{(n)}_{\text{open}}$, and $I^- \sqsubseteq I^+$ holds. The latter two questions can be answered in polynomial time by looping trough the corresponding sets and checking for inclusion. The first one is a little bit more involved because if we compute the whole powerset interjoin with respect to Definition 3.2 and check for set inclusion, the check is not guaranteed to be polynomial in the size of the input, which here are the n interaction systems and the composition information. But, we can analyze the set I^+ in a more direct way: We partition each new interaction with respect to the global action sets of the n interaction systems and check for each partition whether it is a subset of an interaction of the corresponding interaction system. We give an algorithm for this procedure in Appendix B and here refer to Algorithm B.4, which shows that the validity of set I^+ can be checked in polynomial time.

Thus, we can use the three operators without the danger of a hidden exponential explosion such as a global behavior computation, which is, e.g., hidden in the property checks of Chapter 2. Next, we take a look at related work.

3.6 Related Work

In the previous sections, we already mentioned the related work regarding interaction systems, e.g., the composition operator of Gössler and Sifakis [121] that we discussed at the beginning of Section 3.1. A similar operator is defined by Gössler et al. [124, Definition 15] that we shortly discuss and compare with our operator (cf. Definition 3.4), although the authors also use the concept of complete interactions, which, as discussed at the beginning of Section 3.1, renders the operator incompatible with our definition of interaction systems. First, the binary operator of Gössler et al. [124] only allows to introduce new interactions that are the union of interactions already present in one of the two systems given as the parameter. Second, all interactions that are already present in one of the systems remain available. This allows to establish a correctness-by-construction approach that is similar to our Theorem 3.26 but makes the operator incompatible with any decomposition technique where a decomposition operator is introduced by the authors under the name induced system [124, Definition 13], i.e., if a system is decomposed into two distinct parts with respect to a partition of the set of components, then the composition operator of Gössler et al. [124] cannot be used to (re-)construct the original system. Thus, if we neglect the complete interactions as discussed at the end of Section 2.1.2, our composition is a generalization of the one proposed by the authors.

We take a look at other models of concurrency and component-based systems. In the work on process algebra we can find some similarities, although the operators are typically binary, i.e., only allow for the composition of two systems/processes. The composition operator of CCS [200] is only defined for binary synchronizations that become unobservable after the composition step, i.e., it cannot be extended to allow for multiway synchronizations. In CSP [141], this problem is overcome by allowing multiway synchronizations but over actions with the same name which yields an inflexible approach with respect to component-based development. In ACP [37], there is a communication function that maps the pairs of actions that can interact to a certain action, i.e., the function also allows multiway synchronizations. Although this function solves the naming problem of the CSP operator, the operator is still binary with respect to the number of processes. Similar observations have been made by Aldini and Bernardo [6]. As we mentioned in Section 2.1.4 of Chapter 2, the concept of multiactions and the combination of the composition and allow operator in the process algebra mCRL2 [131] is an exception to this list.

Some authors build on top of a process algebra a more component-oriented way of composition. For instance, Bernardo et al. [39] define a composition

operator in the process algebra associated to their PADL architecture description language. The operator is parametrized with a set of actions that happen synchronously in two systems, i.e., the same name is used to identify these parts. However, this operator is not used directly, i.e., in PADL the whole specification is translated into a process (cf. our discussion in Section 2.1.4) where the binary, left-associate composition operator of the process algebra is used to describe the overall system. However, many fresh labels need to be introduced since synchronization takes place over actions with the same name, similarly to CSP as mentioned above.

Similarly, in the formal semantic model of Java/A [137], a composition of two I/O transition systems takes place by a synchronization on identical input and output labels. Here, the composition operator is associative which allows for an extension to multi-ary composition, but again relabeling has to take place to synchronize on the ports. In this setting, similar properties as addressed for our operators are provided for I/O transition system [137, Section 2.2].

We already mentioned several related works regarding our subsystem construction operator at the beginning of Section 3.3. These further operators for interaction systems mainly differ with respect to the underlying interaction model. Similarly, our closing operator is comparable to hiding operators of process algebras such as, for instance, concealment in CSP [141]. Moreover, other component-based formalisms provide related ideas, e.g., in PADL [39], as already mentioned in Section 2.1.4 of Chapter 2, interactions can be declared as architectural interactions or local interactions where the latter are no longer available for further compositions.

In the next section, we summarize the chapter and give some additional remarks.

3.7 Summary and Remarks

In this chapter, we saw how interaction systems allow for hierarchical modeling and that correctness by construction can be achieved under strong requirements. We introduced three operators that are similar to operators typical found in process algebra and showed that these operators play well together and are reasonably defined, e.g., branching bisimilarity with explicit divergence, which is our chosen equivalence relation for interaction systems (cf. Definition 2.15), is a congruence with respect to the composition and the closing operator.

We want to point out that we did not address hierarchical modeling over several levels, i.e., we did not introduce an encapsulation technique that

allows to consider an interaction system as a component which looses the build-up of the underlying system. However, our operator also allows for such multiple levels if the composition information of each step is recorded and kept during following composition steps, i.e., the system is not considered as a whole interaction system but as several applications of our composition operator with hierarchical levels of parentheses, e.g., we always keep the left-hand side of Definition 3.4 instead of only the right-hand one.

Finally, we can also interpret the set of all interaction systems together with our composition operator (in its binary version) as a mathematical algebraic structure, viz. a semigroup, because our composition operator is associative (cf. Proposition 3.10)—if we neglect the automatic parameter adjustment. We can also turn this semigroup into a monoid by defining an appropriate identity element Sys_{id} such that for all valid interaction systems Sys holds: $Sys \otimes_{(I^+,I^-)} Sys_{\mathrm{id}} = Sys$ with $I^+ = \emptyset$ and $I^- = \emptyset$. This identity or neutral element corresponds to the empty interaction system, i.e., $Sys_{\mathrm{id}} = (((\emptyset,\emptyset),\emptyset,\emptyset),\emptyset)$, which is valid with respect to Definitions 2.1, 2.3, and 2.5 because we allow an empty set of components. However here, we do not formally define and prove the above claims; instead, we take a look at architectures for interaction systems in the next chapter.

Chapter 4

Architectures

In this chapter, we introduce restrictions on the way the components in an interaction system are allowed to cooperate, which we call *architectural constraints* or simply *architectures*. Historically, such architectures in interaction systems were introduced by Majster-Cederbaum and Martens [179] as a line of attack to overcome the complexity of deadlock detection that arises if we have to analyze the global behavior of an interaction system—remember the PSPACE-completeness of the decision problem whether a system is deadlock-free discussed in Section 2.3.1—and were successfully exploited to establish the property of deadlock-freedom in polynomial time in several works on interaction systems [160–162, 179, 180].

The idea behind such constraints for deadlock detection can be illustrated as follows: If we have three components $client_1$, $client_2$, and *server*, and the clients cooperate only with the server and never directly with each other, it could be sufficient to analyze the cooperation of the components *server* and $client_1$ and, independently of the first analysis, the cooperation of *server* and $client_2$. Since the two clients do not cooperate directly, we potentially can then conclude the deadlock-freedom of the whole system by the analysis of the cooperation of small system parts. In doing so, the overall architecture enables us to find the systems parts we have to analyze.

We want to point out that authors working with other formalisms also addressed architectural constraints [39, 49, 137], which we also discuss in this chapter. Often, architectures arise naturally when client-server situations as mentioned above or hierarchical structures are studied [2, 146, 221], e.g., the connection diagram of processes constructed with the subordination operator in CSP always forms a certain architecture [141, Section 4.5.2]. Related ideas can be found in the concept of architectural styles of architecture description languages [3, 6, 110, 242] that employ design patterns to maintain flexibility

and comprehensibility of a software system—early approaches can be traced back to work by Dijkstra [91], Brooks and Iverson [52], and Parnas [223], see the book of Bass et al. [28, Section 2.7] for an overview.

In the following sections, we want to take a closer look at how such architectural constraints can be defined and checked in an automatic way. We want to focus on their formal definition, the question of how efficiently a given interaction system can be inspected, and the computational complexity of the decision problems discussed in Chapter 2 restricted to the class of interaction systems satisfying a certain architectural constraint.

In general, such constraints or restrictions are a well-known line of attack in the world of computational complexity and sometimes even allow for tremendous reductions of complexity, e.g., if we restrict the clauses in the 3-satisfiability problem, which is NP-complete [74], to be of length two, we get the 2-satisfiability problem which is known to be solvable in linear time [99]. However, although such restrictions do not always guarantee a reduction of complexity, they still offer new lines of attacking a computationally hard problem. For instance in graph theory, a lot of problems are NP-complete, e.g., existence of a Hamiltonian path or finding a maximum independent set [108, pages 199–200 respectively 194–195], but the restriction to a certain class allows for the application of clever approximation algorithms although the problem remains NP-complete in this class, e.g., Baker [25] derives an approximation algorithm for finding a maximum independent set in planar graphs but the problem in this class is still NP-complete [108, pages 194–195].

As we learn in this chapter, this is also the case for the architectural constraints considered in this thesis. But, these constraints allow for new lines of attack, whose discussion is presented in Chapters 5 and 6.

Before we formally define what we mean by an architecture and architectural constraints, we introduce a further example interaction system that follows us through the remainder of the thesis as our second running example.

4.1 A Further Example Interaction System

We introduce an interaction system $Sys_{\mathrm{DB}}(n)$ representing a database and n users, i.e., the system models a database server and a fixed number of clients that are allowed to read and write to the database. In order to avoid inconsistencies if one of the clients wants to write, the database provides a locking mechanism that ensures that read requests are not answered once a client is granted writing access and starts to write. Additionally, all clients

are informed about the data change and a global commit among the clients ensures the consistency with any local data. After this commit, the database performs an internal backup step; however, an appropriate backup component is not part of the example. Here, we consider n clients with the property that client i with $1 \leq i \leq n$ is able to write up to w_i times in a row (because of local memory constraints), i.e., the client can decide how many times it wants to write to the database as long as this number does not exceed w_i. Thus, the database, which we call d, allows for an arbitrary number of write operations but at least one has to be performed if a client is granted write access.

We define the interaction system $Sys_{\mathrm{DB}}(n)$ where $n \in \mathbb{N} \setminus \{0\}$ as:

$$
\begin{aligned}
&Comp = \{d, 1, \ldots, n\},\\
&A_d = \{backup_d, commit_d, grant_d, inform_d, select_d, update_d\},\\
&A_i = \{commit_i, process_i, read_i, request_i, wait_i, write_i\} \text{ for } 1 \leq i \leq n,\\
&Int = \bigcup_{1 \leq i \leq n} \big\{\{grant_d, request_i\}, \{inform_d, wait_i\}, \{process_i\}, \{select_d, read_i\},\\
&\qquad \{update_d, write_i\}\big\} \cup \big\{\{backup_d\}, \{commit_d, commit_1, \ldots, commit_n\}\big\},\\
&Int_{\mathrm{closed}} = \big\{\big\}.
\end{aligned}
$$

Observe that we fully specified $Sys_{\mathrm{DB}}(n)$'s interaction model with respect to Definition 2.3. Figure 4.1 depicts this model for $n = 2$.

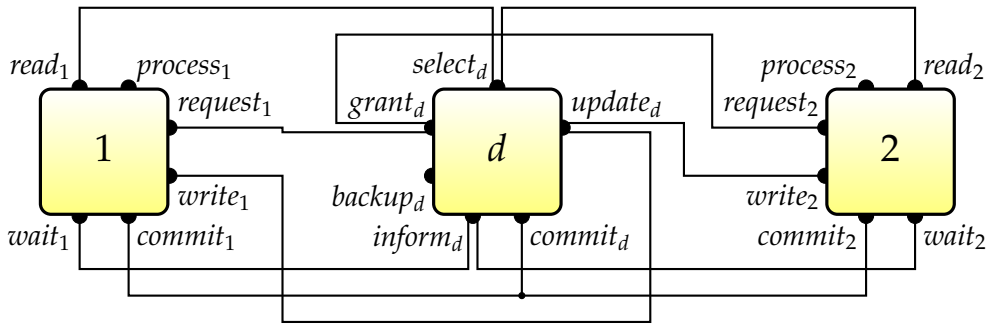
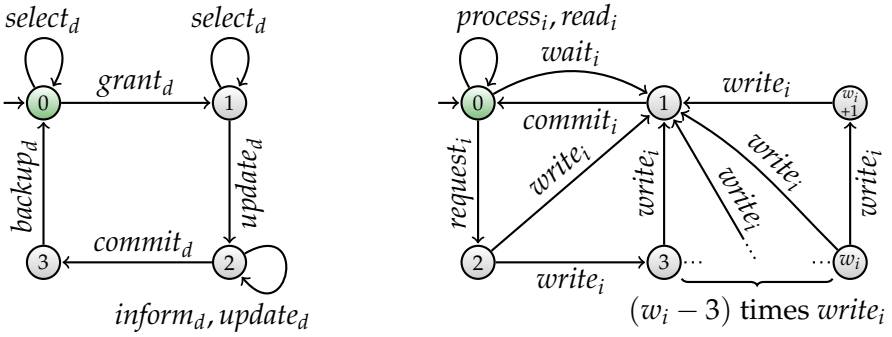


Figure 4.1: Interaction model of the database example $Sys_{\mathrm{DB}}(n)$, depicted for $n = 2$. Overlapping interactions are not connected by default, only if there is a tiny connection dot as for the commit interaction.

We complete $Sys_{\mathrm{DB}}(n)$'s specification by giving its behavioral model (cf. Definition 2.5). The labeled transition systems for the components are given in Figure 4.2 on the following page where Figure 4.2 (a) depicts the behavior $[\![d]\!]$ of the database component d and Figure 4.2 (b) the behavior $[\![i]\!]$ of client component i with $1 \leq i \leq n$. Note that we assume for each labeled transition system $[\![i]\!]$ that the maximal number of consecutive write operations of client i

is known as $w_i \in \mathbb{N}$ and that $w_i \geq 1$ holds for all $1 \leq i \leq n$. Furthermore, for each $[\![i]\!]$ as depicted in Figure 4.2 (b) we assume that if $w_i = 1$, then the depicted states $s_i^{w_i}$ and s_i^1 coincide and the depicted state $s_i^{w_i+1}$ is not contained in the set of states (although state s_i^2 is still contained in spite of $w_i + 1 = 2$). Similarly for $w_i = 2$, the depicted states $s_i^{w_i}$ and s_i^2 coincide as well as the states $s_i^{w_i+1}$ and s_i^3. Finally, for $w_i = 3$ the depicted states $s_i^{w_i}$ and s_i^3 coincide.



(a) Behavior of component d

(b) Behavior of component i with $1 \leq i \leq n$

Figure 4.2: Behavior of the components of interaction system $Sys_{\mathrm{DB}}(n)$

We postpone a detailed analysis of $Sys_{\mathrm{DB}}(n)$. We only want to mentioned that the reachable global behavior of the example already contains more than one million states and roughly fifteen times more transitions for $n = 15$ clients[1] and each $w_i = i$. Thus, even this simple example suffers from state space explosion as discussed in Section 2.2 for relatively small parameters.

We continue with the formal definition of architectural constraints.

4.2 Architectures of Interaction Systems

Typically, architectural constraints are based on graphs that represent the underlying cooperation structure of an interaction systems. In the following, we use common notation from graph-theory which we included in Appendix A for convenience.

4.2.1 Component-Based Architecture

We already mentioned in the introduction of this chapter that various authors studied architectural constraints that are based on the underlying cooperation

[1] The exact numbers are 1 753 104 states and 27 787 745 transitions. Note that for $n \geq 23$ there are more than one billion reachable states and over 25 billion transitions.

structure. This cooperation structure is typically defined by means of an undirected graph where the vertices represent the components and edges exist between any cooperating components, e.g., the architecture of Majster-Cederbaum and Martens [179] is defined this way and also the notion of an architectural topology by Hennicker et al. [137]. Here, we define the *component graph* of an interaction system in the same way.

Definition 4.1 (Component Graph): The *component graph* $G_{\text{comp}} := (V, E)$ of an interaction system *Sys* is defined by the set of vertices $V = Comp$ and the set of edges $E = \{\{i, j\} \in 2^{Comp} \mid \exists\, \alpha \in Int\colon \{i, j\} \subseteq \text{compset}(\alpha)\}$.

As an example for the component graph of an interaction system, we consider again our running example Sys_{MMS} and its extended version $Sys^{(3)}_{\text{MMS}}$ (defined on page 24 and page 64 respectively). The corresponding component graphs of the systems are depicted in Figure 4.3.

(a) Component graph of Sys_{MMS} **(b)** Component graph of $Sys^{(3)}_{\text{MMS}}$

Figure 4.3: Component graph of the merchandise management example

Since we are interested in considering interaction systems that have a particular architecture, we now define two constraints by means of the component graph: the *star-like* and the *tree-like* architecture. First, we define star-like architectures as follows.

Definition 4.2 (Star-Like Architecture): An interaction system *Sys* has a *star-like architecture* if its component graph $G_{\text{comp}} = (V, E)$ is a star in the graph-theoretical sense, i.e., it is connected and $\exists\, m \in Comp \quad \forall\, C \in 2^{Comp}\colon C \in E \implies m \in C$ holds. Component m of the existential quantification is called the *middle component* of *Sys* and all other components are called *border components* of *Sys*.

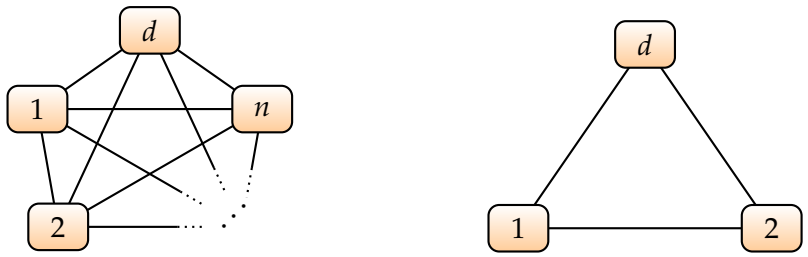
Second, we define tree-like architectures.

Definition 4.3 (Tree-Like Architecture): An interaction system *Sys* has a *tree-like architecture* if its component graph $G_{\text{comp}} = (V, E)$ is a tree in the graph-theoretical sense, i.e., it is connected and $|E| = |V| - 1$ holds. The components whose vertex representation is contained in the center of the component graph, i.e., all $i \in Comp$ with $i \in \text{center}(G_{\text{comp}})$, are called *central components* of *Sys*.

Observe that Sys_{MMS} has a star-like architecture (cf. Figure 4.3 (a)) where component m is the middle component and components c and s are border components. Moreover, $Sys^{(3)}_{\text{MMS}}$ has a tree-like architecture (cf. Figure 4.3 (b)) where component s is the only central component. But, $Sys^{(3)}_{\text{MMS}}$'s architecture is not star-like. Obviously, any interaction system with a star-like architecture also has a tree-like architecture, e.g., Sys_{MMS}'s architecture is also tree-like where component m is the central component. Thus, a tree-like architecture can be seen as a generalization of a star-like architecture.

Both definitions imply that the cooperation structure is acyclic. The intuition behind considering acyclic structures is that in such systems, the presence of circular waiting situations and hence potential deadlocks is reduced. Such requirements of acyclicity exist for many formalisms under various names, e.g., acyclic topologies [137], acyclic architectural types [39], tree networks [49], or tree-like architectures [179]. As mentioned in the introduction, we postpone the exploitation of these architectures to the following chapters.

We want to point out that a tree-like architecture can only be present if all interactions are binary. Otherwise, we find at least three components such that the corresponding vertices in the component graph form a triangle in the graph-theoretical sense, i.e., a simple cycle of length four according to Definition A.8. The database example introduced at the beginning of this chapter illustrates this issue: Figure 4.4 (a) depicts its component graph for an arbitrary number of clients and Figure 4.4 (b) for two clients. Observe that every vertex lies on a simple cycle.



(a) Component graph for arbitrary n **(b)** The same graph for $n = 2$

Figure 4.4: Component graph of interaction system $Sys_{\text{DB}}(n)$

This drawback clearly lies in the definition of the component graph. Note that, for instance, the tree networks of Brookes and Roscoe [49] face the same problem.

Thus, in order to allow for multi-ary interactions on the one side, but still be able to constrain the architecture, we take a look at the cooperation structure of the components.

4.2.2 Cooperation-Based Architecture

We already mentioned the typical drawback of acyclic architectures in the previous section: All cooperations must be binary. However, interaction systems allow for multiway cooperation, which facilitates a very compact and more convenient modeling in many situations since the restriction to binary cooperations typically blows a concrete modeling up in an unnecessary way, e.g., intermediate coordination states are introduced to mimic a multiway synchronization in a binary way.

For this reason, Majster-Cederbaum and Martens [180] extended their notion of a tree-like architecture to be able to deal with multiway cooperations in the setting of interaction systems. They defined a special graph—called *cooperation graph*[2] in the following—in which vertices represent all sets of components that are able to cooperate and whose graph-theoretical property of being a tree defines tree-like interaction systems [180]. To be more precise, the cooperation graph consists of two types of vertices where one type represents each component as a singleton. The other type models any (partial) cooperation situation among the components, i.e., for any two interactions there is a vertex representing the set of components participating in both of them. Thus, all vertices are sets of components, and now, the edges correspond to a set containment relation among these sets, i.e., possible cooperation situations that are related with respect to the set of participating components. Furthermore, two vertices u and v where one set is contained in the other, say $u \subset v$, are only adjacent if there is no further vertex w that lies in between, i.e., $u \subset w \subset v$ does not hold. The intuition behind this requirement is that as few edges as possible should be contained in the graph for the following reason: Multiway cooperations can be dealt with as long as the graph stays acyclic. Indeed, if we have $u \subset w \subset v$ for three vertices (and no further set represented as a vertex lies in between) the edges connecting u with w and w with v already contain the information which cooperation situations are related and a further edge between u and v would introduce a cycle without adding useful information. Now, we define the cooperation graph as follows.

Definition 4.4 (Cooperation Graph): The *cooperation graph* $G_{\text{coop}} := (V, E)$ of an interaction system *Sys* is defined by the set of vertices $V = V_1 \cup V_2$ where $V_1 = \{C \in 2^{Comp} \mid |C| = 1\}$ and $V_2 = \{C \in 2^{Comp} \mid \exists\, \alpha, \beta \in Int\colon C = \text{compset}(\alpha) \cap \text{compset}(\beta) \ \wedge\ C \neq \emptyset\}$ and the set of edges $E = \{\{u, v\} \in 2^V \mid u \subset v \ \wedge\ (\forall\, w \in V\colon u \subset w \implies w \not\subset v)\}$, i.e., there is an edge between two vertices if one is a proper subset of the other and no other vertex lies in between these two sets with respect to set containment.

[2] The graph is called "interaction graph" by Majster-Cederbaum and Martens [180].

As an example for the cooperation graph of an interaction system, we consider again our running example Sys_{MMS}, its extended version $Sys^{(3)}_{\text{MMS}}$, and the database example $Sys_{\text{DB}}(n)$. The corresponding cooperation graphs of the systems are depicted in Figures 4.5, 4.6, and 4.7 respectively. Please note that component and cooperation graphs can be distinguished in our graphical representation by the vertices: In the former, the vertices correspond to the components, and in the latter, they are sets of components.

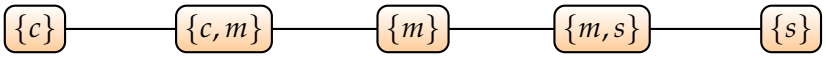


Figure 4.5: Cooperation graph of interaction system Sys_{MMS}

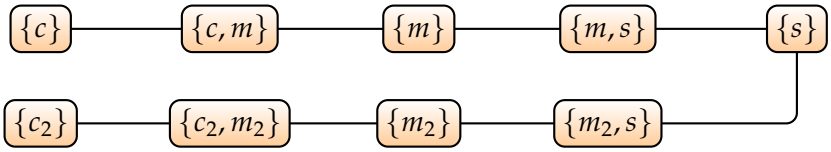
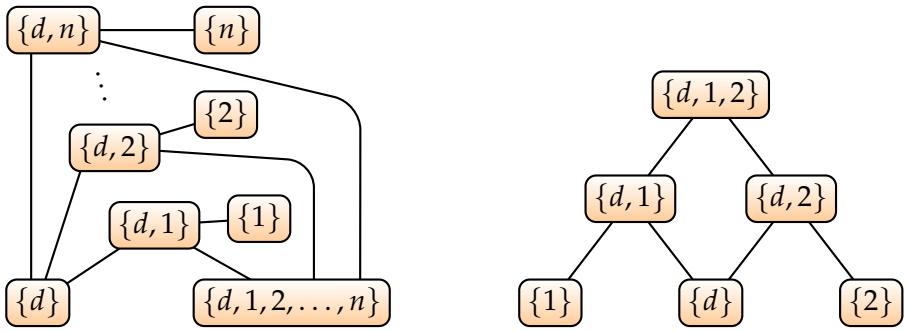


Figure 4.6: Cooperation graph of interaction system $Sys^{(3)}_{\text{MMS}}$



(a) Cooperation graph for arbitrary n **(b)** The same graph for $n = 2$

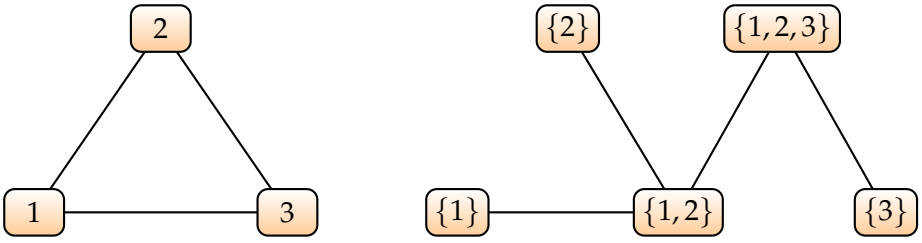
Figure 4.7: Cooperation graph of interaction system $Sys_{\text{DB}}(n)$

Observe that the graph-theoretical structure of the architecture has not changed much for Sys_{MMS} and $Sys^{(3)}_{\text{MMS}}$, they both still have a tree-like architecture (if we define a corresponding architectural constraint based on the cooperation graph). However, the architecture of $Sys_{\text{DB}}(n)$ is now more structured (if we compare Figure 4.4 (a) with Figure 4.7 (a)), and it is striking that the vertex representing component d is the only one that lies on a simple cycle—remember that previously every vertex (or component respectively) lay on a simple cycle.

As mentioned above, Majster-Cederbaum and Martens [180] required that the cooperation graph is acyclic, and they call interaction systems satisfying this

architectural constraint "tree-like". Before we formally define an architectural constraint on the basis of the cooperation graph, we give an example that illustrates a tree-like interaction system as understood by Majster-Cederbaum and Martens [180].

Example 4.5: We consider interaction system $Sys_{\text{bin}}(n)$ for $n = 3$, that was introduced in Example 2.7 on page 32. Here, the three components 1, 2, and 3 represent the three bits of a binary counter. We construct the system's component graph (cf. Definition 4.1) and its cooperation graph (cf. Definition 4.4). These graphs are depicted in Figure 4.8.



(a) The component graph **(b)** The cooperation graph

Figure 4.8: Component and cooperation graph of interaction system $Sys_{\text{bin}}(3)$

Observe that $Sys_{\text{bin}}(3)$'s component graph is not acyclic and thus the interaction system does not have a tree-like architecture as required by Definition 4.3. However, its cooperation graph is acyclic and the interaction system is hence tree-like in the sense of Majster-Cederbaum and Martens [180].

We continue with a novel architecture that is based on the cooperation graph. As already mentioned in Chapter 1, we drop the acyclicity requirement and admit certain cycles. We define this new architectural constraint as follows.

Definition 4.6 (Disjoint Circular Wait Free Architecture): An interaction system *Sys* has a *disjoint circular wait free architecture* if its cooperation graph $G_{\text{coop}} = (V, E)$ is connected and if it holds that on all simple cycles in G_{coop} at most one vertex $v \in V$ correspond to a singleton, i.e., $|v| = 1$, or is an element of V_1 respectively. The components whose vertex representation lies on such a simple cycle are called *cycle components* of *Sys*, i.e., the components $i \in Comp$ with $\{i\} \in \text{cycle}(G_{\text{coop}})$ (cf. Definition A.8).

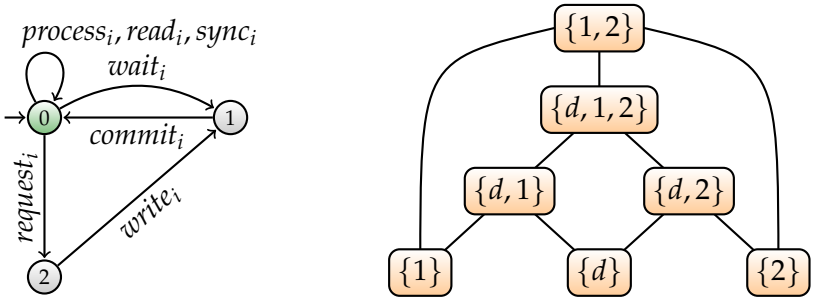
Definition 4.6 can be motivated in the following way with respect to deadlock-freedom: For ensuring freedom from deadlock, we have to consider any way the components are able to cooperate. Clearly, the possibility of deadlocks involving several components is reduced if circular waiting situations among

the components are reduced. Since the components are also related in the cooperation graph, the superimposition of potential waiting situations and the cooperation information could allow us to exclude those waiting situations where the reason of each single wait is completely independent from the other ones, and hence reduce the possibility that deadlocks exist. Here, the vertices of the graph represent all possible cooperation sets, i.e., any vertex whose size is greater than one indicates a possible (partial) cooperation. For instance, consider interaction system $Sys_{\mathrm{DB}}(2)$ and its cooperation graph depicted in Figure 4.7 (b): If component 1 waits for component d, d waits for 2, and 2 waits for 1, the graph tells us that no cooperation situation exists that is independent of d, i.e., we cannot find three waiting situations where the reason of each single wait is completely independent from the other ones. As we see in Chapter 6, the exclusion of the above mentioned waiting situations then allows for the establishment of a condition for deadlock-freedom of the whole system by an analysis of only small system parts.

But, let us return to the architecture of our example interaction systems. Clearly, Sys_{MMS} and $Sys^{(3)}_{\mathrm{MMS}}$ have a disjoint circular wait free architecture since no simple cycle is contained in the corresponding cooperation graphs (cf. Figure 4.5 and Figure 4.6 respectively). The set of cycle components is therefore empty for both systems. Obviously, any interaction system with a tree-like architecture also has a disjoint circular wait free architecture. Thus, a disjoint circular wait free architecture can be seen as a generalization of a tree-like architecture. For the database example $Sys_{\mathrm{DB}}(n)$, we already mentioned above that component d is the only one that lies on a simple cycle (cf. Figure 4.7 (a)). Thus, on all simple cycles in the cooperation graph at most one vertex corresponds to a singleton and the system has a disjoint circular wait free architecture. Note that the database example is not tree-like in the sense of Majster-Cederbaum and Martens [180] because its cooperation graph contains a simple cycle.

We give an additional example in order to further illustrate disjoint circular wait free architectures.

Example 4.7: We modify our running example $Sys_{\mathrm{DB}}(n)$ and allow for an additional synchronization among all clients, which yields interaction system $Sys_{\text{DB-Sync}}(n)$ that is depicted in Figure 4.9 on the facing page. Each client component i is equipped with a further action $sync_i$ that is employed as a self-loop in the initial state (cf. Figure 4.9 (a) where we depicted the case $w_i = 1$ for convenience). The additional interaction $\{sync_1, \ldots, sync_n\}$ models the synchronization among all clients and is added to $Sys_{\mathrm{DB}}(n)$'s interaction set—which we call Int_{DB} in Figure 4.9. The behavior of the database component d is not modified, i.e., we use the same behavior as depicted in Figure 4.2 (a).



(a) Behavior of component i, $1 \leq i \leq n$ **(b)** Cooperation graph for $n = 2$

Figure 4.9: Inter. system $Sys_{\text{DB-Sync}}(n)$ with $Int = \{\{sync_1, \ldots, sync_n\}\} \cup Int_{\text{DB}}$

Now, consider $Sys_{\text{DB-Sync}}(n)$ with $n = 2$ and its cooperation graph depicted in Figure 4.9 (b): If component 1 waits for component d, d waits for 2, and 2 waits for 1, the graph tells us that the components could be waiting in a row such that the reason of each single wait is completely independent from the other ones. As discussed above, such a situation is excluded by a disjoint circular wait free architecture—which is not present in $Sys_{\text{DB-Sync}}(2)$ because its cooperation graph contains a simple cycle where more than one vertex representing a component lies on (cf. Definition 4.6).

This ends our introduction of architectures for interaction systems. In the next section, we consider how the architecture of a given interaction system can be determined in an automatic and efficient way.

4.3 Determining an Interaction System's Architecture

Architectural constraints are only useful if we can determine whether a given interaction system satisfies the constraint in an efficient way, i.e., in our case in polynomial time. If this inspection is more costly and we plan to exploit it for property verification, it should not be the case that a direct approach as, for instance, discussed in Chapter 2 is more efficient than the check for an architecture.

We omit here to give algorithms for computing the component and cooperation graphs since they can be straightforwardly implemented. For the latter one, we give an algorithm in Appendix B (cf. Algorithm B.5) for self-containedness because we need to refer to this algorithm in the sequel.

However, we shortly want to discuss the sizes of the two graphs with respect to a given interaction system *Sys* (because we need them as input in the

following). The size of a component graph $G_{\text{comp}} = (V, E)$ is $|V| = |Comp|$ and $|E| \leq \binom{|Comp|}{2} \leq |Comp|^2$ because it is possible that every pair of components cooperates. For the cooperation graph $G_{\text{coop}} = (V, E)$ with $V = V_1 \cup V_2$ (cf. Definition 4.4) we have: $|V_1| = |Comp|$ and $|V_2| \leq \binom{|Int|}{2} \leq |Int|^2$ since the vertices in V_2 are constructed by considering all pairs of interactions (and their sets of participating components). Thus, we have $|V| \in \mathrm{O}(|Int|^2 + |Comp|)$. Since an undirected graph consists of at most $\binom{|V|}{2}$ edges, we can conclude that $|E| \in \mathrm{O}(|Int|^4 + |Comp|^2)$ holds. Thus, both graphs have a polynomial number of vertices and edges with respect to the given interaction system.

4.3.1 Checking for Star-Like and Tree-Like Architectures

Star- and tree-like architectures can be detected very efficiently. For checking whether an architecture is star-like, we simply have to compute the union and the intersection of all edges of the component graph. The union has to be equal to the set of vertices and the intersection has to contain a single vertex, which is the middle component, if there are more than two components, otherwise the intersection can contain two components if there is one edge.

For checking whether an architecture is tree-like, we have to check whether the component graph is connected and whether the number of edges minus one equals the number of vertices [89, Corollary 1.5.3]. This can be carried out in linear time by a depth-first search starting at an arbitrary vertex [76, Section 22.3]. The central components (cf. Definition 4.3) can also be computed in linear time if the graph is a tree in the graph-theoretical sense, cf. Section 7.2.1 of the book of Wu and Chao [264].

Summarizing, both the check for a star-like and a tree-like architecture can be carried out in polynomial time in the number of components and interactions. The check for a disjoint circular wait free architecture is a little bit more involved but can still be checked efficiently as we discuss in the following section.

4.3.2 Checking for Disjoint Circular Wait Free Architectures

In the following, we show how we can determine whether a given interaction system has a disjoint circular wait free architecture in polynomial time in the number of components and interactions.

Let $G_{\text{coop}} = (V, E)$ be the cooperation graph of an arbitrary interaction system *Sys*. We assume that G_{coop} is connected which can efficiently be determined as discussed in the previous section—otherwise, i.e., if G_{coop} is not connected, the

architecture cannot be disjoint circular wait free (cf. Definition 4.6). In order to check whether G_{coop} satisfies our architectural constraint of disjoint circular wait freedom—recall that on every simple cycle in G_{coop} at most one vertex v with $|v| = 1$ may occur—we model this question as a network flow problem. Such a network is usually build upon a *directed graph* which is similar to an undirected graph (cf. Definition A.5) but the edges are tuples contained in the Cartesian product of the set of vertices, i.e., for the set of edges E' holds $E' \subseteq V' \times V'$ where V' denotes the set of vertices. We refer the reader to the book of Ford and Fulkerson [104, Chapter 1] or to the book of Cormen et al. [76, Section 26.1] for a formal definition of flow networks.

Now, we define the *directed* graph $G' = (V', E')$ on the basis of G_{coop} with

$$V' = \{v_{\text{in}} \mid v \in V\} \cup \{v_{\text{out}} \mid v \in V\}$$
$$E' = \{(v_{\text{in}}, v_{\text{out}}) \mid v \in V\} \cup \{(v_{\text{out}}, w_{\text{in}}), (w_{\text{out}}, v_{\text{in}}) \mid \{v, w\} \in E\}$$

and the capacity function $c\colon E' \to \mathbb{N}$ with $c(e) = 1$ for all $e \in E'$.

For a pair of vertices $s, t \in V$ with $|s| = |t| = 1$, i.e., the vertices represent two of *Sys*'s components in the cooperation graph, we now consider the flow network $N_{s,t} = (G', s_{\text{in}}, t_{\text{in}}, c_{s,t})$ where s_{in} is the source, t_{in} is the sink, and $c_{s,t}$ is the same capacity function as c except that the capacity of the edge $(s_{\text{in}}, s_{\text{out}})$ is increased by one, i.e., $c_{s,t}((s_{\text{in}}, s_{\text{out}})) = 2$.

The idea behind this flow network is that if we find a certain maximum flow in $N_{s,t}$, we can conclude the existence of a certain simple cycle in G_{coop}. This is formalized in the following lemma.

Lemma 4.8: Let G_{coop} be a cooperation graph of an interaction system *Sys* and let $i, j \in \mathit{Comp}$ be two components. There is a simple cycle in G_{coop} on which the corresponding vertices $\{i\} = s$ and $\{j\} = t$ lie if and only if a maximum flow of value 2 exists in the associated flow network $N_{s,t} = (G', s_{\text{in}}, t_{\text{in}}, c_{s,t})$.

A formal proof of Lemma 4.8 can be found in Appendix F on page 250.

Lemma 4.8 offers a convenient way to determine whether a given interaction system obeys to our architectural constraint. We proceed with our database example to demonstrate this way. Figure 4.10 on the following page depicts the primed version of the cooperation graph of $\mathit{Sys}_{\text{DB}}(n)$ for two clients (cf. Figure 4.7 (b) for the original graph).

In order to check for a disjoint circular wait free architecture, we now need to consider the source-sink pairs $(\{d\}_{\text{in}}, \{1\}_{\text{in}})$, $(\{d\}_{\text{in}}, \{2\}_{\text{in}})$, and $(\{1\}_{\text{in}}, \{2\}_{\text{in}})$ in each case in an appropriate network where the capacity between the source and the (unique) successor vertex is set to two. Running these network flow

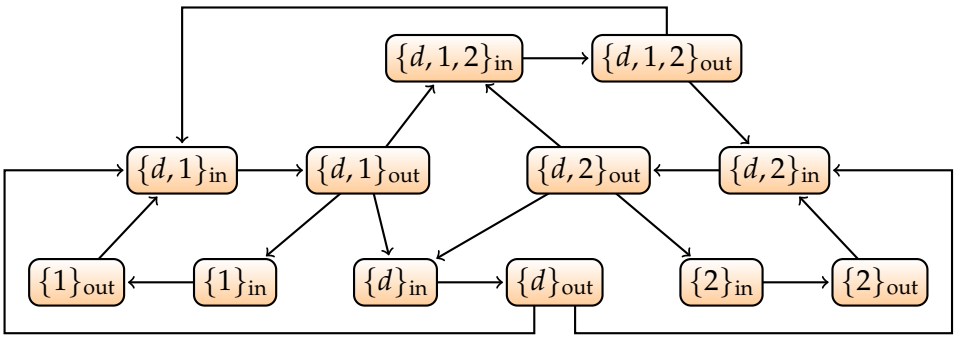


Figure 4.10: The transformation of the cooperation graph of the database example $Sys_{\text{DB}}(n)$ for $n = 2$ (cf. Figure 4.7 (b)) into a flow network. All edges have the capacity of 1. For a particular check, two "in" vertices are designated as source and sink respectively and the capacity of the edge connecting the source with the corresponding "out" vertex is increased by one.

computations yields that our example has a disjoint circular wait free architecture since we can refute the existence of any simple cycle where two vertices representing components lie on.

Of course, not all pairs have to be considered as described above. Any component vertex that has only one neighbor in the cooperation graph cannot be part of a simple cycle, e.g., for $Sys_{\text{DB}}(n)$ we do not need to compute any network flows since all vertices representing components except the vertex representing d have only one neighbor (cf. Figure 4.7 (a)). Thus, no appropriate simple cycle can be found in this cooperation graph.

We summarize our findings and the procedure for an arbitrary interaction system in the following theorem.

Theorem 4.9: Let Sys be an interaction system and let G_{coop} be its cooperation graph. Sys has a disjoint circular wait free architecture if and only if for all unordered pairs of components $i, j \in Comp$, whose vertex representations in G_{coop} have at least two neighbors, it holds that no maximum flow of value 2 exists in the associated flow network $N_{\{i\},\{j\}}$.

A formal proof of Theorem 4.9 can be found in Appendix F on page 251.

Thus, we can also apply Theorem 4.9 for interaction system $Sys_{\text{DB}}(n)$ and conclude that it has a disjoint circular wait free architecture without any network flow computation. Nevertheless, we provide a heuristic if the situation is not as simple as for $Sys_{\text{DB}}(n)$. In the treatment of pairs, we should start with vertices that have many neighbors since these are more likely to be part of simple cycles. This is helpful, since once we identified such a pair, we can

stop the search for any other pairs of vertices representing components and report that the interaction system in question does not satisfy our architectural constraint.

We analyze the costs of an application of Theorem 4.9. Here, we omit to give an algorithm that implements this application but refer the interested reader to Algorithm B.6 given in Appendix B. In the following, let $n = |V|$ denote the number of vertices of the cooperation graph G_{coop} and $m = |E|$ the number of its edges. Since every vertex is doubled for the directed version $G' = (V', E')$ of G_{coop}, we get $|V'| = 2n$, and thus $|V'| \in \mathrm{O}(n)$. Similarly, since every edge is also doubled and an edge between the "in" and "out" version of a vertex is introduced, we get $|E'| = 2m + n$. Because we assumed that G_{coop} is connected, we know that $m \geq n - 1$ holds. Thus, we have $|E'| \in \mathrm{O}(m)$.

Since all edges in the flow network have integral capacities, we can use the well-known Ford–Fulkerson algorithm [103] to find a maximum flow. Its runtime is $\mathrm{O}(M \cdot F)$ where M is the number of edges in the network and F is the value of a maximum flow, since an augmenting path can be found in time $\mathrm{O}(M)$ and such a path increases the flow by at least one. A proof of this upper bound can be found in Section 26.2 of the book of Cormen et al. [76]. Best to our knowledge, this bound is still optimal if the flow value is treated as a parameter. We want to mention work by Karger and Levine [153] that show how the augmenting path computation can be speeded up in case the network is based on a sparse undirected graph. But for our application of flow networks, we currently do not know how we can show the existence of the two distinct paths in the proof of Lemma 4.8 in an undirected network.

Here, this yields an upper bound for a maximum flow determination of $\mathrm{O}(m)$ for a particular network $N_{s,t}$ because $|f| \leq 2$ for any flow f in $N_{s,t}$—cf. the proof of Lemma 4.8. Since there are maximal $\binom{|Comp|}{2}$ unordered pairs of vertices representing components in a cooperation graph, the total running time is bounded by $\mathrm{O}(|Comp|^2 \cdot m)$.

In order to express this bound in terms of the input size, we need to determine the maximal number of edges that a cooperation graph can have, which we already did above at the beginning of Section 4.3. There, we found out that the number of edges is bounded by $\mathrm{O}(|Int|^4 + |Comp|^2)$. Thus, a rough upper bound of the check for disjoint circular wait free architectures is $\mathrm{O}(|Comp|^2 \times |Int|^4 + |Comp|^4)$, which is polynomial in the size of the input (which is our aim throughout the thesis). Although this bound seems very high, the cooperation graph is much smaller in case the interaction system has a disjoint circular wait free architecture and we have to perform a flow computation for all pairs of components that have more than one neighbor. Otherwise, i.e., if the system does not have a disjoint circular wait free architecture, one flow computation

finding a flow of value two is sufficient which is reasonable if we start with pairs of components that have many neighbors.

Finally, we take a look at how the cycle components of the architecture can be determined (cf. Definition 4.6). We use the concept of "bridges" of a graph, which are edges whose removal destroys the connectivity of a given connected graph. An important observation with respect to cycles is that an edge is a bridge if and only if it does not lie on a simple cycle (cf. [76, Chapter 22] or [89, Section 1.4]). Thus, to get to know the cycle components, we simply need to remove all bridges in the cooperation graph and then look for all vertices that represent a component and still have more than one incident edge. Fortunately, Tarjan [251] gives an algorithm for finding all bridges in time $O(|V| + |E|)$. Here, we refer to Appendix B where we give an algorithm that computes the set of cycle components in linear time (cf. Algorithm B.7).

This completes the algorithmic treatment of our architectural constraints. We are now ready to use these constraints as requirements for interaction systems in order to exploit the additional structure, that is introduced among the cooperations, to derive efficient verification techniques. Before we dive into such techniques, which we postpone to the following chapters, we examine whether an architectural constraint has an effect on the computational complexity of a decision problem as mentioned in the beginning of this chapter. We begin with our coarsest architectural constraint, viz. disjoint circular wait free architectures.

4.4 Classifying Disjoint Circular Wait Free Architectures

In the previous sections, we learned that the determination of the architecture of an interaction system can be carried out very efficiently. A natural question that arises now is whether non-constrained or arbitrary interaction systems are more powerful or more computationally complex than, e.g., systems with a disjoint circular wait free architecture?

In this section, we provide a linear-time transformation for arbitrary interaction systems that modifies the cooperations in such a way that the resulting interaction system has a disjoint circular wait free architecture and exhibits isomorphic behavior (up to transition relabeling, cf. Definition 2.18), i.e., the behavior is in fact identical from a verification point of view.

The existence of such a transformation has two important consequences. On the one hand, it shows that the class of interaction systems with a disjoint circular wait free architecture provides a starting point for new verification

techniques since if we want to verify a property of an arbitrary interaction system and only know ways to tackle this problem by constructing the global state space, then we can transform the system into one with a disjoint circular wait free architecture, and potentially, the new cooperation structure offers a line of a attack that does not rely on constructing the global state space.

On the other hand, the transformation allows us to show that computational complexity results for arbitrary interaction systems also hold in systems having a disjoint circular wait free architecture. For instance, Majster-Cederbaum and Minnameier [183] provide various complexity results for decision procedures of properties of interaction systems and, as we discuss in the following, all of them also hold for the class of systems with a disjoint circular wait free architecture.

We proceed as follows: First, we present the transformation and show that it can be performed in linear time and that it yields an interaction system which has a disjoint circular wait free architecture and which has isomorphic behavior up to transition relabeling. Second, we deal with the complexity issues implied by this result.

4.4.1 Transforming Interaction Systems To Ensure A Disjoint Circular Wait Free Architecture

Let Sys be an arbitrary interaction system. Assume that x is a component that is not part of Sys, i.e., $x \notin Comp$ and $A_x \cap Act = \emptyset$, and that has one action $fresh_x$, i.e., $A_x = \{fresh_x\}$. Further, we choose for each component i of Sys a fresh action that we denote by $fresh_i$ in each case, i.e., we require that $fresh_i \notin A_i$ for all $i \in Comp$.

We construct an interaction system Sys' in the following way: We set $Comp' = Comp \cup \{x\}$, $A'_i = A_i \cup \{fresh_i\}$ for all components $i \in Comp$, $A'_x = A_x$, $Int' = \{\alpha \cup \{fresh_x\} \mid \alpha \in Int\} \cup \{\{fresh_x, fresh_i\} \mid i \in Comp\}$, and $Int'_{\text{closed}} = \{\alpha \cup \{fresh_x\} \mid \alpha \in Int_{\text{closed}}\}$, i.e., we add x and its action set as a new component, add a fresh action to the action set of every component in $Comp$, add x's action $fresh_x$ to every interaction, introduce new interactions that consist of the action $fresh_x$ of the new component and the fresh action $fresh_i$ of a component $i \in Comp$ in each case, and adjust the set of closed interactions. Similarly, we take over all labeled transition systems for the behavioral model of Sys' and add for component x a labeled transition system that consists of one initial state s^0_x with a self-loop labeled by $fresh_x$. In particular, a new action $fresh_i$ is not employed in the local behavior $[\![i]\!]$ for all $i \in Comp$.

As an example for the transformation, consider interaction system $Sys_{\text{DB-Sync}}(n)$

that was specified in Example 4.7 on page 96. The cooperation graph of this system is depicted in Figure 4.9 (b) for $n = 2$ (cf. page 97) and shows that $Sys_{\text{DB-Sync}}(2)$ does not have a disjoint circular wait free architecture. If we now apply our transformation, the cooperation graph becomes the one depicted in Figure 4.11, which shows that the transformed system has a disjoint circular wait free architecture.

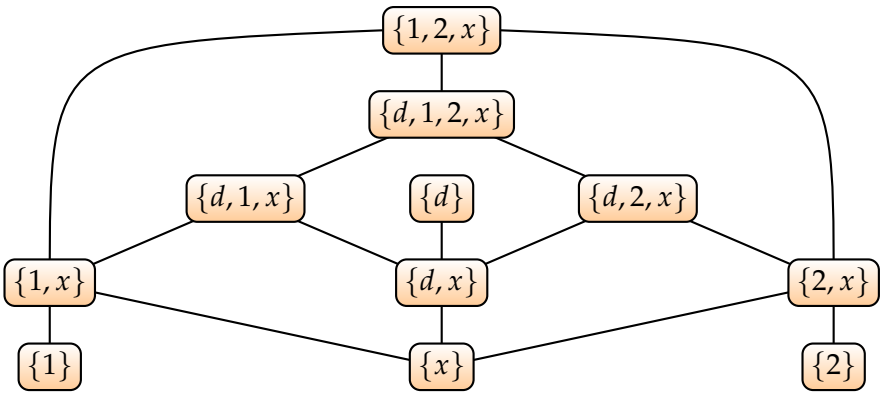


Figure 4.11: The cooperation graph after transforming $Sys_{\text{DB-Sync}}(2)$

Next, we show that this result holds for all interaction systems, i.e., the transformation always yields a systems with a disjoint circular wait free architecture. In the following, we denote a transformed interaction system as the primed version of the original one, e.g., Sys becomes Sys'.

Lemma 4.10: Let Sys be an arbitrary interaction system. Applying our transformation yields an interaction system Sys' that has a disjoint circular wait free architecture.

A formal proof of Lemma 4.10 can be found in Appendix F on page 251.

We proceed by showing that our transformation yields isomorphic (up to transition relabeling) global behavior (cf. Definition 2.18). Since our transformation also introduces new interactions that do not occur in the original interaction system, we have to restrict the alphabet of the global behavior of the transformed system. We provide an argument in the proof of the following lemma that this restriction does not affect the isomorphism.

Lemma 4.11: Let Sys be an arbitrary interaction system. Applying our transformation yields an interaction system Sys' such that the global behavior $[\![Sys]\!]$ is isomorphic up to transition relabeling to $[\![Sys']\!]$ if we restrict the alphabet of $[\![Sys']\!]$ to those interactions that do not merely consist of fresh actions.

A formal proof of Lemma 4.11 can be found in Appendix F on page 252.

We summarize the properties of our transformation in the following theorem and also prove that the transformation can be performed in linear time.

Theorem 4.12: An arbitrary interaction system can be transformed in linear time into an interaction system with a disjoint circular wait free architecture such that the behavior of the new system is isomorphic up to transition relabeling to the behavior of the original one.

A formal proof of Theorem 4.12 can be found in Appendix F on page 253.

In the next section, we discuss complexity issues that are implied by the above result.

4.4.2 Complexity Issues

An important consequence of the existence of the transformation presented above is that most decision problems have the same complexity for the class of arbitrary interaction systems and the class of systems with a disjoint circular wait free architecture. For instance, Majster-Cederbaum and Minnameier [183] entitled their paper "Everything Is PSPACE-Complete in Interaction Systems", and with our transformation, we can extend their various results to the class of interaction systems with a disjoint circular wait free architecture. Note that also the problems of detecting deadlocks and livelocks as introduced in Chapter 2 fall into this category.

We formalize this consequence in the following corollary where we assume that a decision problem in question is closed under the isomorphism of Definition 2.18. For the interesting properties such as deadlock-freedom, this is no restriction at all since those are closed under isomorphism up to transition relabeling.

Corollary 4.13: Let a decision problem for the class of (arbitrary) interaction systems be given that is closed under isomorphism up to transition relabeling. The problem is linear-time many-to-one reducible to the same decision problem for the class of interaction systems with a disjoint circular wait free architecture. Thus, these problems belong to the same complexity class (if the class is closed under linear-time many-to-one reducibility).

A formal proof of Corollary 4.13 can be found in Appendix F on page 253.

Corollary 4.13 allows us to determine the complexity of several decision problems for interaction systems with a disjoint circular wait free architecture. Since the complexity class PSPACE is closed under linear-time many-to-one

reductions, we can state that, e.g., deadlock-freedom is PSPACE-complete for interaction systems with a disjoint circular wait free architecture because deadlock-freedom is PSPACE-complete in (arbitrary) interaction systems as discussed in Section 2.3.1. For more decision problems in interaction systems, we refer the reader to the work of Majster-Cederbaum and Minnameier [183] and to Section 2.3 of this thesis.

Finally, we conclude that 'everything' is PSPACE-complete in interaction systems obeying to the architectural constraint of disjoint circular wait freedom. In the next section, we take a similar look at the other two architectural constraints, viz. tree- and star-like architectures.

4.5 Classifying Tree- and Star-Like Architectures

We can naturally ask whether we can also classify tree- and star-like architectures in a similar way as in the previous section. Unfortunately, a strong classification as for interaction systems with a disjoint circular wait free architecture is not known, i.e., we do not know whether the verification of all reasonable properties in, say, interaction systems with a tree-like architecture has the same computational complexity as in systems with a non-constrained or disjoint circular wait free architecture.

However, some work has been done in this direction. Majster-Cederbaum and Semmelrock [185] discuss a reduction from deciding whether a quantified boolean formula is true (or not) to the question whether a certain global state is reachable in a specially crafted interaction system that has a tree-like architecture. Since the former problem is PSPACE-complete [108, pages 171–172], this reduction establishes the PSPACE-hardness of the reachability question. Now, this reachability question can be used to also show that the decision problem of deadlock detection is PSPACE-hard in interaction system with a tree-like architecture, although this is not explicitly shown in the cited paper. In order to prove this claim, one has to show that the presented construction yields a deadlock-free interaction system, which can be ensured by adding a singleton interaction containing an action that is added as a self-loop to each state of one of the components, and then add a deadlock—which, of course, also involves the previously modified component—that is reachable if and only if the quantified boolean formula in question is false, i.e., the deadlock is only present in the reachable part of the global behavior if the formula does not hold. Then, if one can decide the deadlock-freedom of this system, one can also decide the satisfiability of the formula. Together with the algorithm that decides deadlock-freedom in polynomial space in interaction

system (cf. Section C.1), the PSPACE-completeness of deadlock detection in interaction systems with a tree-like architecture follows. Moreover, the authors show that the computational complexity of the decision problem whether a certain component is able to make progress (cf. Section 2.3.4) is PSPACE-complete in interaction systems with a tree-like architecture by adjusting their reduction for reachability that we mentioned above.

We want to point out that this illustrates the benefit of our isomorphism result (cf. Theorem 4.12 and Corollary 4.13) for disjoint circular wait free architectures, since otherwise, we have to search for reductions for each property in question as it is currently the case for interaction systems with a tree-like architecture. It is thus a challenging and interesting question whether a similar isomorphism result—or a weaker one with respect to, say, bisimilarity—can be derived for the other architectural constraints as well.

Another way of attacking this question is to identify the finest architecture and prove complexity results for this case—where it is not clear what is considered to be the finest architecture. Nevertheless, this idea can be justified as follows: Clearly, a system with a star-like architecture also has a tree-like architecture and also a disjoint circular wait free one. Thus, if the decision problem for a certain property is, say, PSPACE-complete for interaction systems with a star-like architecture, then this result carries over to the other architectures because the respective classes of interaction systems are strictly included in each other. In this direction, Majster-Cederbaum and Semmelrock [186] show that the decision problem whether a certain global state is reachable is PSPACE-complete in interaction systems with a star-like architecture, which then also shows (or supports) the corresponding decision problems for the other architectures. However, one still has to find suitable reductions for each property in question.

Here, we do not further address these complexity issues because we want to continue with the exploitation of architectural constraints to derive efficient verification techniques. But before that, we take a look at related work on architectures.

4.6 Related Work

We already mentioned above that several authors defined architectural constraints with respect to their formalisms similarly to our component graph (cf. Definition 4.1), e.g., acyclic topologies in the semantic model of Java/A by Hennicker et al. [137], acyclic architectural types in the architecture description language PADL by Bernardo et al. [39], or tree networks in the context of the

process algebra CSP by Brookes and Roscoe [49]. For interaction systems, we already discussed the tree-like interaction systems of Majster-Cederbaum and Martens [179, 180].

As mentioned in Chapter 1, the importance of the modularity and structure of software systems was already emphasized by Dijkstra [91], Brooks and Iverson [52], and Parnas [223] in the 1970s. Some of these ideas are similar to our interpretation of architecture, for instance, Brooks and Iverson [52] consider the notion of architecture as the "conceptual structure of a computer ... as seen by the programmer." The work on software architectures as design patterns for structured software development such as the concept of architectural styles in architecture description languages [3, 6, 110, 242] go beyond our interpretation but have the same foundation on a certain level of abstraction. For instance, Garlan and Shaw [110, Section 3] write that a common framework to view architectural styles graphically leads to "a view of an abstract architectural description as a graph in which the nodes represent the components and the arcs represent the connectors" which is similar to our component graph. But, these styles go beyond stipulating topological constraints, such as being acyclic, on a system's architecture, e.g., they also allow for assigning roles to the component. As an example, in the blackboard architectural style [110, Section 3.5] a central component has the task to collect knowledge from computing entities called knowledge sources that do not interact directly. While this corresponds to a star-like architecture in our setting, this style also requires that the blackboard triggers the computing entities if new data is collected, i.e., it can be clearly distinguished from a traditional database. Here, the architectural style also assigns special roles to the components which we did not address in our treatment of the architecture of interaction systems.

Thus, software architecture can embrace more than the cooperation structure. We conclude with the definition of software architecture by Bass et al. [28, Chapter 3] who write: "The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." This definition is at the level of our interaction system, i.e., it also includes the components which are called software elements. However, if we consider interaction systems together with the architectural constraints of this chapter, we have a formal model for software architectures that is in line with this definition.

This ends our discussion of architectures for interaction systems. In the next section, we summarize the chapter and address some ideas for future work.

4.7 Summary and Future Work

In this chapter, we introduced the concept of architectures for interaction systems by means of architectural constraints. We showed that detecting whether a certain constraint holds for a given interaction system can be carried out efficiently, i.e., we can use the detection algorithms as a pre-processing step in the following without loosing our goal of polynomial-time approaches. Moreover, we learned that restricting interaction systems to satisfy one of our constraints does not affect the computational complexity of property verification, in particular for the property of deadlock-freedom. However, as we see in the following chapters, the architectural constraints offer lines of attacking these hard problems by means of sufficient conditions that imply the property under additional assumptions that can be checked efficiently.

We already pointed out some directions for future work, e.g., the classification of star- and tree-like architectures should be more researched on: Maybe there is a way of transforming an arbitrary interaction system into one with a tree-like architecture in a behavior preserving way. We can also extend the idea of a disjoint circular wait free architecture: Currently, we require that on each simple cycle at most one component vertex occurs. This could be extended to a notion of k-disjoint circular wait freedom such that we allow for up to k component vertices to be present on simple cycles. Then, we can potentially exploit this situation by only needing to analyze the global behavior of sets of components of size k.

Furthermore, we can extend our composition operator of Chapter 3 to be sensitive for architectures. For instance, if we have two interaction systems with a tree-like architecture and require that their composition with respect to a given composition information also has a tree-like architecture, special composition rules could establish this property in a convenient way, i.e., without constructing the component graph of the composite interaction system. Such rules could, e.g., restrict the composition information by only allowing new interactions that are compliant to the architectural constraint.

In the next chapter, we consider how we can use the structure implied by architectural constraints for efficient property verification in interaction systems.

Chapter 5

Compositional Reduction

In the previous chapter, we introduced architectural constraints for interaction systems, viz. star-like, tree-like, and disjoint circular wait free architectures. However, we also learned that these architectural constraints do not affect the computational complexity of property verification in general (although their presence can be checked in polynomial time), e.g., only requiring that an interaction system has a tree-like architecture does not help to show its deadlock-freedom efficiently without additional assumptions, because deciding whether an interaction system is deadlock-free is PSPACE-complete for both, interaction systems with a tree-like architecture and non-constrained interaction systems (cf. Section 4.5).

In this chapter, we consider such additional assumptions. Here, we do not focus on a certain property, instead we are interested in—as the title of this chapter already reveals—compositional reduction. The idea can be summarized as follows: We want to identify certain components in a given interaction system that we can safely ignore in future verification steps because they neither affect the global behavior of the system, nor do they alter the validity of system properties in question, i.e., they are non-interfering for the given interaction system. In order to identify such components, we require that the interaction systems satisfies one of our architectural constraints and that certain equivalences between the behavior of certain system parts, which are accessible with our subsystem construction operator, hold. In Chapter 2, we chose branching bisimilarity with explicit divergence as our "working" equivalence for interaction systems (cf. Definition 2.15) and thus any equivalence of this chapter is also of this type.

We proceed with a discussion of our running example Sys_{MMS}, the merchandise management system, in order to illustrate the main idea for such a reduction approach.

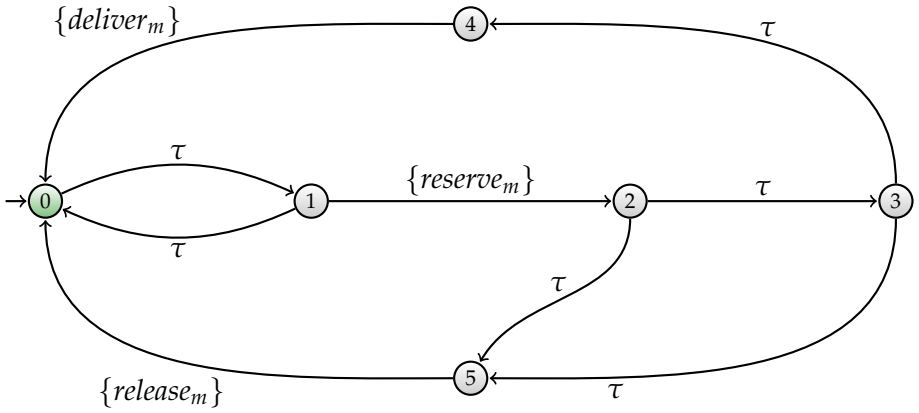


Figure 5.1: Global behavior of interaction system $Sys_{\text{MMS}}[\{m, c\}] \setminus\setminus \hat{I}_{m,c}$

5.1 The Idea behind Compositional Reduction

Consider again our running example Sys_{MMS} and suppose we want to reduce the system by exploiting equivalences to identify non-interfering components as mentioned above. The reduction should ensure that if a property (such as deadlock-freedom) holds in the reduced version, then it also holds in the original system. We showed in the previous chapter that the architecture of Sys_{MMS} is star-like (cf. Definition 4.2 and Figure 4.3 (a)), i.e., we have the management component as the middle component and the customer and storage component as two border components.

A first idea to identify non-interfering components is to check whether the joint behavior—that can be obtained via subsystem construction—of the middle component and a border component affects the behavior of the middle component, i.e., we want to check whether the border component is non-interfering and can be ignored. Of course, we have to abstract from the cooperation of the middle and the border component since in the system that consists only of the middle component, no such cooperation can be observed—simply because the cooperation partner is not part of this system.

Regarding our running example, we thus build the two interaction systems $Sys_{\text{MMS}}[\{m, c\}] \setminus\setminus \hat{I}_{m,c}$ and $Sys_{\text{MMS}}[\{m\}] \setminus\setminus \hat{I}_{m,c}$ where the set $\hat{I}_{m,c}$ contains all interactions that are used for cooperation of m and c and all actions of m that are used in such an interaction as singletons, i.e., we have:

$$\begin{aligned}\hat{I}_{m,c} &:= \big(Int[\{m, c\}] \setminus Int[\{m\}]\big) \cup \big\{\alpha \in Int[\{m\}] \mid \exists\, \beta \in Int[\{m, c\}] \colon \alpha \subset \beta\big\} \\ &= \big\{\{abort_c, cancel_m\}, \{ask_c, order_m\}, \{buy_c, pay_m\}, \{refund_c, reimburse_m\}, \\ &\quad \{cancel_m\}, \{order_m\}, \{pay_m\}, \{reimburse_m\}\big\}.\end{aligned}$$

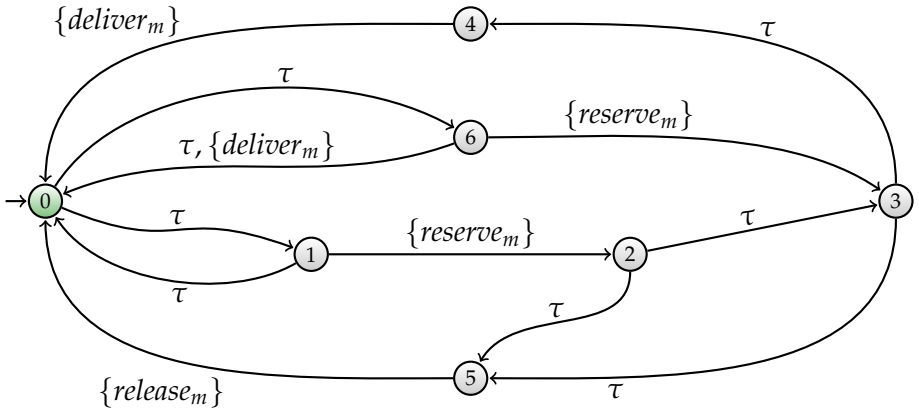


Figure 5.2: Global behavior of interaction system $Sys_{\text{MMS}}[\{m\}] \setminus\!\setminus \hat{I}_{m,c}$

The global behaviors of the two systems are depicted in Figure 5.1 and Figure 5.2 respectively. If we now want to get rid of the border component representing the customer in a property preserving way, we can check whether these two systems are equivalent, i.e., whether

$$Sys_{\text{MMS}}[\{m,c\}] \setminus\!\setminus \hat{I}_{m,c} \overset{?}{\approx}_b^{\Delta} Sys_{\text{MMS}}[\{m\}] \setminus\!\setminus \hat{I}_{m,c}$$

holds. Note that if this equivalence holds, any property that is preserved by branching bisimilarity with explicit divergence and that holds in the system of the left-hand side also holds in the system of the right-hand side, i.e., the customer component does not influence the "interesting" behavior of the management component.

But, from the comparison of the labeled transition system in the figures, we already see that this is not the case because we cannot find an equivalent state for the sixth state of $[\![Sys_{\text{MMS}}[\{m\}] \setminus\!\setminus \hat{I}_{m,c}]\!]$ (cf. Figure 5.2). To put it differently, the global behavior of the interaction system that consists of only the management component satisfies the CTL*–X formula $\Phi = \mathsf{E}(\neg\{reserve_m\}\ \mathsf{U}\ \{deliver_m\})$, i.e., $[\![Sys_{\text{MMS}}[\{m\}] \setminus\!\setminus \hat{I}_{m,c}]\!] \models \Phi$, but the other system that consists of the management and the customer component does not, i.e., $[\![Sys_{\text{MMS}}[\{m,c\}] \setminus\!\setminus \hat{I}_{m,c}]\!] \not\models \Phi$. Note that we found a CTL*–X formula that separates the systems and according to De Nicola and Vaandrager [85], they thus cannot be divergence sensitive branching bisimilar which implies that they are also not branching bisimilar with explicit divergence (cf. Section 2.4), i.e., we have

$$Sys_{\text{MMS}}[\{m,c\}] \setminus\!\setminus \hat{I}_{m,c} \not\approx_b^{\Delta} Sys_{\text{MMS}}[\{m\}] \setminus\!\setminus \hat{I}_{m,c}.$$

Thus for our running example, the first idea does not yield a reduction of the system. But, let us examine why the idea failed. We learned above that the customer component influences the observable behavior of the management

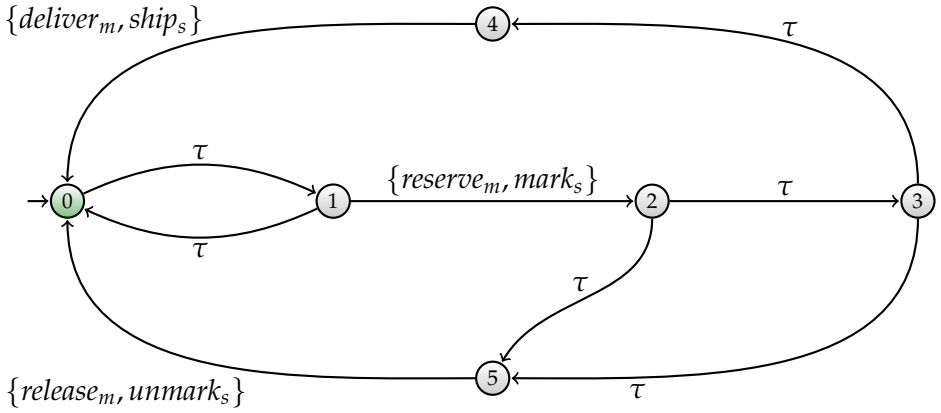


Figure 5.3: Global behavior of interaction system $Sys_{\text{MMS}}[\{m,c,s\}] \setminus\!\setminus \hat{I}_{m,c}$

component (in their joint behavior) such that it is unable to offer its deliver action before offering its reserve action. Interestingly, this restriction is not a restriction at all when we take a look at the cooperation partner for these actions: the storage component indeed requires that its mark action is used before it offers its ship action (cf. Figure 2.4 (b)). Since the corresponding interactions are exactly $\{deliver_m, ship_s\}$ and $\{reserve_m, mark_s\}$, we can try to further exploit this situation by checking whether the customer component has an influence on the cooperation of the management component and the storage component.

Here, we combine the above constructed systems $Sys_{\text{MMS}}[\{m,c\}] \setminus\!\setminus \hat{I}_{m,c}$ and $Sys_{\text{MMS}}[\{m\}] \setminus\!\setminus \hat{I}_{m,c}$ with the storage component via our composition operator and re-check their equivalence, i.e., we check whether

$$Sys_{\text{MMS}}[\{m,c\}] \setminus\!\setminus \hat{I}_{m,c} \underset{\mathcal{I}_{m,s}}{\otimes} Sys_{\text{MMS}}[\{s\}] \overset{?}{\approx}^{\Delta}_{b} Sys_{\text{MMS}}[\{m\}] \setminus\!\setminus \hat{I}_{m,c} \underset{\mathcal{I}_{m,s}}{\otimes} Sys_{\text{MMS}}[\{s\}]$$

holds where $\mathcal{I}_{m,s} = (I^{+}_{m,s}, I^{-}_{m,s})$ is the composition information between the management and the storage component with

$$\begin{aligned} I^{+}_{m,s} &:= \{\alpha \in Int[\{m,s\}] \mid \{m,s\} \subseteq \text{compset}(\alpha)\} \\ &= \{\{deliver_m, ship_s\}, \{release_m, unmark_s\}, \{reserve_m, mark_s\}\} \text{ and} \\ I^{-}_{m,s} &:= \{\alpha \in Int[\{m\}] \cup Int[\{s\}] \mid \alpha \notin Int[\{m,s\}]\} \\ &= \{\{deliver_m\}, \{mark_s\}, \{release_m\}, \{reserve_m\}, \{ship_s\}, \{unmark_s\}\}. \end{aligned}$$

Observe that the sets $I^{+}_{m,s}$ and $I^{-}_{m,s}$ are defined as the sets $I^{+}_{C_1,C_2}$ and $I^{-}_{C_1,C_2}$ respectively for $C_1 = \{m\}$ and $C_2 = \{s\}$ of Proposition 3.21. We can thus apply this proposition—since the sets of closed interactions of the two systems

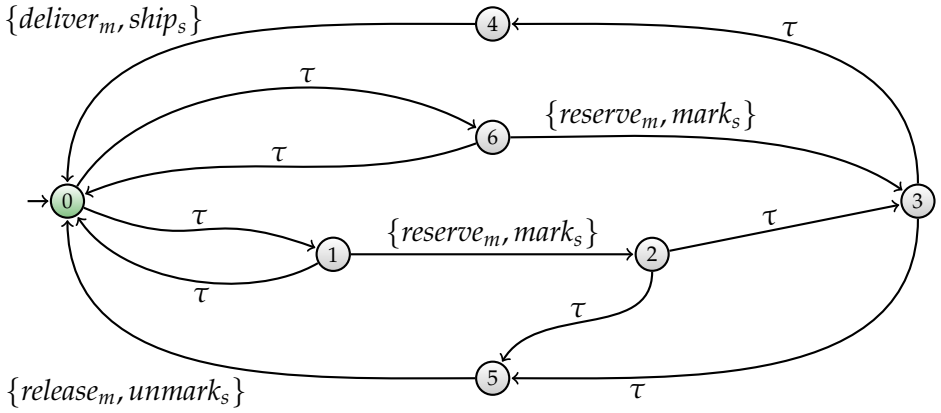


Figure 5.4: Global behavior of interaction system $Sys_{\text{MMS}}[\{m,s\}] \lVert \hat{I}_{m,c}$

are equal—and check whether the equivalence

$$Sys_{\text{MMS}}[\{m,c,s\}] \lVert \hat{I}_{m,c} \overset{?}{\approx}_b^{\Delta} Sys_{\text{MMS}}[\{m,s\}] \lVert \hat{I}_{m,c}$$

holds, which is equal to the check where we use the composition operator. The global behaviors of the two systems are depicted in Figure 5.3 and Figure 5.4 respectively. Observe that in the sixth state of $[\![Sys_{\text{MMS}}[\{m,s\}] \lVert \hat{I}_{m,c}]\!]$, the management component is unable to execute its deliver action since its cooperation partner, viz. the storage component, does not offer the corresponding ship action, i.e., the interaction $\{deliver_m, ship_s\}$ is not enabled in this state—which was the reason why our first idea failed.

Now, these systems are branching bisimilar with explicit divergence, which is established by the following equivalence relation (cf. Definition 2.15):

$$\mathcal{R} = \{(r^0,t^0),(r^1,t^1),(r^2,t^2),(r^3,t^3),(r^4,t^4),(r^5,t^5),(r^1,t^6)\}^{\leftrightarrow}$$

where r denotes the states of $[\![Sys_{\text{MMS}}[\{m,c,s\}] \lVert \hat{I}_{m,c}]\!]$ (cf. Figure 5.3) and t of $[\![Sys_{\text{MMS}}[\{m,s\}] \lVert \hat{I}_{m,c}]\!]$ (cf. Figure 5.4), i.e., we have

$$Sys_{\text{MMS}}[\{m,c,s\}] \lVert \hat{I}_{m,c} \approx_b^{\Delta} Sys_{\text{MMS}}[\{m,s\}] \lVert \hat{I}_{m,c}.$$

Note that this behavioral equivalence can be understood as a reduction of the system, since for any property regarding $Sys_{\text{MMS}} \lVert \hat{I}_{m,c}$ we can now answer the validity question by looking at $Sys_{\text{MMS}}[\{m,s\}] \lVert \hat{I}_{m,c}$ as long as the property is preserved by branching bisimilarity with explicit divergence, e.g., all CTL*–X formulae.

For instance, if we want to ensure that the management component does not execute its deliver action before its reserve action (as corresponding interac-

tions seen from an initial state), we can verify the CTL*–X property

$$\Psi = \mathsf{A}(\mathsf{F}\,\{deliver_m, ship_s\} \Rightarrow \neg\{deliver_m, ship_s\}\,\mathsf{U}\,\{reserve_m, mark_s\})$$

in the system $Sys_{\text{MMS}}[\{m,s\}] \setminus\!\setminus \hat{I}_{m,c}$. Since $[\![Sys_{\text{MMS}}[\{m,s\}] \setminus\!\setminus \hat{I}_{m,c}]\!] \models \Psi$ holds, we can conclude that also $[\![Sys_{\text{MMS}} \setminus\!\setminus \hat{I}_{m,c}]\!] \models \Psi$ holds.

Of course, we have to admit that by checking the equivalence that leads to the reduction, we computed the global behavior of our running example since $Sys_{\text{MMS}} = Sys_{\text{MMS}}[\{m,c,s\}]$ (cf. Proposition 3.20). As mentioned throughout this thesis, this computational should be avoided if we aim for efficient verification algorithms. But, as we elaborate in this chapter, we use this idea, viz. to check whether a border component influences the behavior of the middle component with respect to any other border component, to establish a compositional reduction technique that can be applied efficiently, i.e., we restrict the equivalence checks to systems consisting of at most three components.

Furthermore, we also have to admit that we silently assumed that for all actions of the middle component it holds that they are only used for cooperating with one border component, i.e., all actions are used exclusively with respect to one cooperation partner. This assumption is satisfied by our running example but before we generalize the approach sketched above, we define this special property, which we call *exclusive communication*, in the next section.

5.2 Exclusive Communication

As mentioned in the previous section, the partition of the set of actions of a component with respect to its cooperation partners is a feature that allows for compositional reduction, i.e., the actions of the middle component that are used for cooperation with a certain border component can only be abstracted if such an action is not used for cooperation with other border components. In other words, every action of every component is only contained in interactions where exactly the same set of components participates in.

Regarding interaction systems, such a property was introduced by Majster-Cederbaum and Martens [179] and called "exclusive communication". Their definition is dependent on the architecture of the system, viz. a tree-like one, but this dependence is not really needed and therefore, we re-define their notion in an architecture-independent way as follows.

Definition 5.1 (Exclusive Communication): Let *Sys* be an interaction system. *Sys* is said to have *exclusive communication* if for all $\alpha, \beta \in Int$ with $\alpha \cap \beta \neq \emptyset$ we have $\text{compset}(\alpha) = \text{compset}(\beta)$.

Note that Majster-Cederbaum and Martens [180] also changed their version in an architecture-independent variant, but strengthened the requirement such that every action of every component is only used in exactly one interaction. This variant is called "strongly exclusive communication" and defined as follows.

Definition 5.2 (Strongly Exclusive Communication): Let *Sys* be an interaction system. *Sys* is said to have *strongly exclusive communication* if for all $\alpha, \beta \in Int$ with $\alpha \neq \beta$ we have $\alpha \cap \beta = \emptyset$.

As we already mentioned in the previous section, the running example interaction system Sys_{MMS} has exclusive communication and since all actions occur in exactly one interaction, the system also has strongly exclusive communication. Next, we illustrate the difference between these two notions of exclusiveness by an example.

Example 5.3: Consider an interaction system *Sys* consisting of three components, i.e., $Comp = \{1,2,3\}$, each with an arbitrary local behavior, and the following action sets: $A_1 = \{a_1\}$, $A_2 = \{a_2, b_2\}$, and $A_3 = \{a_3\}$. We now consider six different interaction models for this system where the interaction sets are given in Table 5.1 and the set of closed interactions is empty in each case.

No.	Interaction Set *Int*	Excl. Comm.	Strongly Excl. Comm.
1	$\{\{a_1,a_2\},\{a_1,a_3\},\{b_2\}\}$	no	no
2	$\{\{a_1,a_2\},\{a_1\},\{b_2,a_3\}\}$	no	no
3	$\{\{a_1,a_2,a_3\},\{a_1,b_2,a_3\}\}$	yes	no
4	$\{\{a_1,a_2\},\{a_1,b_2\},\{a_3\}\}$	yes	no
5	$\{\{a_1,a_2\},\{b_2,a_3\}\}$	yes	yes
6	$\{\{a_1\},\{a_2\},\{b_2\},\{a_3\}\}$	yes	yes

Table 5.1: Demonstrating exclusive communication: The six different sets of interactions show different cooperation scenarios of the three components.

First, note that each interaction model is valid with respect to Definition 2.3. For each model, we have determined whether the respective system satisfies exclusive communication and/or the strong variant. If a system does not satisfy one of the notions, i.e., there is a "no" in the corresponding cell of the table, the first two interactions can be set as the interactions α and β of Definitions 5.1 and 5.2 respectively.

From Example 5.3, we already see that if we have strongly exclusive communication, we also have exclusive communication. Of course, this directly follows from the definition, but we nevertheless fix this observation as the following proposition.

Proposition 5.4: Let *Sys* be an interaction system. If *Sys* has strongly exclusive communication, then *Sys* also has exclusive communication.

A formal proof of Proposition 5.4 can be found in Appendix F on page 254.

Next, we deal with the issue of absence of exclusive communication.

5.2.1 Ensuring Exclusive Communication

Example 5.3 shows that interaction systems exist that do not satisfy exclusive communication initially. It is thus an interesting question how we can ensure this property for arbitrary interaction systems. Majster-Cederbaum and Martens [179, 180] illustrate such techniques for their variants of exclusive and strongly exclusive communication.

Since the variant of (non-strongly) exclusive communication is defined in an architecture dependent way, i.e., Definition 5.1 does not occur in the literature, we here elaborate on the technique to ensure exclusive communication as required by Definition 5.1.

The main idea to ensure exclusive communication of an interaction system is to replace all actions in a certain interaction by actions that are superscripted with the set of components that participate in the interaction (similar to the technique presented by Majster-Cederbaum and Martens [179]). Additionally, all transitions where those actions occur must be modified in the local behavior of the components. In the following, we refer to such an algorithm as EXCLUSIVE(Sys) for a given interaction system *Sys*. A pseudocode implementation can be found in Appendix B as Algorithm B.8 which shows that we can ensure the property in polynomial time.

Next, we show that the result of the algorithm is indeed an interaction system with exclusive communication.

Lemma 5.5: Let *Sys* be an interaction system. Applying Algorithm B.8 yields an interaction system EXCLUSIVE(Sys) that has exclusive communication.

A formal proof of Lemma 5.5 can be found in Appendix F on page 254.

For the same reasons as the transformation presented in Section 4.4.1, the

transformation induced by an application of Algorithm B.8 is only useful if properties of the original system are preserved. The following lemma deals with this issue.

Lemma 5.6: Let *Sys* be an interaction system. Applying Algorithm B.8 yields an interaction system EXCLUSIVE(*Sys*) such that the global behavior $[\![Sys]\!]$ is isomorphic up to transition relabeling to $[\![\text{EXCLUSIVE}(Sys)]\!]$.

A formal proof of Lemma 5.6 can be found in Appendix F on page 255.

For strongly exclusive communication, a similar idea can be used to ensure this property: Instead of superscripting each action with the set of components, we simply use an unique interaction identifier, e.g., each action $a \in \alpha$ becomes a^{α} in line 10 of Algorithm B.8. This idea was introduced by Majster-Cederbaum and Martens [180] where a detailed proof and a lemma that can be used to derive a similar isomorphism result as Lemma 5.6 was later given by Martens [190, Lemma 3.2.1]. Note that Majster-Cederbaum and Martens [179, 180] and Martens [190] derive only results for reachability of certain states and for deadlock-freedom, but our proof of isomorphism can be applied with a few adjustments in all cases as well (in fact, the authors use a similar lemma in all cases but did not address property preserving isomorphisms).

Thus, we can now assume that an interaction system has exclusive communication without loss of generality since otherwise, we can apply Algorithm B.8 to ensure this property in polynomial time in the size of the input.

5.3 Exploiting Equivalences in Interaction Systems

In this section, we want to generalize the idea for reducing interaction system as introduced in Section 5.1. We start with the situation as given by our running example: An interaction system with a star-like architecture which has exclusive communication. Note that this situation was researched on in our work [160]. Afterwards, we take a look at other architectures.

5.3.1 Star-Like Architectures with Exclusive Communication

We summarize the idea behind a generalization of the reduction approach presented in Section 5.1: Consider an interaction system with a star-like architecture, i.e., a middle component m is surrounded by border components $b_1, \ldots, b_n$ for an $n \in \mathbb{N}$, that has exclusive communication. If border component b_1 does influence the behavior of m but this influence is not important

for border component b_2, i.e., b_2 serves as a witness of the harmlessness of this influence with respect to the global behavior of the system, then we can safely ignore b_1 if we abstract from the cooperation of m and b_1 (which does not affect any other component because we have exclusive communication). In the same vein, if now border component b_2 influences m but again we find a witness, say b_3, we can further reduce the system by ignoring b_2 and abstracting from the cooperation between m and b_2. Now, if we can proceed this way, we finally arrive at the situation where only the middle component m and border component b_n are left and any cooperation between m and other border components is abstracted away. Thus, we can conclude that the behavior of the resulting system is equivalent to the global behavior of the original system (with the same abstractions applied). The important point is that we can derive this equivalence without computing the global behavior of the original system, and since we only computed the equivalence of small systems on the way, this approach is very efficient.

We formalize this idea in the following theorem. There, we introduce a special function that serves as an oracle for the order of the border components that we called $b_1, \ldots, b_n$ above. After the proof of the theorem and a further example, we discuss how such an order can be obtained. But for the theorem, we assume that it is already known. We also give the proof at this point (and do not refer to an appendix) because it illustrates an application of the results from Chapter 3.

Theorem 5.7: Let *Sys* be an interaction system with a star-like architecture, exclusive communication, and at least three components, i.e., $|Comp| \geq 3$. Let an arbitrary numbering of the components be given, i.e., a bijective function $f\colon Comp \to \{1, \ldots, n\}$ is given with $n = |Comp|$. Let number 1 denote the middle component, viz. $f^{-1}(1)$, of the star-like architecture. For convenience, we identify a component with respect to its unique number in the following. If for all i with $2 \leq i < n$ it holds that

$$Sys[\{1, i, i+1\}] \setminus\setminus \hat{I}_{1,i} \approx_b^{\Delta} Sys[\{1, i+1\}] \setminus\setminus \hat{I}_{1,i}$$

with $\hat{I}_{1,i} := \big(Int[\{1,i\}] \setminus Int[\{1\}]\big) \cup \big\{\alpha \in Int[\{1\}] \mid \exists\, \beta \in Int[\{1,i\}]\colon \alpha \subset \beta\big\}$,

$$Sys \setminus\setminus \hat{I} \approx_b^{\Delta} Sys[\{1, n\}] \setminus\setminus \hat{I}$$

holds with $\hat{I} := \bigcup_{i \in \{2,\ldots,n-1\}} \hat{I}_{1,i}$.

Proof of Theorem 5.7: Assume that the premise of the theorem holds. We show $Sys \setminus\setminus \hat{I} = Sys[\{1, \ldots, n\}] \setminus\setminus \hat{I} \approx_b^{\Delta} Sys[\{1, n\}] \setminus\setminus \hat{I}$ by an inductive argument over the components. The first equality follows from Proposition 3.20. For $n = 3$ holds $Sys[\{1,2,3\}] \setminus\setminus \hat{I}_{1,2} \approx_b^{\Delta} Sys[\{1,3\}] \setminus\setminus \hat{I}_{1,2}$—because of the premise—and since $\hat{I}_{1,2} = \hat{I}$ for $n = 3$ and branching bisimilarity with explicit

divergence is a congruence with respect to closing (cf. Proposition 3.17), the claim follows. Now, for all components i with $2 \leq i < n$ holds:

$$Sys[\{1, i, \ldots, n\}] \setminus\setminus \hat{I} \approx_b^{\Delta} Sys[\{1, i+1, \ldots, n\}] \setminus\setminus \hat{I}$$

because (the used proposition is listed in parentheses at the end of each line):

$$\begin{aligned}
& Sys[\{1, i, \ldots, n\}] \setminus\setminus \hat{I} \\
&= \Big(Sys[\{1, i, i+1\}] \underset{\mathcal{I}_{\{1,i,i+1\},\{i+2,\ldots,n\}}}{\otimes} Sys[\{i+2, \ldots, n\}]\Big) \setminus\setminus \underbrace{\hat{I} \cup Int_{\text{closed}}}_{\hat{I}'} && (3.21) \\
&= \Big(Sys[\{1, i, i+1\}] \underset{\mathcal{I}_{\{1,i+1\},\{i+2,\ldots,n\}}}{\otimes} Sys[\{i+2, \ldots, n\}]\Big) \setminus\setminus \hat{I}' && (3.23) \\
&= \Big(Sys[\{1, i, i+1\}] \setminus\setminus \hat{I}_{1,i} \underset{\mathcal{I}_{\{1,i+1\},\{i+2,\ldots,n\}}}{\otimes} Sys[\{i+2, \ldots, n\}]\Big) \setminus\setminus \hat{I}' && (3.15) \\
&\approx_b^{\Delta} \Big(Sys[\{1, i+1\}] \setminus\setminus \hat{I}_{1,i} \underset{\mathcal{I}_{\{1,i+1\},\{i+2,\ldots,n\}}}{\otimes} Sys[\{i+2, \ldots, n\}]\Big) \setminus\setminus \hat{I}' && \text{(premise)} \\
&= \Big(Sys[\{1, i+1\}] \underset{\mathcal{I}_{\{1,i+1\},\{i+2,\ldots,n\}}}{\otimes} Sys[\{i+2, \ldots, n\}]\Big) \setminus\setminus \hat{I} \cup Int_{\text{closed}} && (3.15) \\
&= Sys[\{1, i+1, \ldots, n\}] \setminus\setminus \hat{I} && (3.21)
\end{aligned}$$

Thus, the argument shows that the theorem holds for all components. Note that when we applied the premise in the reasoning above, we also used Propositions 3.12 and 3.17. In the last step, we can drop the set Int_{closed} from the closing operator because all interactions of $Int[\{1\}]$ that are a subset of a closed interaction of the whole system Sys are also contained in $\hat{I}$ or $Int_{\text{closed}}[\{1, i+1, \ldots, n\}]$ respectively. This is ensured by the exclusive communication of the system. ■

We illustrate the application of Theorem 5.7 by means of the following example.

Example 5.8: We consider a traffic light system. The system consists of three bulb controllers which can be turned on and off and which relay this request to connected electric bulbs or LED grids in the colors red, yellow, and green. The bulb controllers are operated by a traffic light controller that organizes the (four-state) order of light combinations or traffic signals, i.e., signal red for stopping of the traffic, then red and yellow for informing the drivers to start their engines, then green for driving, and then yellow for signaling that the drivers have to stop shortly. Here, a clock is used to initiate the change of each signal, i.e., the clock sends a trigger after waiting a sequence of internal ticks. Furthermore, the traffic light controller is designed to output the current traffic signal and to ensure that when the system is switched on, the red signal is the first one and all other bulb controllers are turned off.

We model this description as an interaction system *Sys* where each controller corresponds to a component of the system. We omit the modeling of electric bulbs or LED grids and only model the controllers which we simply call bulbs in the following. Thus, we have:

$Comp = \{l, clk, 1, 2, 3\}$ —for traffic **l**ight, **c**lo**ck**, and bulbs **1**, **2**, and **3**,

$A_l = \{signal\ red_l, signal\ redyellow_l, signal\ green_l, signal\ yellow_l, wait_l,$
$red\ on_l, red\ off_l, yellow\ on_l, yellow\ off_l, green\ on_l, green\ off_l\},$

$A_{clk} = \{tick_{clk}, trigger_{clk}\}$, and

$A_i = \{on_i, off_i, poweron_i, poweroff_i\}$ for $1 \leq i \leq 3$.

Next, we have to specify the interaction model, which is illustrated in Figure 5.5: Bulb 1 corresponds to the red one, 2 to the yellow, and 3 to the green.

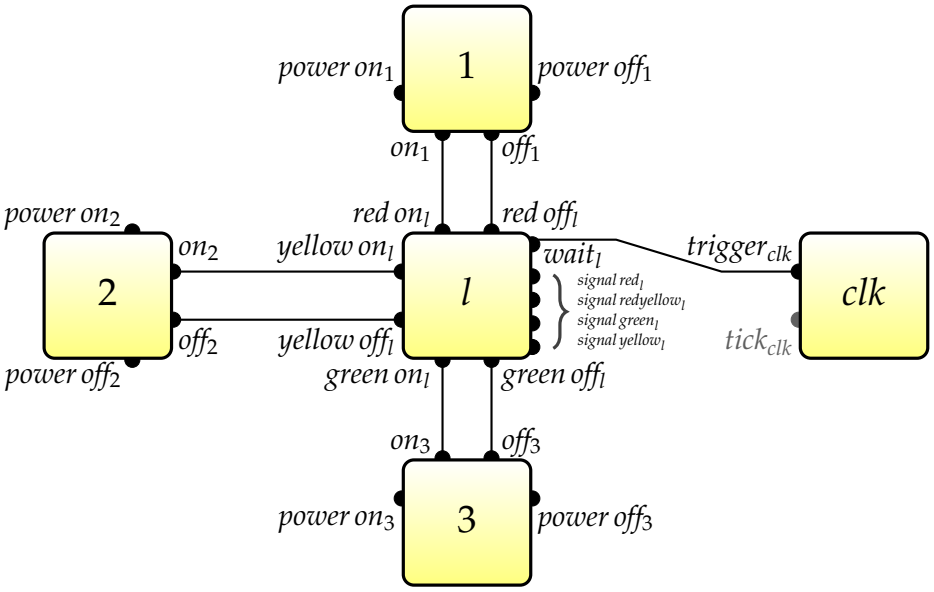


Figure 5.5: Interaction model of the traffic light example

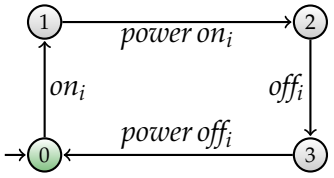
Thus, we have the following interactions:

$Int = \big\{\{signal\ red_l\}, \{signal\ redyellow_l\}, \{signal\ green_l\}, \{signal\ yellow_l\},$
$\{wait_l, trigger_{clk}\}, \{red\ on_l, on_1\}, \{red\ off_l, off_1\}, \{yellow\ on_l, on_2\},$
$\{yellow\ off_l, off_2\}, \{green\ on_l, on_3\}, \{green\ off_l, off_3\}, \{tick_{clk}\}\big\}$
$\cup \big\{\{poweron_i\}, \{poweroff_i\} \mid 1 \leq i \leq 3\big\}$ and

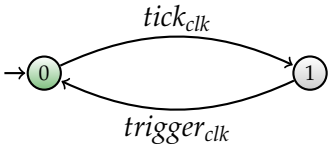
$Int_{\text{closed}} = \big\{\{tick_{clk}\}\big\}.$

Please note that we model the ticks of the clock as closed interactions since these are usually internal.

The behavior of the bulb components is given by the labeled transition system depicted in Figure 5.6 (a) and the behavior of the clock component by the one in Figure 5.6 (b). Here, we simplify the clock component such that it only waits for one tick before offering the next trigger.



(a) Behavior of the bulb comp. i, $1 \leq i \leq 3$



(b) Behavior of the clock component *clk*

Figure 5.6: Behavior of the bulb (a) and the clock component (b)

More complex is the behavior of the traffic light component which is depicted in Figure 5.7 on the following page. Every horizontal layer of the labeled transition system corresponds to a sequence of operations that enable the next traffic signal, which is output by the connection of theses layers (the signal actions). The system is able to turn the green signal off initially (and also other non-initial signals) as requested by the specification.

Observe that the interaction system is now completely specified (with respect to Definition 2.5).

We now apply Theorem 5.7 to this interaction system, i.e., we want to verify properties such as deadlock-freedom of the system but avoid to construct the global behavior. First, we need to check the assumptions of the theorem. Observe that *Sys* has exclusive communication, which can easily be concluded from the interaction model depicted in Figure 5.5 because no action is used in interactions with different sets of participating components. We also have $|Comp| \geq 3$, i.e., at least three components are part of *Sys*. Next, we have to check the architecture of the interaction system. We compute the component graph (cf. Definition 4.1) which is depicted in Figure 5.8 on the following page.

Since it is a star in the graph-theoretical sense, we conclude that *Sys* has a star-like architecture (cf. Definition 4.2) where component l is the middle component and components 1, 2, 3, and *clk* are the border components. Thus, the assumptions of the theorem are satisfied and we can now take a look at the equivalences.

We assume that the following function $f\colon Comp \to \{1,2,3,4,5\}$ is given: $f(l) = 1$, $f(3) = 2$, $f(2) = 3$, $f(1) = 4$, and $f(clk) = 5$.

We check for all i with $2 \leq i < 5$ whether (where $\hat{I}'_i = \hat{I}_{f^{-1}(1),f^{-1}(i)}$)

$$Sys[\{f^{-1}(1), f^{-1}(i), f^{-1}(i+1)\}] \setminus\setminus \hat{I}'_i \approx^{\Delta}_{b} Sys[\{f^{-1}(1), f^{-1}(i+1)\}] \setminus\setminus \hat{I}'_i$$

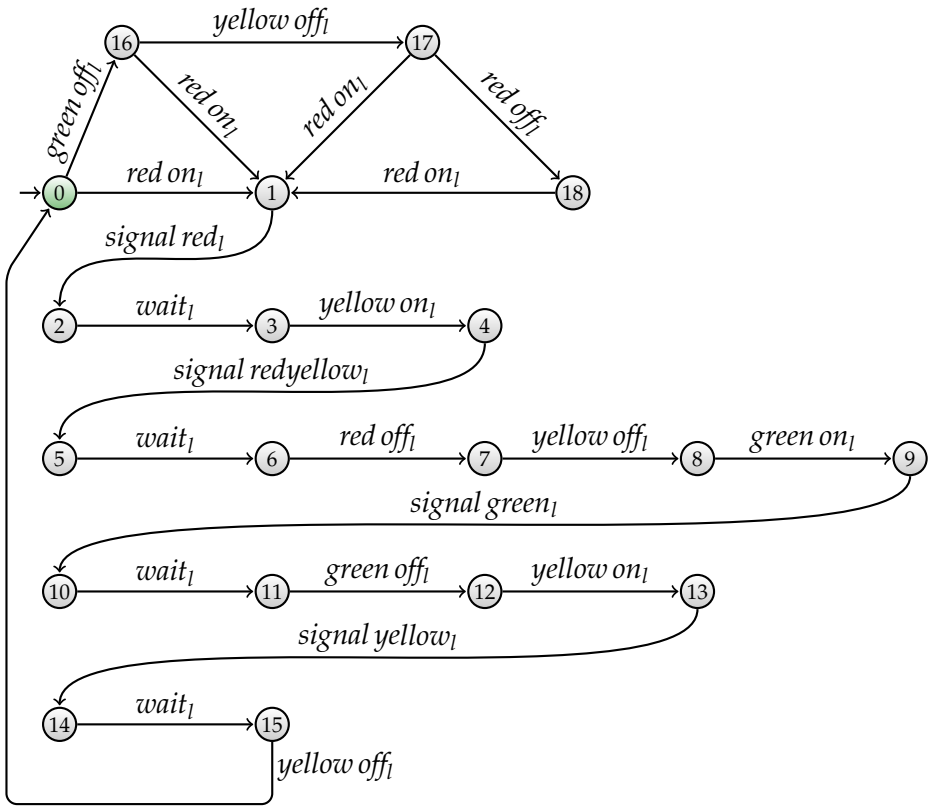


Figure 5.7: Behavior of the traffic light component l

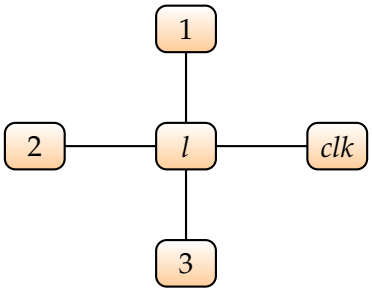


Figure 5.8: Component graph of the traffic light example

holds where the parameter of the closing operator is defined as in Theorem 5.7. Here, we check whether the light component together with the green bulb component influences the observable behavior of the light and yellow bulb components, then wether this is the case for the light and yellow bulb components with respect to the red bulb component, and finally wether the light and red bulb component influence the observable behavior of the light and clock component. We omit to depict the single systems and to give the equivalence relations. But, all equivalences specified in the equation above hold.

We can thus conclude that

$$Sys \setminus\setminus \hat{I} \approx_b^{\Delta} Sys[\{l, clk\}] \setminus\setminus \hat{I}$$

holds where $\hat{I}$, as defined in the theorem, closes all interactions between the omitted components and the light component.

We want to determine the savings that this reduction allows for. The reachable global behavior of *Sys* consists of 256 states and 672 transitions. If we now compute the behavior of $Sys[\{l, clk\}] \setminus\setminus \hat{I}$, we see that this system only consists of 38 states and 59 transitions, i.e., the system is much smaller. However, we can further reduce the system. We compute the quotient of the labeled transition system representing the reachable global behavior, i.e., we have:

$$[\![Sys[\{l, clk\}] \setminus\setminus \hat{I}]\!] \approx_b^{\Delta} [\![Sys[\{l, clk\}] \setminus\setminus \hat{I}]\!]_{\approx_b^{\Delta}}.$$

The resulting system consists of only 8 states and 8 transitions and is depicted in Figure 5.9.

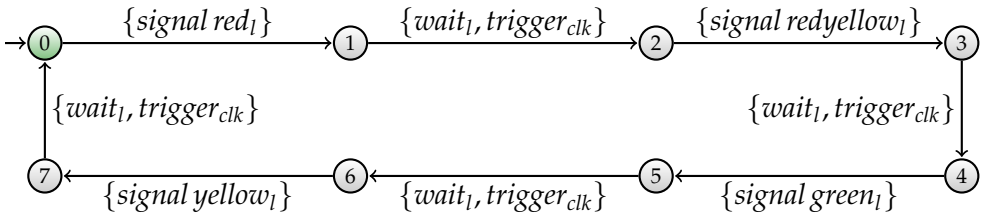


Figure 5.9: The labeled transition system $[\![Sys[\{l, clk\}] \setminus\setminus \hat{I}]\!]_{\approx_b^{\Delta}}$

Since the labeled transition system $[\![Sys[\{l, clk\}] \setminus\setminus \hat{I}]\!]_{\approx_b^{\Delta}}$ contains no state without outgoing transitions, we can conclude that the interaction system $Sys \setminus\setminus \hat{I}$ is deadlock-free and also that *Sys* is deadlock-free since deadlocks are preserved by closing.

This ends the discussion of the example.

An important assumption of Theorem 5.7 is that the components can be ordered as stated in the lemma. Since such an order is not automatically given, we overcome this problem by demanding that certain equivalences hold for all border components of the star-like architecture, i.e., an order of the components is not needed previously. We formalize this idea in the following corollary.

Corollary 5.9: Let *Sys* be an interaction system with a star-like architecture and exclusive communication. Let m denote the middle component and k an arbitrary border component. If for all distinct pairs of border components

$i, j \in Comp \setminus \{m\}$ holds

$$Sys[\{m,i,j\}] \setminus\setminus \hat{I}_{m,i} \approx_b^{\Delta} Sys[\{m,j\}] \setminus\setminus \hat{I}_{m,i}$$

where $\hat{I}_{m,i} := \big(Int[\{m,i\}] \setminus Int[\{m\}]\big) \cup \big\{\alpha \in Int[\{m\}] \mid \exists\, \beta \in Int[\{m,i\}] \colon \alpha \subset \beta\big\}$, then it holds with $\hat{I} := \bigcup_{i \in Comp \setminus \{m,k\}} \hat{I}_{m,i}$ that

$$Sys \setminus\setminus \hat{I} \approx_b^{\Delta} Sys[\{m,k\}] \setminus\setminus \hat{I}.$$

A formal proof of Corollary 5.9 can be found in Appendix F on page 256.

We repeat our intuitive argument for this corollary: The composition of an arbitrary border component i with the middle component m does not interfere the composition of m with any other border component j. This compositional invariant exploits the branching bisimilarity (with explicit divergence) of the composed subsystems, i.e., the non-interference is modeled by equivalent behavior. If this invariant holds for all pairs of components i, j and component m, the behavior of the whole system is already modeled (up to branching bisimilarity with explicit divergence) by the middle component composed with one of the border components.

Here, we do not give a detailed algorithm that implements the approach of Corollary 5.9 because we just have to check whether the architecture is star-like and determine the middle and border components (cf. Section 4.3.1). Afterwards, we need to loop through the set of border components and build the respective subsystems (cf. Definition 3.18) and check their branching bisimilarity with explicit divergence (cf. Section E.5). The sets $\hat{I}_{m,i}$ of closed interactions can clearly be computed in linear time by computing the set $Int[\{m,i\}] \setminus Int[\{m\}]$ and looping through the sets $Int[\{m\}]$ and $Int[\{m,i\}]$ in each case which are already available from the subsystem construction.

However, we are interested in the computational costs of the application of Corollary 5.9. With the algorithm of Groote and Vaandrager [130], we can establish whether two labeled transition systems are branching bisimilar (with explicit divergence) in time proportional to the product of the number of states and the number of transitions in both systems (cf. Appendix E). If $S_{\max}$ denotes the largest local state space, i.e., $|S_{\max}| = \max\{|S_i| \mid i \in Comp\}$, we know that the subsystems can be determined in $O(|S_{\max}|^3 \cdot |Int|)$ time. The number of states of the resulting global behaviors of these subsystems is bounded by $O(|S_{\max}|^3)$ whereas the size of the largest possible family of transition relations is bounded by $O(|S_{\max}|^3 \cdot |Int| \cdot |S_{\max}|^3)$. Thus, each computation of the subsystems and the branching bisimilarity (with explicit divergence) check can be carried out in time polynomial in the size of the input interaction system.

Now, we carry out this computation at most $(|Comp| - 1) \cdot (|Comp| - 2)$ times (each border component together with each other border component) which results in an overall polynomial time bound of the approach of Corollary 5.9.

A further interesting question is whether Corollary 5.9 and Theorem 5.7 are equivalent. Clearly, if the assumptions of the corollary hold, then also the theorem holds (cf. the proof of Corollary 5.9). However, is it the case that if we can find an order function f such that the equivalences of Theorem 5.7 hold, then also the corollary holds? In other words, the order in which the components are treated is not relevant for the reduction approach.

Unfortunately, this is not the case as can be seen by Example 5.8. In the example, we assumed that a particular order function f is given, and indeed, the equivalences of Theorem 5.7 hold with this function. But for Corollary 5.9, we need to check (among other subsystems) whether

$$Sys[\{l, 2, clk\}] \setminus\setminus \hat{I}_{l,2} \overset{?}{\approx}_b^{\Delta} Sys[\{l, clk\}] \setminus\setminus \hat{I}_{l,2}$$

holds where the parameter of the closing operator is defined as in Theorem 5.7 and Corollary 5.9 respectively. Here, we need to check whether the light component together with the yellow bulb component influences the observable behavior of the light and clock component where the interactions of the light and yellow bulb component are closed in each system. We do not depict the corresponding labeled transition systems here, but observe that in the system of the right-hand side, we can execute the interaction sequence $\{green\ off_l\}$, τ, $\{red\ off_l\}$ which directly corresponds to the action sequence $green\ off_l$, $yellow\ off_l$, $red\ off_l$ of the light component's behavior (cf. Figure 5.7). However, in the system of the left-hand side this interaction sequence is not executable because the yellow bulb component does not offer its off_2-action before its on_2-action (cf. Figure 5.6 (a)), i.e., the interaction $\{yellow\ off_l, off_2\}$ is not executable (as a closed interaction) after the $\{green\ off_l\}$-interaction in the system of the left-hand side. Thus, the systems cannot be branching bisimilar with explicit divergence and the assumptions of Corollary 5.9 are not satisfied. More precisely, we established that the systems are not trace equivalent [23, Section 3.2.4] which is a coarser equivalence than branching bisimilarity with explicit divergence [113, Section 2].

Summarizing, we conclude that Theorem 5.7 implies Corollary 5.9 but not the other way round. We finish with a remark about finding an order function f as required for Theorem 5.7.

Remark on Finding an Order Function f: Since we already compute all possible equivalences in Corollary 5.9, we only need to exploit this information accordingly. We define a (directed, bipartite) graph $G = (V, E)$ with $V = V_1 \cup V_2$

and $V_1 \cap V_2 = \emptyset$ where the set V_1 of vertices consists of the tuples (m, i, j) and the set V_2 of the tuples (m, i) where m is the middle component and i, j range over all distinct pairs of border components, i.e., $i \neq j$ except m. We introduce a directed edge between all tuples that share the first two components, i.e., $\big((m, i), (m, i, j)\big) \in E$ for all $i, j \in \mathit{Comp} \setminus \{m\}$ with $i \neq j$. Next, we consider the equivalences computed in the corollary: For all distinct pairs of border components $i, j \in \mathit{Comp} \setminus \{m\}$ if $\mathit{Sys}[\{m, i, j\}] \setminus\!\setminus \hat{I}_{m,i} \approx_b^{\Delta} \mathit{Sys}[\{m, j\}] \setminus\!\setminus \hat{I}_{m,i}$ holds where $\hat{I}_{m,i}$ is defined as in the corollary, we add the directed edge $\big((m, i, j), (m, j)\big)$ to E. Now, if there is a simple path that contains all vertices in V_2, starts at an arbitrary vertex $v \in V_2$, and ends at another vertex $w \in V_2$, we found an order function f for Theorem 5.7. Figure 5.10 illustrates this idea with respect to the traffic light interaction system introduced in Example 5.8 where the four vertices in the middle belong to the set V_2 and the others to V_1.

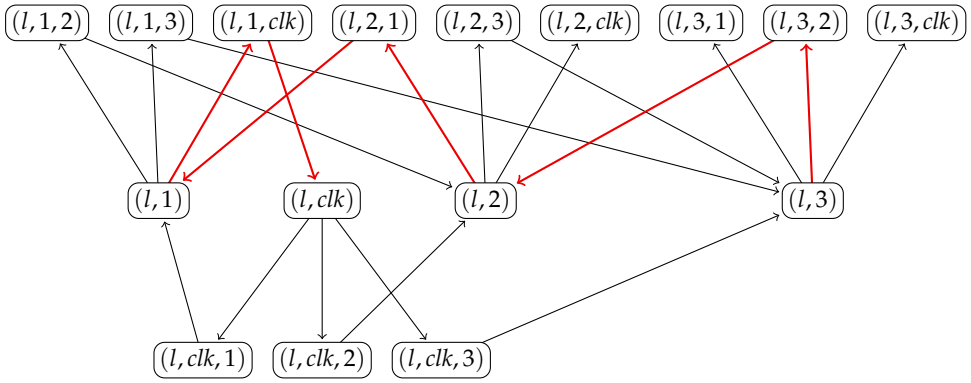


Figure 5.10: Graph that illustrates how an order function f for Theorem 5.7 can be found. The thick red edges constitute a simple path that, projected to V_2, corresponds to the order given by function f in Example 5.8.

We want to point out that this idea can be further extended such that we can ask for a particular border component k whether the whole interaction system can be reduced to a system that consists only of the middle component m and k. To accomplish this, we only need to construct the graph as above, turn the direction of all edges, and search for a simple path starting in the vertex (m, k) that contains all vertices (m, i) with $i \in \mathit{Comp} \setminus \{m\}$. Note that the construction of the graph above and all path queries can be performed in polynomial time in the size of the input where the construction has the same asymptotic costs as the application of Corollary 5.9 which we discussed before this remark. Here, we do not further discuss the order function f but come back to it with some further ideas in Section 5.6 at the end of this chapter.

Next, we take a look at our other architectural constraint that is based on the component graph, viz. interaction systems with a tree-like architecture.

5.3.2 Tree-Like Architectures with Exclusive Communication

In this section, we take a look at interaction systems with a tree-like architecture and exclusive communication. Fortunately, the presented analysis for star-like architectures scales up to an analysis for tree-like architectures. The main idea for the analysis of tree-like architectures is that they can be considered as several overlapping star-like ones. We illustrate this idea in Figure 5.11 which depicts the component graph of an interaction system *Sys* with a tree-like architecture.

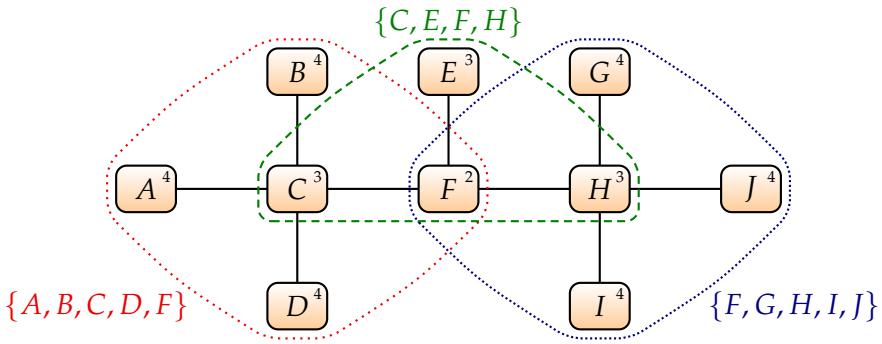


Figure 5.11: In each vertex, the small number in the top right corner corresponds to the eccentricity of the vertex. We can now group the vertices with respect to these eccentricities such that each induced subgraph is a star in the graph-theoretical sense which is maximal with respect to the set of vertices that form a star subgraph. Each such star in the graph is surrounded by either a red dotted, green dashed, or blue densely dotted line.

We motivate the idea by means of the depicted example where we have $Comp = \{A, B, C, D, E, F, G, H, I, J\}$. We do not further specify this interaction system but it should be clear that we can define appropriate action sets and interactions such that the system has the component graph depicted in Figure 5.11. Now, we determine the maximal stars of the tree as mentioned in the caption of the figure and thus consider the following interaction systems: $Sys[\{A, B, C, D, F\}]$, $Sys[\{C, E, F, H\}]$, and $Sys[\{F, G, H, I, J\}]$. All three subsystems have a star-like architecture where components C, F, and H respectively are the middle components.

If now our result of the previous section is applicable for these subsystems, i.e., we assume that we can conclude that

1. $Sys[\{A, B, C, D, F\}] \setminus\setminus \hat{I}_1 \approx_b^{\Delta} Sys[\{C, F\}] \setminus\setminus \hat{I}_1$,
2. $Sys[\{C, E, F, H\}] \setminus\setminus \hat{I}_2 \approx_b^{\Delta} Sys[\{F, E\}] \setminus\setminus \hat{I}_2$, and
3. $Sys[\{F, G, H, I, J\}] \setminus\setminus \hat{I}_3 \approx_b^{\Delta} Sys[\{F, H\}] \setminus\setminus \hat{I}_3$

holds (cf. Corollary 5.9), then we can put these three pieces together and conclude that also $Sys \setminus\setminus \hat{I} \approx_b^{\Delta} Sys[\{F, E\}] \setminus\setminus \hat{I}$ holds with appropriate sets of interactions $\hat{I}_1$, $\hat{I}_2$, $\hat{I}_3$, and $\hat{I}$. Of course, we have to be careful which interactions are contained in these sets such that no interaction becomes closed that is needed for a further application of Corollary 5.9. Moreover, we should work from the outer levels, i.e., the vertices with the maximal eccentricity which correspond to the periphery of the graph (cf. Definition A.7), to the inner levels as depicted in Figure 5.11. Then, i.e., if all equivalences hold, the whole interaction system is equivalent to a system containing only the components in the graph-theoretical center of the component graph.

Next, we formalize the above developed idea for an arbitrary interaction system with a tree-like architecture and exclusive communication.

Theorem 5.10: Let Sys be an interaction system with a tree-like architecture, exclusive communication, and at least three components, i.e., $|Comp| \geq 3$. Let G denote the component graph of Sys, and let for each index $x \in \mathbb{N}$ with $\text{rad}(G) \leq x < \text{diam}(G)$ the set $\mathcal{C}_x^\star \subseteq 2^{Comp}$ denote the set of all subsets of the set of components such that for each $C_x^\star \in \mathcal{C}_x^\star$ holds $|C_x^\star| \geq 3$, $Sys[C_x^\star]$ has a star-like architecture, the vertex representing the middle component of $Sys[C_x^\star]$ has eccentricity x, and there is no component $k \in Comp \setminus C_x^\star$ such that $Sys[C_x^\star \cup \{k\}]$ has a star-like architecture, i.e., the set of components is maximal with respect to satisfaction of the star-like architectural constraint.

We introduce a (double) index $x.y$ with index x as above and index $y \in \mathbb{N}$ with $1 \leq y \leq |\mathcal{C}_x^\star|$ (for each x) such that $C_{x.y}^\star$ lets us refer to all sets of components in $\mathcal{C}_x^\star$, i.e., the two indices allow us to address each star-like subsystem. Let $m_{x.y} \in C_{x.y}^\star$ denote the middle component in each $Sys[C_{x.y}^\star]$, and let $b_{x.y} \in C_{x.y}^\star$ denote the border component whose vertex representation has the smallest eccentricity, i.e., $\text{ecc}_G(b_{x.y}) \leq \min\{\text{ecc}_G(v) \mid v \in V \cap C_{x.y}^\star\}$—if there are several such border components, we choose an arbitrary one. Furthermore, a bijective function $f_{x.y} \colon C_{x.y}^\star \to \{1_{x.y}, 2_{x.y}, \ldots, n_{x.y}\}$ with $n_{x.y} = |C_{x.y}^\star|$ is given where number $1_{x.y}$ denotes the middle component, i.e., $f_{x.y}^{-1}(1_{x.y}) = m_{x.y}$, and number $n_{x.y}$ denotes the border component with the smallest eccentricity, i.e., $f_{x.y}^{-1}(n_{x.y}) = b_{x.y}$. For convenience, we identify a component with respect to its number in the following.

If for all x with $\text{rad}(G) \leq x < \text{diam}(G)$, all $x.y$ with $1 \leq y \leq |\mathcal{C}_x^\star|$, and all $i_{x.y}$ with $2_{x.y} \leq i_{x.y} < n_{x.y}$ holds

$$Sys[\{1_{x.y}, i_{x.y}, i_{x.y}+1\}] \setminus\setminus \hat{I}_{1_{x.y}, i_{x.y}} \approx_b^{\Delta} Sys[\{1_{x.y}, i_{x.y}+1\}] \setminus\setminus \hat{I}_{1_{x.y}, i_{x.y}}$$

where we have $\hat{I}_{1_{x.y}, i_{x.y}} := \big(Int[\{1_{x.y}, i_{x.y}\}] \setminus Int[\{1_{x.y}\}]\big) \cup \big\{\alpha \in Int[\{1_{x.y}\}] \mid \exists\, \beta \in Int[\{1_{x.y}, i_{x.y}\}] \colon \alpha \subset \beta\big\}$, then it holds with $M := \text{center}(G) \cup K$ where

$K = \{n_{\mathrm{rad}(G).1}\}$ if $|\mathrm{center}(G)| = 1$ and $K = \emptyset$ otherwise, that

$$Sys \setminus\setminus \hat{I} \approx_b^{\Delta} Sys[M] \setminus\setminus \hat{I}$$

with $\hat{I} := \bigcup_{\mathrm{rad}(G) \leq x < \mathrm{diam}(G), 1 \leq y \leq |\mathcal{C}_x^*|, 2_{x.y} \leq i_{x.y} < n_{x.y}} \hat{I}_{1_{x.y}, i_{x.y}}$.

A formal proof of Theorem 5.10 can be found in Appendix F on page 256.

We proceed with an example that demonstrates the application of Theorem 5.10.

Example 5.11: We consider a street crossing with four traffic lights but model only two light controllers that show the opposing light signals, i.e., each controller operates two traffic lights that show the same signals but for convenience, we model each of them as one traffic light. Each light controller and the associated three bulb controllers are modeled as in Example 5.8. A further control unit coordinates the switching of the opposing light signals, e.g., if one traffic light shows its green signal, the other one has to show its red signal. Again, a clock is used to initiate the change of each signal, however now the clock sends its trigger to the control unit. Initially, the control unit, that we model as component C, operates the first light controller, that we model as component l, to show its green signal and the other light controller, that we model as component r, to show its red signal. Here, the light controller l is connected to three bulb controllers 1, 2, and 3 as in Example 5.8 whereas the light controller r is connected to three bulb controllers 4, 5, and 6.

Thus, we get the following component model:

$Comp = \{C, l, r, clk, 1, 2, 3, 4, 5, 6\}$,

$A_C = \{status\ green\ red_C, status\ yellow\ red_C, status\ red\ redyellow_C, status\ red\ green_C,$
$status\ red\ yellow_C, status\ redyellow\ red_C, wait_C, trig1_C, sig1_C, trig2_C, sig2_C\}$,

A_l, A_{clk}, and A_i for $1 \leq i \leq 6$ as in Example 5.8, and

A_r as A_l but subscripted with r instead of l.

We assume that bulb 1 corresponds to the red light of traffic light l, 2 to the yellow one of l, 3 to the green one of l, and similarly, 4, 5, and 6 to the red, yellow, and green light of traffic light r. Figure 5.12 on the following page depicts the interaction model of the street crossing example.

Thus, we have the following interactions:

$Int = \{\{status\ green\ red_C\}, \{status\ yellow\ red_C\}, \{status\ red\ redyellow_C\},$
$\{status\ red\ green_C\}, \{status\ red\ yellow_C\}, \{status\ redyellow\ red_C\},$
$\{wait_C, trigger_{clk}\}, \{signal\ red_l, sig1_C\}, \{signal\ redyellow_l, sig1_C\},$

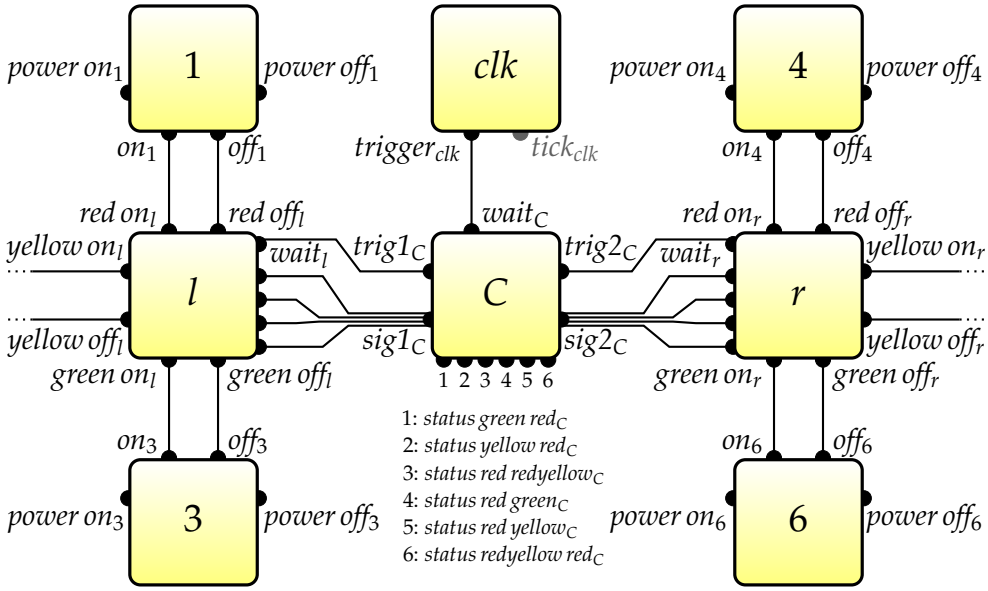


Figure 5.12: Interaction model of the traffic light street crossing example where we omitted the bulb components 2 and 5.

$$\begin{aligned}
&\{signal\,green_l, sig1_C\}, \{signal\,yellow_l, sig1_C\}, \{wait_l, trig1_C\},\\
&\{signal\,red_r, sig2_C\}, \{signal\,redyellow_r, sig2_C\}, \{signal\,green_r, sig2_C\},\\
&\{signal\,yellow_r, sig2_C\}, \{wait_r, trig2_C\}, \{red\,on_l, on_1\}, \{red\,off_l, off_1\},\\
&\{yellow\,on_l, on_2\}, \{yellow\,off_l, off_2\}, \{green\,on_l, on_3\}, \{green\,off_l, off_3\},\\
&\{red\,on_r, on_4\}, \{red\,off_r, off_4\}, \{yellow\,on_r, on_5\}, \{yellow\,off_r, off_5\},\\
&\{green\,on_r, on_6\}, \{green\,off_r, off_6\}, \{tick_{clk}\}\}\\
\cup\,&\{\{poweron_i\}, \{poweroff_i\} \mid 1 \leq i \leq 6\} \text{ and}
\end{aligned}$$

$$Int_{\text{closed}} = \{\{tick_{clk}\}\}.$$

The behavior of the components is similar as in Example 5.8 where the behavior $[\![r]\!]$ of the light controller r is the same as the behavior $[\![l]\!]$ of the light controller l (cf. Figure 5.7 on page 124) but every action is subscripted with r instead of l. Thus, the behavior of the bulb controllers $[\![i]\!]$ for $1 \leq i \leq 6$ is given in Figure 5.6 (a) on page 123 and the behavior of the clock component in Figure 5.6 (b) on the same page. Figure 5.13 on the facing page depicts the behavior of the control component C.

Now, the interaction system modeling the street crossing is completely specified. In the following, we refer to this system as *Sys*.

We now apply Theorem 5.10 to this interaction system. First, we need to check the assumptions of the theorem. Observe that *Sys* has exclusive communi-

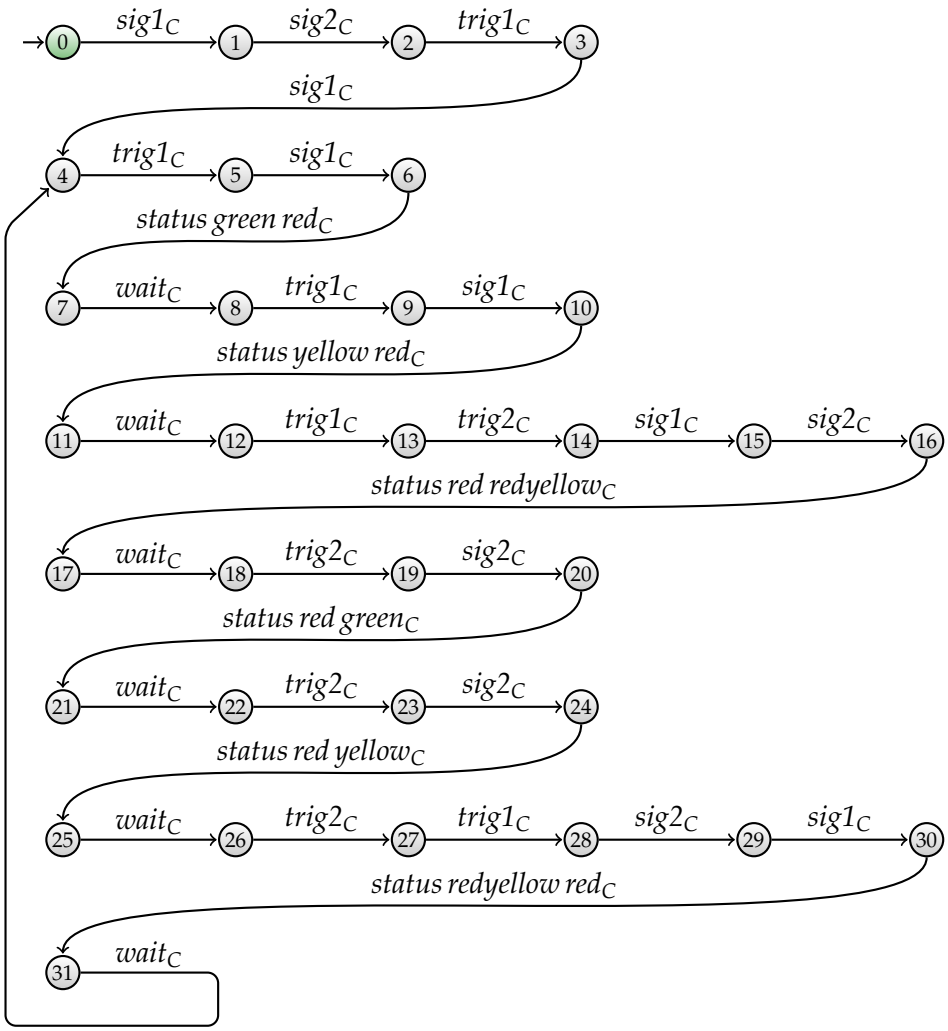


Figure 5.13: Behavior of the control component C

cation, which can easily be concluded from the interaction model depicted in Figure 5.12 because no action is used in interactions with different sets of participating components. We also have $|Comp| \geq 3$, i.e., at least three components are part of *Sys*. Next, we have to check the architecture of the interaction system. We compute the component graph (cf. Definition 4.1) which is depicted in Figure 5.14 on the following page.

Since it is a tree in the graph-theoretical sense, we conclude that *Sys* has a tree-like architecture (cf. Definition 4.3) where component C is the only central component. Thus, the assumptions of the theorem are satisfied and we can now take a look at the equivalences.

First, we need to determine the sets $\mathcal{C}_x^\star \subseteq 2^{Comp}$ for all $x \in \mathbb{N}$ with $\mathrm{rad}(G) \leq$

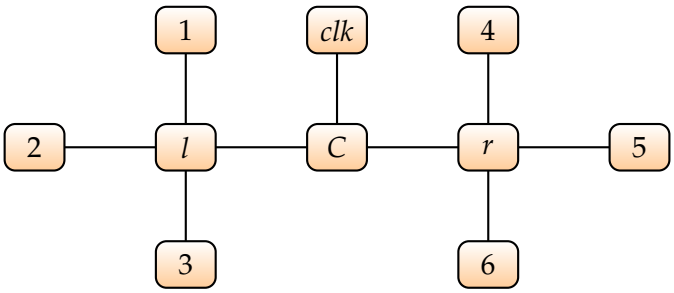


Figure 5.14: Component graph of the traffic light street crossing example

$x < \mathrm{diam}(G)$ where G denotes the component graph of *Sys* (cf. Figure 5.14). Observe that the graph depicted in Figure 5.11 on page 129, that we discussed at the beginning of Section 5.3.2, is isomorphic in the graph-theoretical sense to *Sys*'s component graph, i.e., the graphs are in fact the same up to the name of the vertices. Thus, the eccentricity of every vertex (cf. Definition A.7) in the component graph is given in Figure 5.11.

Here, we get $\mathcal{C}_2^\star = \{\{C, l, r, clk\}\}$ and $\mathcal{C}_3^\star = \{\{l, 1, 2, 3, C\}, \{r, 4, 5, 6, C\}\}$ where the middle component of each subsystem with a star-like architecture (to which the elements of these sets correspond) is the first component in each set (if we regard the given sequence of the components as an order). The last component in each such set corresponds to the border component with the smallest eccentricity or an arbitrary component if there are several such border components as in the case of the first element of $\mathcal{C}_2^\star$. Observe that each set of components is maximal with respect to satisfaction of the star-like architectural constraint.

Now, we denote these sets as in Theorem 5.10 as $C_{2.1}^\star = \{C, l, r, clk\}$, $C_{3.1}^\star = \{l, 1, 2, 3, C\}$, and $C_{3.2}^\star = \{r, 4, 5, 6, C\}$. Further, we assume that the following order functions are given:

- $f_{2.1} \colon C_{2.1}^\star \to \{1_{2.1}, 2_{2.1}, 3_{2.1}, 4_{2.1}\}$ with $f(C) = 1_{2.1}$, $f(l) = 2_{2.1}$, $f(r) = 3_{2.1}$, and $f(clk) = 4_{2.1}$,
- $f_{3.1} \colon C_{3.1}^\star \to \{1_{3.1}, 2_{3.1}, 3_{3.1}, 4_{3.1}, 5_{3.1}\}$ with $f(l) = 1_{3.1}$, $f(3) = 2_{3.1}$, $f(2) = 3_{3.1}$, $f(1) = 4_{3.1}$, and $f(C) = 5_{3.1}$, and
- $f_{3.2} \colon C_{3.2}^\star \to \{1_{3.2}, 2_{3.2}, 3_{3.2}, 4_{3.2}, 5_{3.2}\}$ with $f(r) = 1_{3.2}$, $f(6) = 2_{3.2}$, $f(5) = 3_{3.2}$, $f(4) = 4_{3.2}$, and $f(C) = 5_{3.2}$.

We now check for all x with $\mathrm{rad}(G) \leq x < \mathrm{diam}(G)$, all $x.y$ with $1 \leq y \leq |\mathcal{C}_x^\star|$,

and all $i_{x.y}$ with $2_{x.y} \leq i_{x.y} < n_{x.y}$ whether

$$Sys[\{f_{x.y}^{-1}(1_{x.y}), f_{x.y}^{-1}(i_{x.y}), f_{x.y}^{-1}(i_{x.y}+1)\}] \setminus\setminus \hat{I}_{f_{x.y}^{-1}(1_{x.y}),f_{x.y}^{-1}(i_{x.y})} \approx_b^{\Delta} Sys[\{f_{x.y}^{-1}(1_{x.y}), f_{x.y}^{-1}(i_{x.y}+1)\}] \setminus\setminus \hat{I}_{f_{x.y}^{-1}(1_{x.y}),f_{x.y}^{-1}(i_{x.y})}$$

holds where the parameter of the closing operator is defined as in Theorem 5.10.

Here, we first check, as in Example 5.8, for both light components whether the respective light component (l or r) together with the associated green bulb component (3 or 6, respectively) influences the observable behavior of the respective light and the corresponding yellow bulb components (l and 2 or r and 5, respectively), then wether this is the case for the respective light and corresponding yellow bulb components with respect to the associated red bulb component (1 or 4, respectively), and finally wether the respective light and associated red bulb component influence the observable behavior of the respective light and the control component (l and C or r and C, respectively). Afterwards, we check whether the control component C together with the light component l influences the observable behavior of C and the light component r and then whether this is the case for C and r with respect to the clock component. We omit to depict the single systems and to give the equivalence relations. But, all equivalences specified in the equation above hold.

We can thus conclude that

$$Sys \setminus\setminus \hat{I} \approx_b^{\Delta} Sys[\{C, clk\}] \setminus\setminus \hat{I}$$

holds where $\hat{I}$, as defined in Theorem 5.10, closes all interactions between the omitted components and the control component. Observe that the set $\{C, clk\}$ corresponds to the set M of the theorem because we have $|\text{center}(G)| = |\{C\}| = 1$ and thus $M = \text{center}(G) \cup K$ where $K = \{f_{\text{rad}(G).1}^{-1}(n_{\text{rad}(G).1})\} = \{f_{2.1}^{-1}(4_{2.1})\} = \{clk\}$.

We want to determine the savings that this reduction allows for. The reachable global behavior of Sys consists of 6974 states and 30 241 transitions. If we now compute the behavior of $Sys[\{C, clk\}] \setminus\setminus \hat{I}$, we see that this system only consists of 64 states and 64 transitions, i.e., the system is much smaller. As in Example 5.8, we can further reduce the system by computing the quotient of the system: $[\![Sys[\{C, clk\}] \setminus\setminus \hat{I}]\!]_{\approx_b^{\Delta}}$, which is depicted in Figure 5.15 on the following page, consists of 12 states and 12 transitions.

Now, we can for instance ask whether Sys is deadlock-free and instead of computing its global behavior, we can consider $[\![Sys[\{C, clk\}] \setminus\setminus \hat{I}]\!]_{\approx_b^{\Delta}}$. Since the latter labeled transition system does not contain a state without outgoing transitions (cf. Figure 5.15), we can conclude that the interaction system

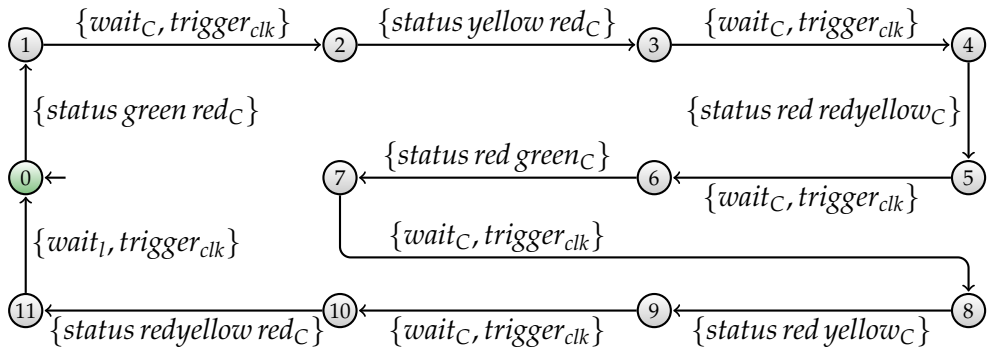


Figure 5.15: The labeled transition system $[\![Sys[\{C, clk\}] \setminus\!\setminus \hat{I}]\!]_{\approx_b^A}$

$Sys \setminus\!\setminus \hat{I}$ is deadlock-free and also that Sys is deadlock-free since deadlocks are preserved by closing.

This ends the discussion of the example.

Analogously to the discussion succeeding Theorem 5.7 and Example 5.8 respectively, we can now derive a corollary for Theorem 5.10 similar to Corollary 5.9. However, we omit such a corollary here because of its analogous nature.

Instead, we are interested in the costs of an application of the reduction approach for tree-like architectures. Clearly, we have to apply Corollary 5.9 or Theorem 5.7 respectively several times. To be more precise, the number of applications equals the number of maximal stars that can be derived from the tree-like architecture as illustrated in Figure 5.11 on page 129. Moreover, we need an automatic way to actually output the corresponding subsystems with a star-like architecture.

Since each component that has more than one neighbor in the component graph can be the middle component of a subsystem with a star-like architecture, we just have to count the corresponding vertices in the component graph, i.e., there are, for a given interaction system Sys, exactly $|Comp| - |\{i \in Comp \mid |\mathrm{nb}_G(i)| = 1\}|$ (cf. Definition A.5) many such subsystems. All we have to do in order to construct these subsystems—if Sys has a tree-like architecture—is to take each component with more than one neighbor in Sys's component graph, add all its neighbors, and use the resulting set for the subsystem construction operator (cf. Definition 3.18). Since the number of these subsystems is clearly bounded by the number of components, the overall approach is polynomial in the size of the input.

We want to finish the discussion of compositional reduction in interaction systems with a tree-like architecture by addressing where to start the equivalence checks. In the discussion preceding Theorem 5.10, we wrote that we

should work from the outer levels (with respect to the eccentricities) to the inner levels. Thus, we need to compute the eccentricity of every vertex and start from the periphery of the graph that contains the border components of the outermost subsystems. Now, if the corresponding equivalences hold, we successively remove the components at the periphery of the graph, recompute the periphery and check again. Clearly, this approach terminates by reaching the center of the graph and is polynomial in the size of the input if the eccentricities can be computed efficiently. Fortunately, the eccentricity of a vertex can be computed in linear time by a simple search strategy if the graph is a tree [264, Section 7.2.1], which is ensured by our tree-like architectural constraint.

In the next section, we take a look at the savings our approach allows for in interaction systems.

5.3.3 Compositional Reduction in Interaction Systems Allows For Exponential Savings

We discuss the potential savings that our reduction approach allows for. Consider the following interaction system *Sys* consisting of component 0 that is surrounded by n components $1, \ldots, n$. Each component has n actions, one for each of the other components. Thus, we have:

$Comp = \{0, 1, 2, \ldots, n\}$ and

$A_i = \{a_i^1, \ldots, a_i^n\}$ where i ranges over the components.

Now, the actions are used for cooperation among components but only component 0 cooperates with each of the other components, i.e., we have

$$Int = \big\{\{a_0^i, a_i^i\}, \{a_i^k\} \;\big|\; i \in \{1, \ldots, n\} \;\wedge\; k \in \{1, \ldots, n\} \setminus \{i\}\big\}.$$

We set $Int_{\text{closed}} = \emptyset$. Figure 5.16 on the following page illustrates the resulting interaction model.

Obviously, interaction system *Sys* has a star-like architecture (cf. Definition 4.2) because all interactions are either unary or binary and there is no cooperation where component 0 does not participate in. Thus, component 0 is the middle component and all other components are the border components of *Sys*. We omit to depict the component graph (cf. Definition 4.1) since it looks similar to the interaction model depicted in Figure 5.16.

Observe that the system also has exclusive communication (and even strongly exclusive communication), i.e., the first two assumptions of Corollary 5.9 of this chapter are already satisfied.

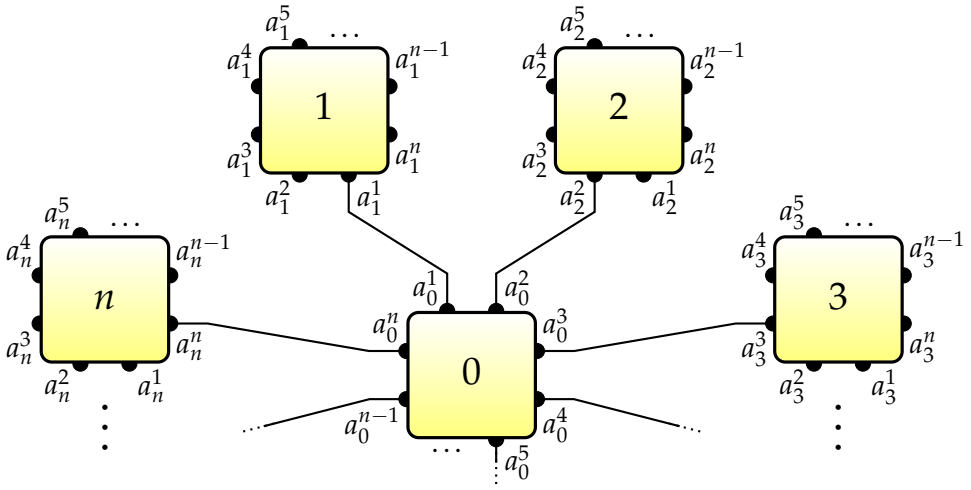


Figure 5.16: Interaction model of the example in Section 5.3.3

But before we further analyze this example, we have to specify *Sys*'s behavioral model. The behavior of each component is the same up to their action sets: Each component successively executes its actions in their natural order, i.e., component $i \in \mathit{Comp}$ executes its actions a_i^1, a_i^2, and so on until action a_i^n is executed which brings the component back to its initial state. The corresponding labeled transition system is depicted in Figure 5.17. Observe that the size of all labeled transition systems is linear in n.

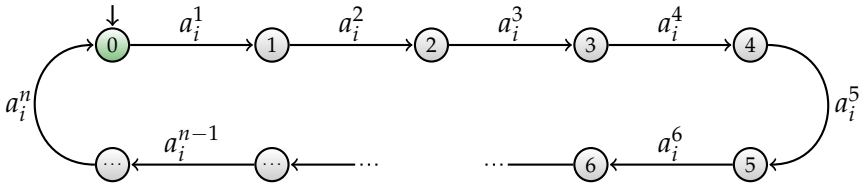


Figure 5.17: Behavior of component i for $0 \leq i \leq n$

Now, we take a closer look at the application of Corollary 5.9 to our example. We set $m = 0$ and $k = n$, and have to check whether for all distinct pairs of border components $i, j \in \mathit{Comp} \setminus \{m\}$ the following equivalence holds:

$$\mathit{Sys}[\{m,i,j\}] \setminus\setminus \hat{I}_{m,i} \approx_b^{\Delta} \mathit{Sys}[\{m,j\}] \setminus\setminus \hat{I}_{m,i}$$

with $\hat{I}_{m,i} := \big(\mathit{Int}[\{m,i\}] \setminus \mathit{Int}[\{m\}]\big) \cup \{\alpha \in \mathit{Int}[\{m\}] \mid \exists\, \beta \in \mathit{Int}[\{m,i\}] \colon \alpha \subset \beta\}$ which here is equal to $\hat{I}_{0,i} = \{\{a_0^i\}, \{a_0^i, a_i^i\}\} \cup \{\{a_i^k\} \mid k \in \{1,\ldots,n\} \setminus \{i\}\}$ for $1 \leq i \leq n$.

If we compute these equivalences, we find out that all of them indeed hold. Since the systems are already too big to be depicted here, we only consider the costs of this operation. For each computation, we have to construct two labeled

transition systems whose size is at most cubic in n. The branching bisimilarity (with explicit divergence) check can be carried out in time proportional to the product of the number of states and number of transitions in both systems (cf. Appendix E). Since the number of these computations is bounded by n^2, we can check whether all equivalences hold in polynomial time in n, which, admittedly, we already know from the discussion after Corollary 5.9.

We can thus conclude according to Corollary 5.9 that

$$Sys \setminus\setminus \hat{I} \approx_b^{\Delta} Sys[\{m,k\}] \setminus\setminus \hat{I}$$

holds with $\hat{I} = \bigcup_{i \in Comp \setminus \{m,k\}} \hat{I}_{m,i}$ which is equal to $\hat{I} = Int \setminus \big\{\{a_0^n, a_n^n\}, \{a_n^i\} \mid i \in \{1,\ldots,n\}\big\}$, i.e., all interactions are closed except the one that models the cooperation between components 0 and n and all singleton interactions of component n.

Now, we further reduce the global behavior of the last interaction system and compute its quotient, viz. $[\![Sys[\{m,k\}] \setminus\setminus \hat{I}]\!]_{\approx_b^{\Delta}}$ with $m = 0$ and $k = n$, which results in the labeled transition system depicted in Figure 5.18. The size of this system is linear in n.

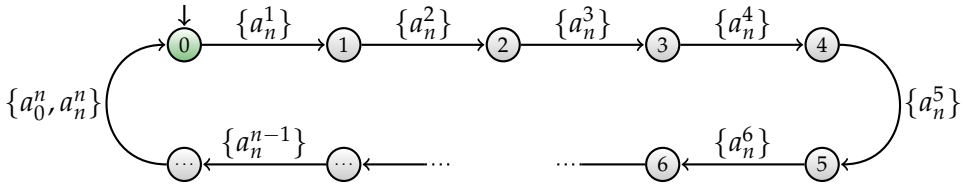


Figure 5.18: The labeled transition system $[\![Sys[\{m,k\}] \setminus\setminus \hat{I}]\!]_{\approx_b^{\Delta}}$ with $m = 0$ and $k = n$

Since the interaction model does not put strong restrictions on the cooperation of the components, all state combinations are reachable, i.e., the global state space of *Sys*'s global behavior consists of all n^{n+1} state combinations (each local component can be in one of its n states and there are $n+1$ components). Thus, the size of the global behavior of *Sys* is exponential in n.

Summarizing, the example shows that our reduction technique is able to save an exponential factor. All verification questions that are answerable in $Sys \setminus\setminus \hat{I}$ are also answerable in $Sys[\{0,n\}] \setminus\setminus \hat{I}$ if there are preserved by branching bisimilarity with explicit divergence which holds, e.g., for all CTL*–X formulae as discussed in Chapter 2.

In the next section, we take a closer look at the assumption of exclusive communication and whether we can drop it for our reduction approach.

5.4 Getting Rid Of Exclusive Communication

We want to examine where exactly the property of exclusive communication was needed in the previous sections. As we mentioned in Section 5.2, this property can be established via a transformation in polynomial time. But nevertheless, it is an interesting question whether we can get along without requiring this property for a given interaction system.

Thus, consider an interaction system with a star-like architecture. We learned in Section 5.3.1 that if a border component b of the system does influence the middle component m but there is a border component b' such that this influence is not observable later on, we can ignore b. This is achieved by closing the interactions needed for cooperation between m and b. In the resulting system, we also need to close those singletons that correspond to the actions of m that are present as interactions in the subsystem. If we now do not have exclusive communication, this closing may affect interactions that are needed for cooperation with other border components.

For instance if there is an interaction $\{a_m, a_b\}$, where a_m is an action of the middle component m and a_b is an interaction of border component b, we also need to close the interaction $\{a_m\}$ in the resulting system. But, if there is an interaction $\{a_m, a_{b'}\}$, where $a_{b'}$ is an action of border component b', the previous closing does not allow us to use a_m in the composition step. Observe that if we have exclusive communication, this situation cannot happen in a system with a star-like architecture since the cooperation partners for every action of the middle component need to be the same.

We thus have to find a way to not close the stub of the middle component. One way to achieve this is to keep all interactions. However then, we cannot consider valid subsystems since all components participating in the interactions need to be present. But, if we want to reduce the whole system, we can keep a very small representative of the components, that we previously ignored, after appropriate equivalences were established. We define such a small representative as the *always accepting version* of a component as follows.

Definition 5.12 (Always Accepting Version): Let *Sys* be an interaction system and let $i \in \mathit{Comp}$ be a component. The *always accepting version* of i, denoted by $\mathrm{Acc}(i)$, is the replacement of the local behavior $[\![i]\!]$ of component i with the labeled transition system $[\![\mathrm{Acc}(i)]\!] := (\{s_i^0\}, A_i, \{\{(s_i^0, s_i^0)\}\}_{a_i \in A_i}, \{s_i^0\})$ for an arbitrary initial state $s_i^0 \in S_i^0$.

Next, we formalize the idea using these small representatives. Instead of ignoring a border component, we check whether we can replace it by its

always accepting version. Observe that we do not require the property of exclusive communication in the following theorem.

Theorem 5.13: Let *Sys* be an interaction system with a star-like architecture and at least three components, i.e., $|Comp| \geq 3$. Let an arbitrary numbering of the components be given, i.e., a bijective function $f\colon Comp \to \{1,\ldots,n\}$ is given with $n = |Comp|$. Let number 1 denote the middle component, viz. $f^{-1}(1)$, of the star-like architecture. For convenience, we identify a component with respect to its unique number in the following. If for all i with $2 \leq i < n$ it holds that

$$Sys[\{1,i,i+1\}] \setminus\setminus \hat{I}_i \approx_b^{\Delta} Sys[\{1,\mathrm{Acc}(i),i+1\}] \setminus\setminus \hat{I}_i$$

with $\hat{I}_i := \{\alpha \in Int \mid \{i\} \subseteq \mathrm{compset}(\alpha)\}$, then it holds that

$$Sys \setminus\setminus \hat{I} \approx_b^{\Delta} Sys[\{1,\mathrm{Acc}(2),\ldots,\mathrm{Acc}(n-1),n\}] \setminus\setminus \hat{I}$$

with $\hat{I} := \bigcup_{i\in\{2,\ldots,n-1\}} \hat{I}_i$.

Proof of Theorem 5.13: Assume that the premise of the theorem holds. We show $Sys \setminus\setminus \hat{I} = Sys[\{1,2,\ldots,n-1,n\}] \setminus\setminus \hat{I} \approx_b^{\Delta} Sys[\{1,\mathrm{Acc}(2),\ldots,\mathrm{Acc}(n-1),n\}] \setminus\setminus \hat{I}$ by an inductive argument over the components. The first equality follows from Proposition 3.20. For $n = 3$ holds $Sys[\{1,2,3\}] \setminus\setminus \hat{I}_2 \approx_b^{\Delta} Sys[\{1,\mathrm{Acc}(2),3\}] \setminus\setminus \hat{I}_2$ because of the premise, and since $\hat{I}_2 = \hat{I}$ for $n = 3$ and branching bisimilarity with explicit divergence is a congruence with respect to closing the claim follows. For all components i with $2 \leq i < n$ holds:

$$Sys[\{1,\mathrm{Acc}(2),\ldots,\mathrm{Acc}(i-1),i,\ldots,n\}] \setminus\setminus \hat{I} \approx_b^{\Delta} Sys[\{1,\mathrm{Acc}(2),\ldots,\mathrm{Acc}(i),i+1,\ldots,n\}] \setminus\setminus \hat{I}$$

because (the used proposition is listed in parentheses above the equation symbol):

$$\begin{aligned}
&Sys[\{1,\mathrm{Acc}(2),\ldots,\mathrm{Acc}(i-1),i,\ldots,n\}] \setminus\setminus \hat{I}\\
&\overset{(3.21)}{=} \Big(Sys[\{1,i,i+1\}] \underset{\mathcal{I}_{\{1,i,i+1\},\{\mathrm{Acc}(2),\ldots,\mathrm{Acc}(i-1),i+2,\ldots,n\}}}{\otimes} Sys[\{\mathrm{Acc}(2),\ldots,\mathrm{Acc}(i-1),i+2,\ldots,n\}]\Big) \setminus\setminus \hat{I} \cup Int_{\mathrm{closed}}\\
&\overset{(3.23)}{=} \Big(Sys[\{1,i,i+1\}] \underset{\mathcal{I}_{\{1,i+1\},\{\mathrm{Acc}(2),\ldots,\mathrm{Acc}(i-1),i+2,\ldots,n\}}}{\otimes} Sys[\{\mathrm{Acc}(2),\ldots,\mathrm{Acc}(i-1),i+2,\ldots,n\}]\Big) \setminus\setminus \hat{I} \cup Int_{\mathrm{closed}}\\
&\overset{(3.15)}{=} \Big(Sys[\{1,i,i+1\}] \setminus\setminus \hat{I}_i \underset{\mathcal{I}_{\{1,i+1\},\{\mathrm{Acc}(2),\ldots,\mathrm{Acc}(i-1),i+2,\ldots,n\}}}{\otimes} Sys[\{\mathrm{Acc}(2),\ldots,\mathrm{Acc}(i-1),i+2,\ldots,n\}]\Big) \setminus\setminus \hat{I} \cup Int_{\mathrm{closed}}
\end{aligned}$$

$$
\begin{aligned}
&\overset{(pre.)}{\approx_b^\Delta} \Big(Sys[\{1, \mathrm{Acc}(i), i+1\}] \setminus\setminus \hat{I}_i \underset{\mathcal{I}_{\{1,i+1\},\{\mathrm{Acc}(2),\ldots,\mathrm{Acc}(i-1),i+2,\ldots,n\}}}{\otimes} \\
&\qquad\qquad Sys[\{\mathrm{Acc}(2), \ldots, \mathrm{Acc}(i-1), i+2, \ldots, n\}] \Big) \setminus\setminus \hat{I} \cup Int_{\text{closed}} \\
&\overset{(3.15)}{=} \Big(Sys[\{1, \mathrm{Acc}(i), i+1\}] \underset{\mathcal{I}_{\{1,i+1\},\{\mathrm{Acc}(2),\ldots,\mathrm{Acc}(i-1),i+2,\ldots,n\}}}{\otimes} \\
&\qquad\qquad Sys[\{\mathrm{Acc}(2), \ldots, \mathrm{Acc}(i-1), i+2, \ldots, n\}] \Big) \setminus\setminus \hat{I} \cup Int_{\text{closed}} \\
&\overset{(3.21)}{=} Sys[\{1, \mathrm{Acc}(2), \ldots, \mathrm{Acc}(i), i+1, \ldots, n\}] \setminus\setminus \hat{I}
\end{aligned}
$$

Thus, the argument lets us conclude that the theorem holds. Note that when we applied the premise in the reasoning above, we also used Propositions 3.12 and 3.17. ∎

Similarly to the reduction result for star-like architectures, we also derive a corollary that allows for an application of Theorem 5.13 without having access to an order function f.

Corollary 5.14: Let *Sys* be an interaction system with a star-like architecture. Let m denote the middle component and k an arbitrary border component. If for all distinct pairs of border components $i, j \in Comp \setminus \{m\}$ holds

$$Sys[\{m, i, j\}] \setminus\setminus \hat{I}_i \;\approx_b^\Delta\; Sys[\{m, \mathrm{Acc}(i), j\}] \setminus\setminus \hat{I}_i$$

with $\hat{I}_i := \{\alpha \in Int \mid \{i\} \subseteq \mathrm{compset}(\alpha)\}$, then it holds that

$$Sys \setminus\setminus \hat{I} \;\approx_b^\Delta\; Sys[\{m, k\} \cup \{\mathrm{Acc}(i) \mid i \in Comp \setminus \{m, k\}\}] \setminus\setminus \hat{I}$$

holds with $\hat{I} := \bigcup_{i \in Comp \setminus \{m,k\}} \hat{I}_i$.

A formal proof of Corollary 5.14 can be found in Appendix F on page 261.

We want to point out that the order function f of Theorem 5.13 can be derived from the equivalences of Corollary 5.14 in a similar way as we discussed in the remark following the proof of Corollary 5.9 (cf. page 127). The price we have to pay for not requiring exclusive communication beforehand is that we have to keep all components, although their behavior is replaced by a very compact version.

Here, we do not further deepen the discussion of exclusive communication, but observe that we can also derive a similar result for interaction systems with a tree-like architecture. Instead, we take a look at related work on compositional reduction in component-based settings in the next section.

5.5 Related Work

We discuss approaches from the literature that are comparable to our setting. In these works, the architecture is similarly defined as our component graph (cf. Definition 4.1)—as already mentioned in Chapter 4.

Bernardo et al. [39] exploit the architecture and behavioral equivalences in order to efficiently verify the property of deadlock-freedom with respect to systems specified in their architecture description language PADL, which we already discussed in Section 2.1.4. They introduce the notion of component compatibility [39, Definition 4.3] which means that the composite behavior of two components, where any joint action is concealed, is weakly bisimilar to the behavior of one of the components with corresponding concealed actions. For instance, if a component b which only interacts with one component m is compatible to m, then b is not important for the overall behavior of the system. Instead of considering both components one can safely consider only m.

This approach is extensible to the whole system and is particularly suited for acyclic architectures, such as star-like or tree-like ones. For instance in star-like architectures, the compatibility check is performed for every border component together with the middle component. The authors derive results for star-like and tree-like architectures [39, Theorem 4.5 and Corollary 4.6, respectively] that imply the deadlock-freedom of the whole system if all pairwise compatibility checks among the components succeed. Although their result opens the possibility for a reduction technique, in particular their Lemma 4.4, Bernardo et al. [39] do not address this feature explicitly such that the reduced version can be used later on instead of the original behavior for, e.g., other verification purposes.

Hennicker et al. [137] address the last point in their study of component neutrality in the semantic model of the Java/A architecture description language (cf. Section 2.1.4). Here, neutrality of a component x with respect to another component y means that the combined behavior (with binary connectors between the components which are unobservable in the behavior) is weakly bisimilar to the behavior of y where all involved connectors are replaced by unary ones (which are also unobservable in the behavior). The authors describe a reduction strategy based on this notion of neutrality that successively removes neutral components at the border of the architecture (which are hence called leaf components), i.e., the strategy is particularly applicable to acyclic architectures that have many leaf components. After the removal of neutral leafs, the reduction process can be repeated because potentially new leaf components are now present in the remaining system.

If the above system is considered as a composite component, the reduction strategy allows for the determination of the minimal observable behavior that can be substituted for the typically much larger original behavior of the new component in further composition steps. Thus, once the minimal behavior of the corresponding system part is determined, all verification steps that involve this part benefit from the smaller size of the behavior. For interaction systems, we shortly addressed in Section 3.7 that such an encapsulation technique can also be defined. Then, the ideas of this chapter also allow to determine the minimal observable behavior of a corresponding composite component.

Cheung and Kramer [61] propose a further method with respect to their study of context constraints for compositional reachability analysis. These constraints are modeled as so-called interface processes that represent the environment of a component, whose behavior is also modeled as a process, in a composition step. The key observation for such context constraints is that typically the behavior of a component is restricted by its cooperation with its environment, and if this restriction can be observed before the actual composition, which possibly results in a too large system, and does not restrict the overall behavior, then the smaller behavior with respect to the constraints by the environment can be substituted for the original behavior in analysis steps. The correctness of this approach is based on a similar idea as discussed above for the other formalisms which is called transparency [61, Section 6.3]: An interface process I capturing the context constraints of a composite process P is transparent to P if the composition of P and I is equivalent—with respect to strong bisimilarity—to P. The authors formulate a so-called interface theorem [61, Section 6.4] that provides conditions under which transparency is guaranteed.

However, the mentioned strategies are not suited for versatile components that offer many behavioral variants for different contexts as, e.g., our merchandise management example (cf. Section 1.3). This versatility supports the reusability property that is typically desired in component-based system design. Please note that all three strategies are comparable to the first idea for compositional reduction that we discussed in Section 5.1 at the beginning of this chapter, i.e., they are not applicable for our example Sys_{MMS}. For instance in the approach of Bernardo et al. [39], if a border component alters the behavior of the middle component but this alteration is not important for the interaction with the other components, the approach already fails.

Furthermore, we use branching bisimilarity with explicit divergence instead of weak bisimilarity, which is used by Bernardo et al. [39] and Hennicker et al. [137], because branching bisimilarity with explicit divergence preserves more properties of systems (remember that, as discussed in Section 2.4, a

logical characterization of branching bisimilarity with explicit divergence in CTL*–X exists if the system is deadlock-free), it is more efficient to calculate (cf. Appendix E), and, as remarked by Van Glabbeek and Weijland [115], many system that are weakly bisimilar are also branching bisimilar which also holds for the variant with explicit divergence.

Finally and as already mentioned in Section 1.1.3 of Chapter 1, more general techniques exist that also aim for a reduction of the state space in order to avoid or mitigate the state space explosion problem. For instance, Groote and Moller [126] discuss decomposition techniques for the verification of parallel systems such that the equivalence of two systems can be concluded by an analysis of appropriate subsystems. Larsen and Xinxin [167] study decomposition rules that are based on an operational semantics of contexts and show how a property of the composite system can be transformed into properties for the components. The cones and foci proof strategy [102, 129] shows how branching bisimilarity of two processes (described as linear process equations [40]) can be established without generating corresponding labeled transition systems. Several real-life protocols have been verified this way [102] and, although the approach is not completely automatic, it is interesting for our approaches of this chapter since we also want to show the branching bisimilarity (with explicit divergence) of two interaction systems without generating the global behavior of one of the systems.

We want to point out that there is an interesting connection to work on confluence [128, 197]. Interestingly, in all examples of this chapter, the labeled transition systems representing the global behavior (after the closing operator has been applied) are τ-confluent and a priorisation of the τ-transitions as described by Groote and Sellink [128] yields the same systems as in our reduction—except for the example from the beginning which is not convergent, i.e., it contains a cycle of τ-transitions. Given that the check for τ-confluence can be carried out very efficiently [127] and also in a compositional way [218], the relation of τ-confluence and our work is an interesting open problem, which we at this point have to postpone as future work.

This ends our discussion of compositional reduction in interaction systems. In the next section, we summarize the chapter and give some remarks regarding future work.

5.6 Summary and Future Work

In this chapter, we showed how the architectural constraints introduced in Chapter 4 can be used in conjunction with the equivalence for interaction

systems (cf. Section 2.4.1) to derive a reduction technique for interaction systems whose application is guaranteed to be polynomial in the size of the input, which is one of our main goals, and that thus allows for very efficient property verification. For instance, we can establish whether an interaction system is deadlock-free by an analysis of a particular subsystem that consists of only two components if all required equivalences hold.

We conclude with some remarks for future work regarding the order function f—as promised in the remark after the proof of Corollary 5.9 (cf. page 127).

The results of this chapter can be generalized as follows. Currently, we require that for each border component in a star-like architecture, we find a witness that guarantees that the influences of the border component on the observable behavior of the middle component are negligible. In doing so, we require that we find a unique witness for each border component (cf. Theorem 5.7) or that all other border components can serve as a witness (cf. Corollary 5.9).

Now, there is an additional scenario: One particular border component can serve as a witness for several or even all other border components. In order to allow for this scenario, we could replace the addition operator among the order function, that we use to get the next witness, by a witness function g as follows: $Sys[\{f^{-1}(1), f^{-1}(i), g(f^{-1}(i))\}]$ instead of $Sys[\{f^{-1}(1), f^{-1}(i), f^{-1}(i+1)\}]$. However, we have to ensure that we do not neglect a witness that is needed in later steps. Here, we do not deepen this idea and instead, take a closer look at deadlock-freedom in the next chapter.

Chapter 6

Efficient Deadlock Analysis

In this chapter, we focus on efficient ways to establish the property of deadlock-freedom in interaction systems, i.e., techniques whose computational complexity is guaranteed to be polynomial in the size of the input. The only assumption that we make about interaction systems under analysis beforehand is that they satisfy one of our architectural constraints which boils down to that they have a disjoint circular wait free architecture. We thus exploit, as mentioned in Chapter 4, the additional structure that is introduced among the possible cooperations of the components in order to overcome the computational complexity of the decision problem of deadlock detection. However, we know from Section 4.4.2 that we cannot hope for an efficient approach that works for all such constrained interaction systems—unless P = PSPACE is proven—but we nevertheless can derive sufficient conditions that imply the deadlock-freedom for some of these systems and that are checkable in polynomial time.

Before we go into the details of detecting deadlocks, we want to justify an assumption that comes with all our architectural constraints, viz. the graph-theoretical connectivity of the cooperation graph (cf. Definition A.6). What can we say about deadlock-freedom if this is not the case? Interestingly, we can already answer this question with results from earlier chapters. Suppose that we already determined the graph-theoretical connected components of the cooperation graph of a given interaction system *Sys*, which can be computed in linear time in the size of the graph by starting a depth-first search for each vertex contained in a different connected component and marking all vertices that are found [144]. Let $C_1 \cup \ldots \cup C_k = \mathit{Comp}$ be a partition of the set of components with respect to these connected components—alternatively, we can also use the component graph where the connected components directly correspond to the sets $C_1, \ldots, C_k$. Now, we can successively decompose interaction

system *Sys* with respect to these sets, i.e., we first consider the expansion with respect to Proposition 3.21 where we get with $C_{2,k} = C_2 \cup \ldots \cup C_k$ and Proposition 3.20:

$$Sys = Sys[C_1 \cup C_{2,k}] = \big(Sys[C_1] \underset{\mathcal{I}_{C_1,C_{2,k}}}{\otimes} Sys[C_{2,k}]\big) \setminus\!\setminus Int_{\text{closed}}[C_1 \cup C_{2,k}].$$

The composition information $\mathcal{I}_{C_1,C_{2,k}}$ equals the tuple $(\emptyset,\emptyset)$ (cf. Proposition 3.21) because otherwise, a component contained in C_1 cooperates with a component contained in $C_{2,k}$ and the set C_1 is thus not strictly contained in a graph-theoretical connected component of the cooperation graph—-if the elements of C_1 are regarded as singletons.

Fortunately, for such a composition situation the requirements of Theorem 3.26 are satisfied, and we can conclude from this theorem that *Sys* is deadlock-free if the two interaction systems $Sys[C_1]$ and $Sys[C_{2,k}]$ are deadlock-free. Note that the closing operator does not affect the deadlock-freedom of an interaction system as a consequence of Proposition 3.17: Branching bisimilarity with explicit divergence is a congruence with respect to this operator. For interaction system $Sys[C_1]$, we know that its cooperation graph is connected, i.e., we can tackle the system with an approach that is presented in this chapter to establish its deadlock-freedom if it is also disjoint circular wait free. If it has a tree-like architecture, we can also try to apply a result from Chapter 5. For the other interaction system, $Sys[C_{2,k}]$, we can repeat the idea from above and further decompose the system into subsystems with respect to the connected components of the cooperation graph.

Summarizing, the assumption of an architectural constraint that the cooperation graph is connected is not a restriction with respect to deadlock-freedom, because we can analyze appropriate subsystems as explained above to conclude the deadlock-freedom of the whole system.

Thus, we can now examine the property of deadlock-freedom more thoroughly and safely assume that the cooperation graph is connected. As we mentioned in Chapter 1, we here extend and earlier approach by Majster-Cederbaum and Martens [180]. We come back to a comparison of the approaches in Section 6.4. We proceed with observations about deadlocks that become the main idea behind the approaches that we consider in this chapter.

6.1 Observations about Deadlocks

A deadlock typically induces a circular waiting among some components, i.e., for each components' local state that is part of such a deadlocked global state

we have that either all interactions α are blocked by another component that participates in α, or the local state itself has no outgoing transition. Here, we only consider the first case, i.e., we reasonably assume that the local behavior of every component is "locally deadlock-free": At least one outgoing transition is present in any local state of the corresponding labeled transition system, i.e., we have $\mathrm{Suc}(s_i) \neq \emptyset$ for all $i \in \mathit{Comp}$ and $s_i \in S_i$ in a given interaction system *Sys*. Note that this can easily be checked in polynomial time by traversing each component's behavior or by analyzing the subsystem $\mathit{Sys}[\{i\}]$ for each $i \in \mathit{Comp}$ for deadlock-freedom as in Chapter 2—remember that the definition of a deadlock was given for an interaction system (cf. Definition 2.8).

We need to analyze the interactions that each local state wants to execute. The next definition allows us to do this in a convenient way. We also define a notion for states that corresponds to a first simple observation about deadlocks: If a local state of a component is able to execute an interaction that merely consists of a single action of the component, every global state where this local state is part of cannot be a deadlock. We call such a state *independent*.

Definition 6.1 (Performability and Independence): Let *Sys* be an interaction system. For a local state $s_i \in S_i$ of a component $i \in \mathit{Comp}$, we put $\mathit{Int}(s_i) := \{\alpha \in \mathit{Int} \mid \mathrm{Suc}(s_i, A_i \cap \alpha) \neq \emptyset\}$ and say for an interaction $\alpha \in \mathit{Int}(s_i)$ that i *wants to perform* α *in* s_i. Moreover, local state s_i is called *independent* if $\exists\, \alpha \in \mathit{Int}(s_i)\colon\ \mathrm{compset}(\alpha) = \{i\}$, otherwise s_i is called *dependent*.

We want to mention that the state property of independence in Definition 6.1 is called "complete" and "incomplete" respectively in the work of Majster-Cederbaum and Martens [180]. Originally, this notion was introduced by Gössler and Sifakis [121] as a property of certain interactions (cf. the discussion at the end of Section 2.1.2). However, we think that the name "independent" is more intuitive for states. Next, we formalize the above mentioned circular waiting observation about deadlocks.

Lemma 6.2 (Deadlock Properties): For every deadlock s in an interaction system *Sys*, there is a set $D \subseteq \mathit{Comp}$ such that the components in D can be ordered in such a way that each component is waiting for the next one in a circular way, i.e., each component $i \in D$ wants to perform an interaction α in its local state s_i of the deadlock ($\alpha \in \mathit{Int}(s_i)$), but α is not enabled in s because the next component is unable to perform it.

A formal proof of Lemma 6.2 can be found in Appendix F on page 261.

As mentioned in Chapter 4, architectural constraints are one approach to constrain circular waiting situations among the components. Such constraints are

typically defined by means of undirected graphs, e.g., the component graph (cf. Definition 4.1) or the cooperation graph (cf. Definition 4.4), which yield constraints such as star-like, tree-like, or disjoint circular wait free architectures (cf. Definitions 4.2, 4.3, and 4.6). The idea behind constraining the cooperation among the components to be acyclic, i.e., considering star-like and tree-like architectures, is that the absence of cycles in the graph reduces the presence of circular waiting situations. For interaction systems, such an approach was first taken by Majster-Cederbaum and Martens [179]. For other models of component-based systems and concurrency, similar approaches can be found in the literature [39, 49, 137]. However, as remarked in Section 4.2.2, with such an acyclicity requirement, any cooperation of three or more components induces a cycle in the corresponding graph, and thus disqualifies the system for analysis. We thus extended the component graph to the cooperation graph and derived the architectural constraint of disjoint circular wait freedom in Section 4.2.2. Originally, this extension was invented by Majster-Cederbaum and Martens [180] who demanded that the cooperation graph is acyclic. Here, we dropped this requirement and admitted certain cycles in disjoint circular wait free architectures.

We repeat our motivation of considering this potentially cyclic but disjoint circular wait free architecture with respect to our observation about deadlocks of this chapter. For ensuring deadlock-freedom, we have to consider any way the components are able to cooperate. Lemma 6.2 shows that a deadlock induces a circular waiting among some of the components. Since these components are also related in the cooperation graph, the superimposition of the waiting and cooperation information allows us to exclude those waiting situations where the reason of each single wait is completely independent from the other ones. As we see in the next section, the exclusion of the above mentioned waiting situations then allows for the establishment of a condition for deadlock-freedom of the whole system by an analysis of small, fixed size subsystems, which results in a polynomial-time bound of the approach.

In the following, we consider the database example introduced in Section 4.1 as our running example because, as we learned in Section 4.2.2, it has a disjoint circular wait free architecture (cf. Figure 4.7 on page 94).

6.2 Exploiting Disjoint Circular Wait Free Architectures

Before we exploit disjoint circular wait free architectures of interaction systems, we formalize the idea behind the superimposition of the waiting and cooperation information mentioned at the end of the previous section. As

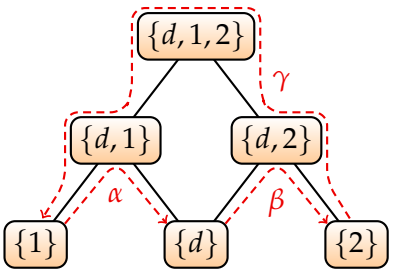
in Lemma 6.2, if we have a deadlock s in a given interaction system and a component i wants to execute an interaction α, i.e., $\alpha \in Int(s_i)$, but one of the other participating components j, i.e., $\{i,j\} \in \text{compset}(\alpha)$ and $i \neq j$, blocks this execution, i.e., $\alpha \notin Int(s_j)$, then we have a waiting situation: i waits for j because of α. Now, for the mentioned superimposition, we consider a certain path—called *cooperation path* in the following—in the cooperation graph that corresponds to this waiting situation.

6.2.1 Cooperation Paths

The following definition of *cooperation paths* corresponds to the mentioned superimposition of a waiting caused by a deadlock to the cooperation graph. Please note that a formal definition of a path can be found in Appendix A.

Definition 6.3 (Cooperation Path): Let Sys be an interaction system, and let $G_{\text{coop}} = (V, E)$ be its cooperation graph which is assumed to be connected. A finite path over G_{coop} is called *cooperation path* $\pi^{\alpha}_{i,j}$ for components $i, j \in Comp$ and interaction $\alpha \in Int$ with $i \neq j$ and $\{i,j\} \subseteq \text{compset}(\alpha)$ if it connects the corresponding vertices $\{i\}$, $\{j\}$, and $\text{compset}(\alpha)$ in G_{coop}, i.e., $|\pi^{\alpha}_{i,j}| = k$ for $k \in \mathbb{N}$, $\pi^{\alpha}_{i,j}[0] = \{i\}$, $\pi^{\alpha}_{i,j}[k-1] = \{j\}$, and $\pi^{\alpha}_{i,j}[k'] = \text{compset}(\alpha)$ for $k' \in \mathbb{N}$ with $0 < k' < k-1$.

Figure 6.1 illustrates cooperation paths of the database example $Sys_{\text{DB}}(n)$ for $n = 2$ clients together with possible waiting situations. Note that the specification of $Sys_{\text{DB}}(n)$ was given in Section 4.1.



$$\alpha = \{grant_d, request_1\}$$
$$\beta = \{update_d, write_2\}$$
$$\gamma = \{commit_d, commit_1, commit_2\}$$
$$\pi^{\alpha}_{1,d} = \langle \{1\}, \{d,1\}, \{d\} \rangle$$
$$\pi^{\beta}_{d,2} = \langle \{d\}, \{d,2\}, \{2\} \rangle$$
$$\pi^{\gamma}_{2,1} = \langle \{2\}, \{d,2\}, \{d,1,2\}, \{d,1\}, \{1\} \rangle$$

Figure 6.1: Three cooperation paths of our running example $Sys_{\text{DB}}(n)$ with $n = 2$. Additionally, possible "waiting for" situations are shown as red dashed lines, e.g., $\{1\} \dashrightarrow^{\alpha} \{d\}$ means component 1 waits for component d because of interaction α. Observe that each dashed line corresponds to the sequence of edges of a cooperation path, e.g., $\{1\} \dashrightarrow^{\alpha} \{d\}$ corresponds to $\pi^{\alpha}_{1,d}$.

Since we are interested in using cooperation paths to derive information about interaction systems with a disjoint circular wait free architecture, we

state three simple observations about cooperation paths between cooperating components in the following lemma.

Lemma 6.4 (Cooperation Path Properties): Let *Sys* be an interaction system and G_{coop} its cooperation graph which is assumed to be connected. For all components $i, j \in \mathit{Comp}$ and interactions $\alpha \in \mathit{Int}$ with $i \neq j$ and $\{i, j\} \subseteq \text{compset}(\alpha)$ exists a cooperation path $\pi^{\alpha}_{i,j}$ with the following properties:

1. Every vertex on the cooperation path is a subset of the set of components participating in α, i.e., $\forall\, v \in \pi^{\alpha}_{i,j} \colon v \subseteq \text{compset}(\alpha)$.
2. Every vertex on the cooperation path except the ones that represent components i and j contain at least two components, i.e., $\forall\, v \in \pi^{\alpha}_{i,j} \colon v \neq \{i\} \;\wedge\; v \neq \{j\} \implies |v| \geq 2$.
3. The cooperation path consists of at least three vertices, i.e., $|\pi^{\alpha}_{i,j}| \geq 3$.

A formal proof of Lemma 6.4 can be found in Appendix F on page 261.

In the following, we only consider cooperation paths that have the properties of Lemma 6.4. Observe that the proof of the lemma shows how such paths can be constructed for all suitable pairs of components and interactions. In the next section, we combine cooperation paths and our observations about deadlocks made in Section 6.1.

6.2.2 Cooperation Paths and Deadlocks

We now use the information about deadlocks of Lemma 6.2 to conclude the existence of a certain vertex in the cooperation graph. Intuitively, this vertex provides the information that at least two components have to (partially) wait for *each other* if the corresponding interaction system has a disjoint circular wait free architecture but is not deadlock-free.

Lemma 6.5 (Common Vertex): Let *Sys* be an interaction system and $G_{\text{coop}} = (V, E)$ its cooperation graph. Assume that *Sys* has a disjoint circular wait free architecture and contains a deadlock. There exist components $i, j, k \in \mathit{Comp}$ and interactions $\alpha, \beta \in \mathit{Int}$ with $i \neq j$, $j \neq k$, $\{i, j\} \subseteq \text{compset}(\alpha)$, $\{j, k\} \subseteq \text{compset}(\beta)$, and $\alpha \neq \beta$ such that two cooperation paths $\pi^{\alpha}_{i,j}$ and $\pi^{\beta}_{j,k}$ exist that have a vertex in common that does not represent a component, i.e., $\exists\, v \in V \colon v \in \pi^{\alpha}_{i,j} \;\wedge\; v \in \pi^{\beta}_{j,k} \;\wedge\; |v| \geq 2$.

A formal proof of Lemma 6.5 can be found in Appendix F on page 262.

Next, we combine our knowledge about cooperation paths from Lemma 6.4 and the existence of a common vertex in at least two such paths (if a deadlock exists) of Lemma 6.5 in order to prove a condition that is checkable among two components without considering whole cooperation paths, i.e., in the theorem we do not need to access elements of the cooperation graph.

Theorem 6.6 (Common Interactions): Let *Sys* be an interaction system. Assume that *Sys* has a disjoint circular wait free architecture and contains a deadlock s. There exist components $i, j \in Comp$ and interactions $\alpha, \beta \in Int$ with $i \neq j$, $\{i, j\} \subseteq \text{compset}(\alpha)$, $\{j\} \subseteq \text{compset}(\beta)$, $\alpha \in Int(s_i)$, $\alpha \notin Int(s_j)$, $\beta \in Int(s_j)$, and $|\text{compset}(\alpha) \cap \text{compset}(\beta)| \geq 2$ (where s_i and s_j are the local states of the components i and j in the global state s).

A formal proof of Theorem 6.6 can be found in Appendix F on page 262.

Theorem 6.6 offers a sufficient condition to verify the deadlock-freedom of interaction systems. To see this, consider the negation of the statement of the theorem, that we formalize as the following corollary. We also simplify the statement such that we only need to consider one interaction α instead of every pair of distinct interactions.

Corollary 6.7 (Absence of Deadlocks): Let *Sys* be an interaction system with a disjoint circular wait free architecture. For component $i \in Comp$ and state $s_i \in S_i$, let $\text{coopset}(s_i) := \bigcup_{\alpha \in Int(s_i)} \text{compset}(\alpha) \setminus \{i\}$ denote the set of components that *i wants to cooperate with* in s_i. If no two components $i, j \in Comp$, interaction $\alpha \in Int$, and local states $s_i \in S_i$ and $s_j \in S_j$ with $i \neq j$, $\{i, j\} \subseteq \text{compset}(\alpha)$, $\alpha \in Int(s_i)$, $\alpha \notin Int(s_j)$, and $\text{compset}(\alpha) \cap \text{coopset}(s_j) \neq \emptyset$ exist, then *Sys* contains no deadlock.

A formal proof of Corollary 6.7 can be found in Appendix F on page 263.

In the next section, we address how we can use the sufficient condition stated in Corollary 6.7 in an automated manner, i.e., how we can efficiently check whether such components, interactions, and local states as stated in the corollary exist. We also provide an example to illustrate this first deadlock analysis result.

6.2.3 Problematic States and Deadlock-Freedom

We use the information provided by Corollary 6.7 to define the notion of *problematic states* as follows that allows us to determine those components, interactions, and states that are candidates for the sufficient condition of the corollary.

Definition 6.8 (Problematic States): Let Sys be an interaction system. For component $i \in Comp$, dependent local state $s_i \in S_i$, interaction $\alpha \in Int(s_i)$, and component $j \in \text{compset}(\alpha) \setminus \{i\}$, we define $\text{PS}_j(s_i, \alpha) := \{s_j \in S_j \mid \alpha \notin Int(s_j) \wedge \text{compset}(\alpha) \cap \text{coopset}(s_j) \neq \emptyset \wedge s_j \text{ dependent} \wedge (\forall \beta \in Int(s_i) \cap Int(s_j) \colon \text{compset}(\beta) \neq \{i, j\}) \wedge (s_i, s_j) \text{ reachable in } [\![Sys[\{i, j\}]]\!]\}$.

Please note that we incorporated two observations to refine these sets with respect to Corollary 6.7: States are only problematic regarding deadlock-freedom to each other if both states are dependent—otherwise, they can execute a singleton interaction globally and thus are never involved in a deadlock. Similarly, since we compute the behaviors of subsystems of size two, if an interaction in which only components i and j participate in is enabled in a state combination, it is also globally enabled. The second observation concerns the reachability of state combinations: If the combination (s_i, s_j) of two states is not reachable in the behavior of the subsystem consisting of the components i and j, it is clear that this combination is not part of any reachable global state. With this definition, we are able to state a sufficient condition for deadlock-freedom that can be efficiently checked.

Theorem 6.9 (Deadlock-Freedom): Let Sys be an interaction system with a disjoint circular wait free architecture. Sys is deadlock-free if for all components $i \in Comp$, dependent local states $s_i \in S_i$, interactions $\alpha \in Int(s_i)$, and components $j \in \text{compset}(\alpha) \setminus \{i\}$ it holds that $\text{PS}_j(s_i, \alpha) = \emptyset$.

A formal proof of Theorem 6.9 can be found in Appendix F on page 263.

As an example, we determine the problematic states of the database example system $Sys_{\text{DB}}(n)$ according to Definition 6.8. For state $s_d^2 \in S_d$ and all components $i, j \in Comp \setminus \{d\}$ with $1 \leq i, j \leq n$ and $i \neq j$ and states $s_i^1 \in S_i$, we get (all other combinations are empty):

$$\text{PS}_i(s_d^2, \{update_d, write_i\}) = \{s_i^1\}, \qquad \text{PS}_i(s_d^2, \{inform_d, wait_i\}) = \{s_i^1\},$$
$$\text{PS}_j(s_i^1, \{commit_d, \ldots\}) = \{s_j^k \mid 2 \leq k \leq w_j + 1\}.$$

Note that these states are also intuitively problematic to each other, e.g., if the database d is in state s_d^2 and wants to inform client i which is already in state s_i^1, this is not possible because i does not offer its wait action in this state and wants to cooperate with d. Since not all sets of problematic states are empty, we cannot conclude deadlock-freedom for interaction system $Sys_{\text{DB}}(n)$ by means of Theorem 6.9.

But, lets take a look at our other running example, the merchandise management interaction system Sys_{MMS}. As we learned in Chapter 4, the system has a disjoint circular wait free architecture (cf. Figure 4.5 on page 94). Now, if

we compute the sets $\mathrm{PS}_j(s_i, \alpha)$ for all components $i \in \mathit{Comp}$, dependent states $s_i \in S_i$, interactions $\alpha \in \mathit{Int}(s_i)$, and components $j \in \mathrm{compset}(\alpha) \setminus \{i\}$, i.e., for all appropriate pairs of the customer, management, and storage components, we find out that no problematic states exist for $\mathit{Sys}_{\mathrm{MMS}}$. Thus, the deadlock-freedom of interaction system $\mathit{Sys}_{\mathrm{MMS}}$ is implied by Theorem 6.9.

Next, we take a closer look at the computational costs of applying Theorem 6.9 and derive an algorithm in order to be able to perform this task in an automatic way and to understand why the application of the theorem is guaranteed to be in polynomial time in the size of the input.

6.2.4 Algorithmic Treatment of Problematic States

In this section, we derive an algorithm that checks the condition of Theorem 6.9. We show how the sets of problematic states can be determined in an automatic way and address the computational costs of this operation. Clearly, in order to check the condition of the theorem, we just need to search for one problematic state since its existence implies that not all potential sets with respect to Definition 6.8 are empty.

But first, we need to initialize the new attributes that were introduced in this chapter, viz. the set $\mathit{Int}(s)$, the set $\mathrm{coopset}(s)$, and the dependency for all local states s of all components (cf. Definition 6.1 and Corollary 6.7), that are needed to determine whether problematic states exist. Algorithm 6.1 provides this initialization.

Algorithm 6.1 Initialization for the problematic states computation

PS-INITIALIZATION(Sys)

```
 1  INITIALIZATION(Sys) // cf. Algorithm B.1 on page 211
 2  for each component i ∈ Comp
 3      for each state s_i ∈ S_i
 4          s_i.dependent = TRUE
 5          s_i.Int = ∅
 6          s_i.coopset = ∅
 7          for each interaction α ∈ Int
 8              if i ∈ α.compset and not s_i.Suc_{α.i} == ∅
 9                  s_i.Int = s_i.Int ∪ {α}
10                  s_i.coopset = s_i.coopset ∪ α.compset
11                  if |α.compset| == 1
12                      s_i.dependent = FALSE
13          s_i.coopset = s_i.coopset \ {i}
```

As we address in Appendix B, the runtime bound for the initialization algorithm called in line 1 of Algorithm 6.1 is polynomial in the size of the input interaction system. Since we afterwards loop once through all components (cf. line 2) and local states (cf. line 3), and for each such state through all interactions (cf. line 7), we also have polynomial runtime costs for Algorithm 6.1.

Next, we need to examine all potential problematic state combinations which is carried out by the following algorithm. In its first step, Algorithm 6.2 checks wether the input interactions system has a disjoint circular wait free architecture which is an assumptions of Theorem 6.9. We addressed this architectural check in Section 4.3.2 of Chapter 4 where we learned that Algorithm B.6, which is called in line 1 of Algorithm 6.2 and given in Appendix B, checks whether an interaction system has a disjoint circular wait free architecture in polynomial time in the number of components and interactions.

Algorithm 6.2 Check of the condition of Theorem 6.9

DEADLOCK-FREEDOM-CONDITION(Sys)

```
1  if not DCWF(Sys) // cf. Algorithm B.6 on page 218
2       return FALSE
3  PS-INITIALIZATION(Sys) // cf. Algorithm 6.1 on page 155
4  for each component i ∈ Comp
5       for each state s_i ∈ S_i
6            if s_i.dependent
7                 for each interaction α ∈ s_i.Int
8                      for each component j ∈ α.compset \ {i}
9                           for each state s_j ∈ S_j
10                               if IS-PROBLEMATIC(Sys, i, s_i, α, j, s_j)
11                                    return FALSE   // ↪ cf. Algorithm 6.3
12 return TRUE
```

Algorithm 6.3 Problematic state decision with respect to Definition 6.8

IS-PROBLEMATIC($Sys, i, s_i, \alpha, j, s_j$)

```
1  if α ∉ s_j.Int and not s_j.coopset ∩ α.compset == ∅ and s_j.dependent
2       for each interaction β ∈ s_i.Int ∩ s_j.Int
3            if β.compset == {i, j}
4                 return FALSE
5       (S_{i,j}, Σ, {—a→}_{a∈Σ}, S^0_{i,j}) = BEHAVIOR-TRAVERSAL(Sys[{i, j}])
6       return (s_i, s_j) ∈ S_{i,j}   // ↪ cf. Algorithm B.2, use hashing to cache
7  else return FALSE                  // results and avoid "double" computations.
```

Algorithm 6.2 enumerates all combinations that possibly constitute a problematic state, i.e., all dependent states s_i and s_j of components i and j respectively and interactions α such that $\alpha \in Int(s_i)$ and $j \in \text{compset}(\alpha) \setminus \{i\}$ holds (cf. Definition 6.8). For each combination, Algorithm 6.3 on the facing page is called in line 10 of Algorithm 6.2, which carries out the further checks.

Observe that the if-clause in the first line of Algorithm 6.3 corresponds to the first three terms of the logical conjunction in the definition of $\text{PS}_j(s_i, \alpha)$ (cf. Definition 6.8). If it is true, we check in lines 2–4 whether an interaction is performable in both s_i and s_j where only components i and j participate in, which means that the state combination (s_i, s_j) cannot be part of a deadlock. Finally, we check the reachability of (s_i, s_j) in the subsystem consisting of the components i and j in line 5. As we mentioned in Section 2.2 of Chapter 2, Algorithm B.2, which is given in Appendix B, computes an interaction system's reachable global behavior and we can thus decide the reachability of (s_i, s_j) by checking whether it is contained in the state space of the labeled transition system returned by Algorithm B.2. Observe that we treated all five terms in the logical conjunction that defines a problematic state.

Now, we are interested in the computational costs of these algorithms. One call of Algorithm 6.3 can be performed in $\text{O}(|Int| + |S_{\max}|^2 \cdot |Int|)$ where the latter summand corresponds to the costs of Algorithm B.2 for two components and $S_{\max}$ denotes the largest local state space. We have to apply this algorithm to all pairs of local states and interactions in the worst case, i.e., $\text{O}(|Comp|^2 \cdot |S_{\max}|^2 \cdot |Int|)$ times. This means that Algorithm 6.2 has a polynomial runtime bound. Thus, we derived an efficient approach according to the computational view and our goals mentioned in Chapter 1. However, we can dramatically speed up this computation by caching the results of the traversal of the subsystems, which is indicated as a comment after line 5 of Algorithm 6.3.

In the next section, we further refine the information provided by problematic states to also verify the deadlock-freedom of the database example.

6.3 Refinement: Problematic States Reachability

In the previous section, we saw that deadlock-freedom of the merchandise management example Sys_{MMS} can be concluded without constructing the global behavior $[\![Sys_{\text{MMS}}]\!]$, but for the database example $Sys_{\text{DB}}(n)$ the approach failed. Here, we refine the information provided by problematic states where we use a similar observation on the reachability of global states as Majster-Cederbaum and Martens [180]. We motivate this refinement by means of the database example.

Consider $Sys_{\mathrm{DB}}(n)$ with $n = 2$ and the global state (s_d^1, s_1^2, s_2^2). We want to exclude its reachability by only considering subsystems of size two. In state s_d^1, component d wants to cooperate with client 1 and client 2, i.e., $\mathrm{coopset}(s_d^1) = \{1, 2\}$—cf. Corollary 6.7 for the definition of the set $\mathrm{coopset}(s)$ for a component's local state s. Now, we take a look at $[\![Sys_{\mathrm{DB}}(2)[\{d, 1\}]]\!]$ and see that the state (s_d^1, s_1^2) is *only* reachable by performing interaction $\{grant_d, request_1\}$ and analogously in $[\![Sys_{\mathrm{DB}}(2)[\{d, 2\}]]\!]$, the state (s_d^1, s_2^2) is *only* reachable by $\{grant_d, request_2\}$. Comparing these two interactions, we see that they are not consistent, i.e., only one of them can be used to enter either (s_d^1, s_1^2, s_2^0) or (s_d^1, s_1^0, s_2^2).

In the next section, we use this observation to exclude the reachability of problematic state combinations in interaction systems that have a disjoint circular wait free architecture and are deadlock-free.

6.3.1 Non-Interfering Backward Reachable Set and Entry Interactions

Let *Sys* be an interaction system with a disjoint circular wait free architecture. Assume that *Sys* is deadlock-free but we find problematic states. Consider a (potentially reachable) global state $s = (\ldots, s_i, \ldots, s_j, \ldots, s_k, \ldots)$, i.e., we fix the three components i, j, and k and three of their local states that are part of the global state s. From the point of view of component i, this global state is a potential deadlock if i wants to execute, say, two interactions β_j and β_k, i.e., we have $Int(s_i) = \{\beta_j, \beta_k\}$, but s_j and s_k are problematic states, i.e., $s_j \in \mathrm{PS}_j(s_i, \beta_j)$ and $s_k \in \mathrm{PS}_k(s_i, \beta_k)$ holds. Recall that we know from Definition 6.8 that $j \in \mathrm{compset}(\beta_j)$ and $\mathrm{compset}(\beta_j) \cap \mathrm{coopset}(s_j) \neq \emptyset$ holds, which means that component j in state s_j also wants to cooperate with i or with one of the components i wants to cooperate with in state s_i, i.e., the components i and j are or share cooperation partners in their states s_i and s_j. The same observation can be made for component k and interaction β_k.

We now use this observation to derive information about the reachability of the global state s—similarly to the idea mentioned at the beginning of Section 6.3 for $Sys_{\mathrm{DB}}(2)$. We ask: How can the cooperation partners of the components influence the reachability of the potential deadlock s (from the point of view of component i)? Here, we mean by influencing whether the cooperation partners are also involved on the way through the global behavior, i.e., in interactions that lead to global state s (on a global execution path).

First, we want to exclude the situation that component i reaches state s_i without cooperating with one of its later cooperation partners, i.e., the components contained in $\mathrm{coopset}(s_i)$. In order to achieve this, we backwardly search i's

state space starting in state s_i and only consider actions that are used in interactions where i's cooperation partners in s_i have no influence on. In the comparison of the interactions which lead to a certain global state, these states can be reached as intermediate steps without affecting the reachability of a state combination in question. We define this information as follows similarly to the notion of "backward search" of Majster-Cederbaum and Martens [180] which we here, as already mentioned in Chapter 1, adjust in order to work without previously establishing the property of exclusive communication (cf. Section 5.2).

Definition 6.10 (Non-Interfering Backward Reachable Set): Let *Sys* be an interaction system. We define the *non-interfering backward reachable set (NBRS)* of a state $s_i \in S_i$ of a component $i \in \mathit{Comp}$ as the set of all states from which s_i is reachable without using actions that are (only) used for cooperation with components that i wants to cooperate with in s_i:

$$\begin{aligned}
\mathrm{NBRS}^0(s_i) &:= \{s_i\} \\
\mathrm{NBRS}^{l+1}(s_i) &:= \big\{t_i \in S_i \mid \exists\, r_i \in \mathrm{NBRS}^l(s_i) \colon t_i \in \mathrm{Pre}(r_i, \{a_i \in A_i \mid \\
&\qquad\qquad \exists\, \alpha \in \mathit{Int} \colon a_i \in \alpha \ \wedge\ \mathrm{compset}(\alpha) \cap \mathrm{coopset}(s_i) = \emptyset\})\big\} \\
\mathrm{NBRS}(s_i) &:= \bigcup_{l \in \mathbb{N}} \mathrm{NBRS}^l(s_i)
\end{aligned}$$

where $\mathrm{Pre}(s_i, A)$ denotes the A-predecessors of a state s_i as introduced in Definition A.2.

For instance, computing the NBRSs for the running example $\mathit{Sys}_{\mathrm{DB}}(n)$ yields for all clients i and states $s_i \in S_i$: $\mathrm{NBRS}(s_i) = \{s_i\}$ and for the states of the database d:

$$\begin{aligned}
&\mathrm{NBRS}(s_d^0) = \{s_d^0, s_d^3\}, && \mathrm{NBRS}(s_d^2) = \{s_d^2\}, \text{ and} \\
&\mathrm{NBRS}(s_d^1) = \{s_d^1\}, && \mathrm{NBRS}(s_d^3) = \{s_d^3, s_d^2, s_d^1, s_d^0\} = S_d.
\end{aligned}$$

Here, the NBRS of state s_d^3 is d's whole state space because in this state the component only "cooperates" with itself. However, the NBRS of state s_d^0 shows that for reachability concerns also combinations involving s_d^3 have to be considered since component d can transit from this state to s_d^0 without an influence of any cooperation partner.

Thus, coming back to *Sys* and its global state $s = (\ldots, s_i, \ldots, s_j, \ldots, s_k, \ldots)$, if the components j and k can influence the reachability of i's state s_i, we have to assume that $\mathrm{NBRS}(s_i) \cap I_i = \emptyset$ holds, i.e., there is no initial state of i from which s_i can be reached without cooperating with component j or k.

With this additional assumption, we can now take a look at the paths leading to pairwise state combinations, i.e., how can (s_i, s_j) and (s_i, s_k) respectively

be reached—as we did for interaction system $Sys_{\mathrm{DB}}(2)$ and the global state (s_d^1, s_1^2, s_2^2) at the beginning of Section 6.3. Since we have $s_j \in \mathrm{PS}_j(s_i, \beta_j)$, we know that $\mathrm{compset}(\beta_j) \cap \mathrm{coopset}(s_j) \neq \emptyset$ holds, which means that $\mathrm{NBRS}(s_j)$ contains all states from which the reachability of s_j in j's behavior cannot be influenced by one of its cooperation partners in s_j which are also cooperation partners of component i in state s_i.

Now, we search for the set of those interactions α such that a global state could be reached where component i is in state s_i and component j is in state s_j and no cooperation partner of i in state s_i can influence this reachability after the execution of such an α. We define such interactions as follows.

Definition 6.11 (Entry Interactions): Let Sys be an interaction system. We define the *entry interactions (EI)* of a state $s_i \in S_i$ of a component $i \in Comp$ and a state $s_j \in S_j$ of a component $j \in Comp \setminus \{i\}$ as those interactions $\alpha \in Int$ such that an action a_i of i, i.e., $a_i \in A_i$, is used for cooperation with components that i wants to cooperate with in s_i and a_i is also used in α to enter a reachable state in the behavior of the subsystem of i and j from which the state (s_i, s_j) can be reached without using actions that are (only) used for cooperation with components that i (and j resp.) wants to cooperate with in s_i (and s_j resp.):

$$\begin{aligned}\mathrm{EI}(s_i, s_j) := \big\{ & \alpha \in Int \mid \exists\, a_i \in A_i \colon a_i \in \alpha \\ & \wedge \ \big(\exists\, \beta \in Int \colon a_i \in \beta \ \wedge\ \mathrm{compset}(\beta) \cap \mathrm{coopset}(s_i) \neq \emptyset\big) \\ & \wedge \ \big(\exists\, (r_i, r_j) \in \mathrm{NBRS}(s_i) \times \mathrm{NBRS}(s_j) \ \ \exists\, (t_i, t_j) \in S_i \times S_j \colon \\ & \qquad t_i \xrightarrow{a_i}_i r_i \ \wedge\ \big((j(\alpha) = \{a_j\} \ \wedge\ t_j \xrightarrow{a_j}_j r_j) \vee (j(\alpha) = \emptyset \\ & \qquad \wedge\ t_j = r_j)\big) \ \wedge\ (t_i, t_j) \text{ reachable in } [\![Sys[\{i,j\}]]\!]\big)\big\}.\end{aligned}$$

Thus, if there is an entry interaction $\alpha \in \mathrm{EI}(s_i, s_j)$ and it gets executed globally, then a global state $(\ldots, s_i, \ldots, s_j, \ldots)$ can be reached afterwards and no cooperation partner of component i in state s_i can influence this reachability once α has been executed. We can proceed similarly for s_i and s_k and also compute $\mathrm{EI}(s_i, s_k)$. Now, if we find an interaction α that is contained in both of these sets, i.e., $\alpha \in \mathrm{EI}(s_i, s_j) \cap \mathrm{EI}(s_i, s_k)$, we know that after the execution of α a global state can be reached such that component i is in state s_i, j is in state s_j, and k is in state s_k and no cooperation partner of i in s_i can influence this reachability after α has been executed. But conversely, i.e., if no such α exists, we can conclude that such a global state cannot be reached—however, if there is such an α, we gained no additional information since it could be the case that no global state is reachable where interaction α is enabled. But, this idea is a criterion for excluding the reachability of problematic state combinations.

Summarizing, we exploit the above observations that we made for state s_i in Sys—similarly to Majster-Cederbaum and Martens [180]: In an interaction

system with a disjoint circular wait free architecture, a state $s_i \in S_i$ of any component $i \in \mathit{Comp}$ is only part of a global deadlock if *all* interactions α that i wants to perform in s_i are blocked by the corresponding cooperation partners, i.e., for all such $\alpha \in \mathit{Int}(s_i)$ there is a component $j \in \mathrm{compset}(\alpha) \setminus \{i\}$ that is in a state $s_j \in S_j$ where it does not want to perform α, i.e., $\alpha \notin \mathit{Int}(s_j)$ holds. Thus, since we want to determine which entry interactions of s_i may lead to a state where α is not enabled, we take the union of all entry interactions of problematic states of components participating in α. Since this argument may only result in a global deadlock if it holds for all interactions that s_i wants to perform, i.e., if there is an entry interaction that is part of the union as above for all $\alpha \in \mathit{Int}(s_i)$, we compare these sets: In order to determine whether such an interaction exists, we compute the intersection of all these unions.

In the next section, we formalize the above discussion.

6.3.2 Refined Condition for Deadlock-Freedom

The following theorem formalizes the discussion after the definition of the entry interactions (cf. Definition 6.11) above. The first condition ensures that we actually are able to perform the described comparison, because if a state can be reached from an initial state without using actions that are (only) needed for cooperation with its cooperation partners, we cannot rely on any entry information of this state. Hence, we simply demand that no such state is reachable, or otherwise, that no corresponding problematic state of another component is also reachable in this way.

Theorem 6.12 (Refined Deadlock-Freedom): Let *Sys* be an interaction system with a disjoint circular wait free architecture. For an interaction $\alpha \in \mathit{Int}$, let $\mathrm{cycleset}(\alpha)$ denote the set of cycle components that participate in α (cf. Definition 4.6). If the following two conditions hold, then *Sys* is deadlock-free:

1. For all components $i \in \mathit{Comp}$ and dependent local states $s_i \in S_i$ it holds that $\mathrm{NBRS}(s_i) \cap S_i^0 = \emptyset$ or there is an interaction $\alpha \in \mathit{Int}(s_i)$ such that for all components $j \in \mathrm{compset}(\alpha) \setminus \{i\}$ holds $\left(\bigcup_{s_j \in \mathrm{PS}_j(s_i,\alpha)} \mathrm{NBRS}(s_j)\right) \cap S_j^0 = \emptyset$.
2. For all interactions $\alpha \in \mathit{Int}$ with $|\mathrm{compset}(\alpha)| \geq 2$ exists a component $k \in \mathrm{compset}(\alpha)$ such that for all components $i \in \mathrm{cycleset}(\alpha) \cup \{k\}$ and all dependent local states $s_i \in S_i$ holds

$$\alpha \notin \bigcap_{\beta \in \mathit{Int}(s_i)} \bigcup_{j \in \mathrm{compset}(\beta) \setminus \{i\}} \bigcup_{s_j \in \mathrm{PS}_j(s_i,\beta)} \mathrm{EI}(s_i, s_j).$$

A formal proof of Theorem 6.12 can be found in Appendix F on page 263.

For enhanced readability, we use for a state s_i of a component i and an interaction $\beta \in \mathit{Int}$ the abbreviation $\mathrm{PEI}(s_i, \beta) := \bigcup_{j \in \mathrm{compset}(\beta) \setminus \{i\}} \bigcup_{s_j \in \mathrm{PS}_j(s_i, \beta)} \mathrm{EI}(s_i, s_j)$ for the union of entry interactions, which we call the *problematic entry interactions*, and $\mathrm{IPEI}(s_i) := \bigcap_{\beta \in \mathit{Int}(s_i)} \mathrm{PEI}(s_i, \beta)$ for the corresponding intersections, which we call the *intersection of problematic entry interactions*.

We continue with the database example $\mathit{Sys}_{\mathrm{DB}}(n)$ where we get for component d and all components $i \in \mathit{Comp} \setminus \{d\}$ (all other sets are empty):

$$\begin{aligned}\mathrm{PEI}(s_d^2, \{\mathit{update}_d, \mathit{write}_i\}) &= \mathrm{PEI}(s_d^2, \{\mathit{inform}_d, \mathit{wait}_i\}) \\ &= \textstyle\bigcup_{1 \le j \le n} \big\{\{\mathit{inform}_d, \mathit{wait}_j\}, \{\mathit{update}_d, \mathit{write}_j\}\big\}, \\ \mathrm{PEI}(s_i^1, \{\mathit{commit}_d, \mathit{commit}_1, \ldots, \mathit{commit}_n\}) &= \big\{\{\mathit{inform}_d, \mathit{wait}_i\}, \\ &\qquad \{\mathit{update}_d, \mathit{write}_i\}\big\}.\end{aligned}$$

These problematic entry interactions correspond to the intuition, e.g., the entry interactions $\mathrm{EI}(s_d^2, s_i^1)$ of state s_d^2 of d and state s_i^1 of a client i are that d informs the client or updates its database with the value provided by the client.

This results in the following intersections: For all $i \in \mathit{Comp} \setminus \{d\}$ and $s_i^1 \in S_i$, we get (all other sets are empty):

$$\mathrm{IPEI}(s_i^1) = \big\{\{\mathit{inform}_d, \mathit{wait}_i\}, \{\mathit{update}_d, \mathit{write}_i\}\big\}.$$

Now, we see that for all $s_d \in S_d$ the set $\mathrm{IPEI}(s_d)$ is empty. Since component d participates in every interaction $\alpha \in \mathit{Int}$ with $|\mathrm{compset}(\alpha)| \geq 2$, $\mathrm{cycleset}(\alpha) = \{d\}$ holds, and $\alpha \notin \mathrm{IPEI}(s_d)$ for all s_d, the second condition of Theorem 6.12 holds. Note that also the first condition holds, since for all dependent local states s_i of all components $i \in \mathit{Comp}$ holds $\mathrm{NBRS}(s_i) \cap S_i^0 = \emptyset$—except for state s_d^0. But for this state, we learned in Section 6.2.3 (after the statement of Theorem 6.9) that all problematic state sets are empty, i.e., $\mathrm{PS}_i(s_d^0, \alpha) = \emptyset$ for all suitable components i and interactions α, which makes the first condition true for all dependent states. Thus, we can conclude that $\mathit{Sys}_{\mathrm{DB}}(n)$ is deadlock-free.

Next, we take a closer look at the computational costs of applying Theorem 6.12 and derive an algorithm in order to be able to perform this task in an automatic way and to understand why the application of the theorem is guaranteed to be in polynomial time in the size of the input.

6.3.3 Algorithmic Treatment of the Refined Condition

In this section, we derive an algorithm that checks the two conditions of Theorem 6.12. We show how the sets of interesting problematic entry interactions can be determined in an automatic way and address the computational costs

of this operation. First, we need to initialize the new attributes that were introduced in the previous section, viz. the set NBRS(s), the non-interfering backward reachable set, of all component's local states s (cf. Definition 6.10). Algorithm 6.4 provides this initialization.

Algorithm 6.4 Computation of the NBRS with respect to Definition 6.10

NBRS-COMPUTATION(Sys)

```
 1  PS-INITIALIZATION(Sys) // cf. Algorithm 6.1 on page 155
 2  for each component i ∈ Comp
 3      for each state s_i ∈ S_i
 4          NBRS-Actions = ∅
 5          s_i.coop-Actions = ∅
 6          for each interaction α ∈ Int
 7              if i ∈ α.compset
 8                  if α.compset ∩ s_i.coopset == ∅
 9                      NBRS-Actions = NBRS-Actions ∪ α.i
10                  else s_i.coop-Actions = s_i.coop-Actions ∪ α.i
11          s_i.NBRS = ∅        // ↪ The coop-Actions of a state s_i are those
12          current = {s_i}     // actions a_i ∈ A_i such that ∃ β ∈ Int: a_i ∈ β
13          repeat              // ∧ compset(β) ∩ coopset(s_i) ≠ ∅.
14              s_i.NBRS = s_i.NBRS ∪ current
15              found = ∅
16              for each state t_i ∈ current
17                  for each action a ∈ NBRS-Actions
18                      found = found ∪ t_i.Pre_a
19              current = found \ s_i.NBRS
20          until current == ∅
```

We learned in Section 6.2.4 of this chapter that the runtime bound for Algorithm 6.1 called in line 1 of Algorithm 6.4 is polynomial in the size of the input. After the initialization, we loop for each component and each of its local states through all interactions to determine the set of actions that are needed for the NBRS computation, i.e., all actions $a_i \in A_i$ for a component i and one of its states s_i such that $\exists\, \alpha \in Int\colon a_i \in \alpha \ \wedge\ \text{compset}(\alpha) \cap \text{coopset}(s_i) = \emptyset$ (cf. Definition 6.10). Here, we can directly store the non-NBRS actions because they are needed later for the computation of the entry interactions (cf. the comment in lines 11–13 of Algorithm 6.4). After the treatment of the actions, we then compute the NBRS for each state where we just have to traverse the state space backwardly using only the actions computed in the previous step. Overall, these nested loops run in time $O(|Comp| \cdot |S_{\max}| \cdot (|Int| + |S_{\max}| \cdot |A_{\max}|))$ where $S_{\max}$ and $A_{\max}$ denote the largest local state space and action set respec-

tively. We here assume that computing the set containment in line 7 and the set intersection in line 8 can be carried out in constant time, otherwise a factor of $|Comp|$ needs to be added to the upper bound. Thus, the whole runtime of Algorithm 6.4 is polynomial in the size of the input.

With this initialization, we can now compute the sets $\text{IPEI}(s_i)$ and $\text{PEI}(s_i, \alpha)$ respectively for a state $s_i \in S_i$ of a component i and an interaction $\alpha \in Int$, which allow to check the conditions of Theorem 6.12. Algorithm 6.5 on the facing page provides this computation.

The algorithm is called with an interaction system *Sys*. In the first step, cf. line 1 of Algorithm 6.5, we check wether the input interaction system has a disjoint circular wait free architecture which is an assumption of Theorem 6.12 and which can be decided in polynomial time (cf. Section 4.3.2). We already discussed above that the initialization that is then called in line 2 also runs in polynomial time. In lines 3–33, we compute the set *i.dependent-States-IPEI* for each component $i \in Comp$. This set corresponds to the union of the sets $\text{IPEI}(s_i)$ of all dependent local states s_i of the current component i. We compute these sets in lines 5–33 where we check in parallel with the variable *cond-one* whether the first condition of Theorem 6.12 holds for each dependent local state. In line 9, we check wether an initial state of the current component is contained in an NBRS, i.e., we check whether the first part of the first condition of the theorem holds. Since we now have to compute consecutive intersections of sets of interactions, we initialize the set *IPEI* with the set of all interactions in line 10.

Next, we examine all performable interactions with respect to the current local state in lines 11–31, i.e., we start computing the sets $\text{PEI}(s_i, \alpha)$ of all dependent states $s_i \in S_i$ of a component $i \in Comp$ and interactions $\alpha \in Int$ that are performable in s_i, i.e., $\alpha \in Int(s_i)$ holds. We now have to compute the problematic states of all components $j \in \text{compset}(\alpha) \setminus \{i\}$, which we consider in lines 14–27. For each local state s_j of such a component j (cf. lines 16–27) we call Algorithm 6.3 in line 17 to decide whether s_j is problematic, i.e., whether $s_j \in \text{PS}_j(s_i, \alpha)$ holds. If this is the case, we collect the NBRS of such a state s_j in the set *PS-NBRS* that collects the states that we need to check for the second part of the first condition of Theorem 6.12. Note that the final check is performed in line 28 after this collection has been finished.

At this point of the algorithm, we can compute the entry interactions for the pair of states (s_i, s_j). We loop trough all interactions $\beta \in Int$ in lines 19–27 and such a β is a potential entry interaction if component i participates in it and no NBRS action with respect to state s_i is an element of β. This works because the NBRS actions are those actions $a_i \in A_i$ of component i such that $a_i \in \beta \ \wedge \ \text{compset}(\beta) \cap \text{coopset}(s_i) = \emptyset$ (for all β) holds (cf.

Algorithm 6.5 Check of the conditions of Theorem 6.12

REFINED-DEADLOCK-FREEDOM-CONDITION(Sys)

```
1  if not DCWF(Sys) then return FALSE // cf. Algorithm B.6 on page 218
2  NBRS-COMPUTATION(Sys) // cf. Algorithm 6.4 on page 163
3  for each component i ∈ Comp
4     i.dependent-States-IPEI = ∅      // We collect the sets IPEI(s_i) for all
5     for each state s_i ∈ S_i          // dependent states s_i of component i.
6        IPEI = ∅ // for the set IPEI(s_i) = ⋂_{β∈Int(s_i)} PEI(s_i, β)
7        if s_i.dependent
8           cond-one = TRUE // First condition of Theorem 6.12
9           if not s_i.NBRS ∩ S_i^0 == ∅ then cond-one = FALSE
10          IPEI = Int
11          for each interaction α ∈ s_i.Int
12             PEI = ∅ // for PEI(s_i, α) = ⋃_{j∈compset(α)\{i}} ⋃_{s_j∈PS_j(s_i,α)} EI(s_i, s_j)
13             cond-one-interaction = TRUE // Second part the first condition
14             for each component j ∈ α.compset \ {i}
15                PS-NBRS = ∅ // for the set ⋃_{s_j∈PS_j(s_i,α)} NBRS(s_j)
16                for each state s_j ∈ S_j
17                   if IS-PROBLEMATIC(Sys, i, s_i, α, j, s_j) // cf. Algorithm 6.3
18                      if not cond-one then PS-NBRS = PS-NBRS ∪ s_j.NBRS
19                      for each interaction β ∈ Int
20                         if not β ∩ s_i.coop-Actions == ∅ // cf. line 10 of Alg. 6.4
21                            for each (t_i, t_j) ∈ s_i.NBRS ⨯̌ s_j.NBRS
22                               Predecessor = {t_j}
23                               if j ∈ β.compset then Predecessor = t_j.Pre_{β.j}
24                               (S_{{i,j}}, ...) = BEHAVIOR-TRAVERSAL(Sys[{i, j}])
25                               if not (t_i.Pre_{β.i} ⨯̌ Predecessor) ∩ S_{{i,j}} == ∅
26                                  PEI = PEI ∪ {β}
27                                  break
28                if not cond-one and not PS-NBRS ∩ S_j^0 == ∅
29                   cond-one-interaction = FALSE
30             IPEI = IPEI ∩ PEI
31             if not cond-one and cond-one-interaction then cond-one = TRUE
32       if not cond-one then return FALSE
33       i.dependent-States-IPEI = i.dependent-States-IPEI ∪ IPEI
34 Cycle-Comp = CYCLE-COMPONENTS(Sys) // cf. Algorithm B.7 on page 219
35 for each interaction α ∈ Int
36    if |α.compset| ≥ 2
37       contained = TRUE
38       for each component i ∈ α.compset
39          if not α ∈ i.dependent-States-IPEI then contained = FALSE; break
40       for each component i ∈ α.compset ∩ Cycle-Comp
41          if α ∈ i.dependent-States-IPEI then contained = TRUE; break
42       if contained then return FALSE
43 return TRUE
```

Definition 6.10 and line 8 of Algorithm 6.4). Thus, for the other actions a_i holds $a_i \in \beta \ \wedge \ \text{compset}(\beta) \cap \text{coopset}(s_i) \neq \emptyset$ (for all β) as required by the definition of the entry interactions (cf. Definition 6.11). Observe that we already computed these actions and stored them in the set $s_i.\mathit{coop\text{-}Actions}$ (cf. the comment in lines 11–13 of Algorithm 6.4). According to Definition 6.11, we now have to determine whether a reachable state exists in the behavior of the subsystem of both components such that i's action in β is used to enter a state after which only NBRS actions are used in appropriate interactions until the state combination (s_i, s_j) is reached. We carry this determination out in lines 21–27 straightforwardly to Definition 6.11. The computation in line 24 of the reachable global behavior of the interaction system $\mathit{Sys}[\{i,j\}]$ for the reachability check was already discussed for the problematic states (cf. the analysis of Algorithm 6.3 in Section 6.2.4). Here, a hash table should be used to cache results and avoid "double" computations of the behaviors (both, in this computation and in the computation of the problematic states). However, there is no exponential blowup in the computational complexity if we do not cache results. Finally, we can update the set *IPEI* in line 30 with the potential entry interactions for the current state. At this point, we can check in line 32 wether the first condition of Theorem 6.12 holds and answer the check of the whole condition negatively if it is violated by the currently considered dependent states.

Now, after the loop over the components is finished (cf. lines 3–33), we determined for each component the union of the sets $\text{IPEI}(s_i)$ of all dependent local states s_i of the component i and know that the first condition of Theorem 6.12 holds. Thus, we need to consider the second condition as well. First, we compute the cycle components of *Sys* in line 34 by calling Algorithm B.7, which is given in Appendix B and whose runtime is linear in the size of the cooperation graph as we discussed at the end of Section 4.3.2. Second, we treat in lines 35–42 each interaction where at least two components participate in. In lines 38–39, we check whether there is a component such that the currently considered interaction is not contained in the collection of entry interactions of the components. In lines 40–41, we ensure that all cycle components do not contain the interaction in question in their collection of entry interactions that potentially lead to a problematic state. After the check of all interactions, we can finally decide whether the second condition of Theorem 6.12 holds.

This brings us to the question of the overall runtime of Algorithm 6.4. From the detailed discussion above we know that we loop once through all local states of all components, and in each loop we consider at most all interactions and all participating components and their local states. If such a local state is a problematic state, which can be decided in $\mathrm{O}(|\mathit{Int}| + |S_{\max}|^2 \cdot |\mathit{Int}|)$ time where $S_{\max}$ denotes the largest local state space (cf. Algorithm 6.3), we com-

pute the entry interactions: We loop through all interactions and check for all suitable (with respect to Definition 6.11) pairs of NBRS states whether they are reachable in a subsystems of size two, which overall consumes $O(|Int| \cdot |S_{\max}|^2 \cdot |S_{\max}|^2 \cdot |Int|)$ time (the latter two factors correspond to the computation of the global behavior of the subsystem of size two). Thus, since we call the problematic state check and the entry interactions determination at most $O(|Comp|^2 \cdot |S_{\max}|^2 \cdot |Int|^2)$ times, we have an overall polynomial time bound of our approach.

As we already mentioned above, we can speed up parts of the computation if we use a hash table to store already computed sets of reachable states of the relevant subsystems of size two for both, the problematic state and the entry interaction computation, i.e., overall we only need to perform at most $|Comp|^2$ reachability analyses bounded by $O(|S_{\max}|^2)$. In this case, the overall runtime bound is $O(|Comp|^2 \cdot |S_{\max}|^4 \cdot |Int|^4)$. However, the approach is much faster since in typical instances not all components participate in all interactions. We address this observation by an evaluation of our approach with respect to a prototype implementation in Section 6.5.

But first, we take a look at related work in the following section.

6.4 Related Work

First and foremost, as we already mentioned several times above, our approach presented in the previous sections is inspired by an approach of Majster-Cederbaum and Martens [180], which itself is an extension of an earlier approach by the authors [179] and whose formal proofs can be found in the dissertation of Martens [190] and in a journal version [181]. As we pointed out in Chapter 4, Majster-Cederbaum and Martens [180] define a similar graph as our cooperation graph and require for their architectural constraint that this graph is a tree in the graph-theoretical sense. This constraint structures the cooperations among the components and reduces the possibility of waiting situations. Here, we extended this idea and admitted certain cycles in our disjoint circular wait free architectural constraint (cf. Definition 4.6).

The authors make a similar observation regarding problematic states and define a notion called "backward search" that is comparable to our notion of the non-interfering backward reachable set of a component's state. However, for the latter notion and thus also for their following results regarding deadlock-freedom, Majster-Cederbaum and Martens [180] require that an interaction system has strongly exclusive communication—as we discussed in Section 5.2 of Chapter 5. We showed in the referenced section that this property can be

established for a given interaction system in polynomial time—and also find a similar result by Majster-Cederbaum and Martens [180, Lemma 1]—without modifying the global behavior of the interaction system under analysis in a severe way with respect to deadlock-freedom. Furthermore, we here showed how we can drop this requirement completely by comparing more information in our entry interactions (cf. Definition 6.11) than Majster-Cederbaum and Martens [180] in their corresponding entry information called "problematic actions" [180, Definition 9]. We never assumed exclusive communication and thus improve their approach by avoiding a polynomial-time preprocessing step that possibly enlarges the behavior of the components for the verification process. We demonstrate the effect of this avoidance by means of an experimental evaluation in the following section.

Summarizing, for interaction systems which already have strongly exclusive communication and a cooperation graph that is a tree in the graph-theoretical sense, the two approaches coincide, i.e., our Theorem 6.12 and Proposition 1 in the paper by Majster-Cederbaum and Martens [180]. Thus, our result from this chapter is a generalization of all previous approaches that exploit the architecture of interaction systems to establish the property of deadlock-freedom in polynomial time.

We want to mention that Martens [190, Definition 3.4.1] presents a further refinement for the set of problematic states. However, Becker [32, Example 3.5.2] already showed that this refinement cannot be used in interaction systems with a disjoint circular wait free architecture because it depends on a property that can only be found when the architecture is a tree in the graph-theoretical sense. To be more precise, if the cooperation graph is a tree and a deadlock is present in the corresponding interaction system, then for every two consecutive interactions that are part of a circular waiting situation as in Lemma 6.2 it holds that the intersection of the sets of participating components contains at least two components. Note that in the disjoint circular wait free case, we can only show the existence of two such interactions (cf. Lemma 6.5 and Theorem 6.6 respectively). In the tree-like case, one can then conclude that for all components involved in the circular waiting situation, certain problematic states exist. This is unfortunately not the case in situations where a vertex representing a component lies on a simple cycle in the cooperation graph, i.e., if the architecture is disjoint circular wait free but not tree-like (with respect to the cooperation graph).

As we already discussed in Section 5.5, acyclic architectures are also exploited for the verification of deadlock-freedom in the work of Bernardo et al. [39] and Hennicker et al. [137]. Remember that both approaches rely on behavioral equivalences among certain key components in the architecture, i.e., if the

behavior of such a key component is not influenced by the cooperation with the remaining components (which is checked with weak bisimilarity), the question of deadlock-freedom is answerable by only looking at the behavior of the key component. Apart from the fact that such equivalences can be found in many systems and that we also considered a comparable approach in Chapter 5, however also numerous examples with no behavioral equivalences at all exist modeling realistic scenarios that are still verifiable with our approach presented in this chapter, e.g., such an example with respect to the work by Bernardo et al. [39] can be found in [180, Figure 4]. For a more detailed discussion of the two approaches, we refer to Section 5.5.

The work of Brookes and Roscoe [49] considers tree-like networks in the context of CSP [141] restricted to binary communication. For such networks, the authors additionally require that cooperating components (or processes in this case) have at most one cooperation partner in every state. This directly allows to imply the deadlock-freedom of the whole network by an analysis of all cooperating pairs of components. However, for networks without this additional property, a tedious case analysis is required to exclude certain waiting situations among the components. For a more detailed comparison of this approach with respect to interaction systems, we refer to the dissertation of Martens [190, pages 68–69 and pages 93–94].

For interaction systems, also several approaches for proving deadlock-freedom exist, e.g., Bensalem et al. [34–36] worked in the context of BIP [30] (for which interaction systems are a theoretical model). Their approach is based on finding invariants for the components, which must be provided for each property, and for the interactions, which are computed automatically. Unfortunately, according to Bensalem et al. [36], for this computation "there is a risk of explosion, if exhaustiveness of solutions is necessary in the analysis process." Although the authors mitigate this risk by using BDDs, the explosion risk remains and thus, this approach is not guaranteed to be polynomial in the number and size of the components and interactions which is an important property of our approach, e.g., the conditions of Theorem 6.12. But, their approach is applicable in situations where no architectural constraint is present and moreover, it also could benefit from the ideas presented in this chapter (in case the system has a disjoint circular wait free architecture) in the following way. In order to prove deadlock-freedom, a special predicate called DIS is constructed in [36] which characterizes the set of all deadlock situations. Now, if our architecture is present, we can use ideas from this chapter to refine this predicate, i.e., exclude those situations whose global reachability can be refuted by a comparison of entry interactions. Moreover, components for which we find no problematic states can potentially be neglected in the analysis process.

A further approach for interaction systems that is guaranteed to be checkable in polynomial time is presented by Majster-Cederbaum et al. [187]. The authors compute an over-approximation of the set of reachable states of an interaction system's global behavior by projecting it to certain subsystems (similarly to our subsystem construction operator, cf. Definition 3.18) of a fixed size d. Here, parameter d is the size of the set of components that constitute a subsystem and is understood as a parameter that adjusts the quality of the over-approximation: The larger the parameter, the more precise the approximation. But, the approach is exponential in this parameter and thus only feasible for small d. The authors then check a sufficient condition on the over-approximation that excludes a waiting scenario where three components are waiting in a row which could be a part of a circular waiting situation among the components. However, Majster-Cederbaum and Martens [179] point out that their deadlock analysis for tree-like interaction systems already is more powerful for interaction system satisfying their architectural constraint (where all interactions are binary), which thus also holds for our approach in this case. But, it is still an open question how these approaches generally compare to each other, e.g., whether the over-approximation approach applied in interaction systems with a disjoint circular wait free architecture can verify the property of deadlock-freedom but our Theorem 6.12 fails.

The above mentioned over-approximation can be further refined by a technique called "cross-checking" of Majster-Cederbaum and Minnameier [184] that compares subsystems that overlap with respect to the set of constituting components to check the reachability of states. For instance, if a state (s_1, s_2, s_3) is reachable in a certain subsystem consisting of components 1, 2, and 3 but in another subsystem consisting of components 1, 2, and 4 no state (s_1, s_2, s_4) for any s_4 is reachable, then no global state can exist where s_1 and s_2 are part of. Thus, the state (s_1, s_2, s_3) can be excluded from the subsystem, which corresponds to a refinement. For a further discussion of this approach, we refer to the dissertation of Minnameier [205, Section 6.5]. A similar refinement with respect to the transitions is proposed by Semmelrock [241].

6.5 Implementation and Experimental Evaluation

We evaluate our deadlock-detection approach, viz. Theorem 6.12 and its implementation in Algorithm 6.5, with respect to interaction systems that can be parametrized such as the database example $Sys_{\mathrm{DB}}(n)$. We compare a prototype implementation of Algorithm 6.5 (cf. page 165) with an implementation of Algorithm B.2 (cf. page 213) that constructs the global behavior of an interaction system to search for deadlocks as discussed in Section 2.3.1.

Moreover, as mentioned in the previous section, the approach of Majster-Cederbaum and Martens [180] requires that an interaction system has strongly exclusive communication and that its cooperation graph is a tree in the graph-theoretical sense. Here, we also evaluate the first assumption. In Section 5.2 of Chapter 5, we discussed how (strongly) exclusive communication can be enforced in an interaction system as a pre-processing step in polynomial time (cf. Algorithm B.8). However, although the global behavior of the resulting interaction system is isomorphic up to transition relabeling (cf. Lemma 5.6), the local behavior of the components is modified by adding fresh actions and transitions. An interesting question now is how this addition affects the runtime of Algorithm 6.5 with respect to our implementation. We thus transformed each of the following example interaction systems into a version with strongly exclusive communication and also evaluated the new system with respect to our approach in order to make this effect visible—the transformation time is, of course, not counted as verification time. We are interested in this effect since we showed in this chapter that the property of strongly exclusive communication is not necessary for deadlock detection, and we are thus interested in the time savings that our approach allows for.

Since we also evaluate an algorithm which constructs the global behavior, we can use its output to determine the exact size of our examples, i.e., the number of states and transitions. However, due to the size of the corresponding labeled transition systems we only give these numbers in a logarithmic scale.

We shortly address some key aspects of our implementation. All algorithms are implemented in the C++ programming language and they use a common framework that allows to treat interaction systems as data structures from the C++ standard library. We use one (external) library for the representation of labeled transition systems. This library implements the concept of binary decision diagrams that were introduced by Lee [169] to represent switching functions and further developed together with efficient manipulation algorithms by Bryant [55]—to be more precise, he studied reduced ordered binary decision diagrams which are commonly referred to as binary decision diagrams, or BDDs for short, nowadays. The encoding of labeled transition systems as binary decision diagrams was proposed by Burch et al. [57] which, to quote Baier and Katoen [23, Section 6.7.2], was "the milestone for symbolic model checking" and enabled the analysis of large and realistic systems. Here, we use the implementation of binary decision diagrams available in the BuDDy library[1] (version 2.4) as an efficient data structure to store and manipulate the behavior of the components and the global behavior in interaction systems.

[1] `http://buddy.sourceforge.net/`

The algorithms and our framework were compiled using the C++ compiler from the GNU Compiler Collection[2] (g++ (Debian 4.4.5-8) 4.4.5) with the "-O3" optimization switch turned on. All experimental evaluations were executed on a standard personal computer system with an Intel Core 2 Duo E6320 processor with a clock speed of 1.86 GHz and 4 GB of main memory running Debian's[3] precompiled kernel (version 2.6.32-5-amd64) and no further processor- or memory-intensive processes other than our evaluation program.

In the following, we first consider a parametrized extension of the merchandise management example, then evaluate the database example, and finally take a look at a further interaction system modeling a typical banking scenario. In each case, we first formally introduce the interaction system (with appropriate references to earlier sections if we already did so), formally analyze the system with respect to Theorems 6.9 and 6.12, and then apply our experimental evaluation sketched above.

6.5.1 Parametrized Merchandise Management Interaction System

In Chapter 3, we used our running example, the merchandise management system, as an illustration for our composition operator, where we composed the original system Sys_{MMS} with an additional management and customer component to yield the system $Sys^{(3)}_{\text{MMS}}$; more precisely, an interaction system consisting of the three original components and two new components (cf. Figure 3.2 on page 65). Here, we further extend this composition: We add n management components (with attached customer components) to a single storage component. However, we do not specify the system as an n-ary composition operation but directly.

Thus, we define interaction system $Sys_{\text{Para-MMS}}(n)$ where $n \in \mathbb{N} \setminus \{0\}$ as:

$$
\begin{aligned}
Comp &= \bigcup_{1 \leq i \leq n} \{c_i, m_i\} \cup \{s\},\\
A_{c_i} &= \{abort_{c_i}, ask_{c_i}, buy_{c_i}, refund_{c_i}\} \text{ for } 1 \leq i \leq n,\\
A_{m_i} &= \{cancel_{m_i}, deliver_{m_i}, order_{m_i}, pay_{m_i}, print_{m_i}, reimburse_{m_i}, release_{m_i},\\
&\qquad reserve_{m_i}\} \text{ for } 1 \leq i \leq n,\\
A_s &= \{mark_s, unmark_s, ship_s\},\\
Int &= \bigcup_{1 \leq i \leq n} \{\{abort_{c_i}, cancel_{m_i}\}, \{ask_{c_i}, order_{m_i}\}, \{buy_{c_i}, pay_{m_i}\}, \{deliver_{m_i}, ship_s\},\\
&\quad \{print_{m_i}\}, \{refund_{c_i}, reimburse_{m_i}\}, \{release_{m_i}, unmark_s\}, \{reserve_{m_i}, mark_s\}\},
\end{aligned}
$$

[2] http://gcc.gnu.org/
[3] http://www.debian.org/

$$Int_{\text{closed}} = \bigcup_{1 \leq i \leq n} \{\{print_{m_i}\}\}.$$

The behavior of the components is defined as follows. For each management component m_i with $1 \leq i \leq n$ we take the labeled transition system depicted in Figure 2.3 (cf. page 23) but add an index, viz. the variable i, to all transition labels (and states), e.g., the transition $s^0_m \xrightarrow{order_m}_m s^1_m$ becomes $s^0_{m_i} \xrightarrow{order_{m_i}}_{m_i} s^1_{m_i}$. We proceed analogously for the behavior of each customer component c_i, where we use the labeled transition system depicted in Figure 2.4 (a) (cf. page 24). For the storage component s, we use the labeled transition system depicted in Figure 2.4 (b) (cf. page 24) without any relabelings. Observe that the interaction systems Sys_{MMS} and $Sys_{\text{Para-MMS}}(1)$ are identical up to the superscript of the components m/m_1 and c/c_1 respectively.

Now, suppose we want to establish the deadlock-freedom of $Sys_{\text{Para-MMS}}(n)$ in an efficient way, i.e., without computing the global behavior. We apply Theorem 6.9 of this chapter, i.e., we have to compute the problematic states with respect to all components' local states, performable interactions, and cooperation partners. For all $1 \leq i \leq n$, we get (all other sets are empty):

$$\begin{aligned}
&\text{PS}_s(s^1_{m_i}, \{reserve_{m_i}, mark_s\}) = \{s^1_s\},\\
&\text{PS}_s(s^4_{m_i}, \{deliver_{m_i}, ship_s\}) = \text{PS}_s(s^5_{m_i}, \{release_{m_i}, unmark_s\}) = \{s^0_s\},\\
&\text{PS}_{m_i}(s^0_s, \{reserve_{m_i}, mark_s\}) = \{s^4_{m_i}, s^5_{m_i}\}, \text{ and}\\
&\text{PS}_{m_i}(s^1_s, \{deliver_{m_i}, ship_s\}) = \text{PS}_{m_i}(s^1_s, \{release_{m_i}, unmark_s\}) = \{s^1_{m_i}\}.
\end{aligned}$$

However, the non-emptiness of the above sets means that we cannot conclude deadlock-freedom with Theorem 6.9. But, we can refine the problematic state information via reachability analyses, i.e., apply Theorem 6.12. First, we have to compute the NBRSs. For all components $j \in Comp$ and states $s_j \in S_j$, we have $\text{NBRS}(s_j) = \{s_j\}$ except for the five states $s^0_{m_i}, s^2_{m_i}, s^3_{m_i}, s^4_{m_i}, s^5_{m_i}$ of every management component m_i with $1 \leq i \leq n$ where we have: $\text{NBRS}(s^0_{m_i}) = \{s^0_{m_i}, s^3_{m_i}, s^4_{m_i}, s^5_{m_i}, s^6_{m_i}\}$, $\text{NBRS}(s^2_{m_i}) = \{s^1_{m_i}, s^2_{m_i}\}$, $\text{NBRS}(s^4_{m_i}) = \{s^2_{m_i}, s^3_{m_i}, s^4_{m_i}\}$, $\text{NBRS}(s^3_{m_i}) = \{s^3_{m_i}, s^6_{m_i}\}$, and $\text{NBRS}(s^5_{m_i}) = \{s^2_{m_i}, s^3_{m_i}, s^5_{m_i}\}$.

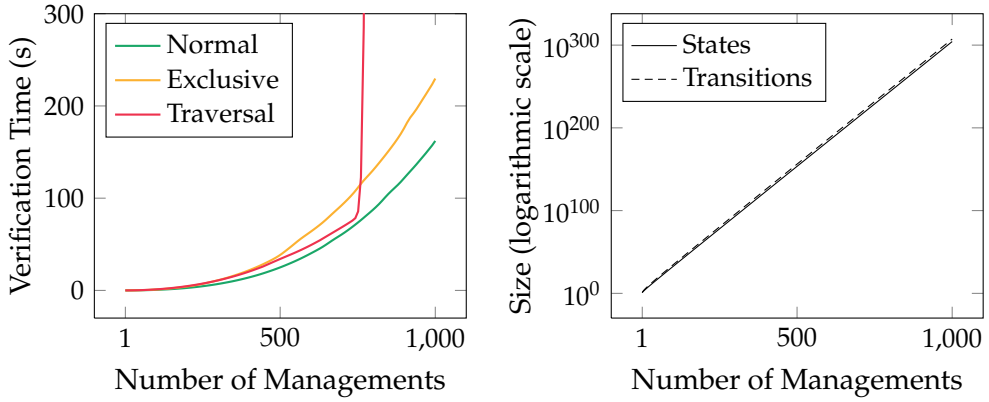
Then, we compute the interesting entry interactions (cf. Definition 6.11) with respect to Theorem 6.12. For all $1 \leq i \leq n$, we get (all other sets are empty):

$$\begin{aligned}
&\text{PEI}(s^0_s, \{reserve_{m_i}, mark_s\})\\
&\qquad = \textstyle\bigcup_{1 \leq j \leq n, j \neq i} \big\{\{deliver_{m_j}, ship_s\}, \{release_{m_j}, unmark_s\}\big\},\\
&\text{PEI}(s^1_s, \{deliver_{m_i}, ship_s\}) = \text{PEI}(s^1_s, \{release_{m_i}, unmark_s\})\\
&\qquad = \textstyle\bigcup_{1 \leq j \leq n, j \neq i} \big\{\{reserve_{m_j}, mark_s\}\big\}, \text{ and}\\
&\text{PEI}(s^1_{m_i}, \{reserve_{m_i}, mark_s\}) = \big\{\{ask_{c_i}, order_{m_i}\}\big\}.
\end{aligned}$$

For all components $j \in \mathit{Comp}$ and states $s_j \in S_j$, we have $\mathrm{IPEI}(s_j) = \emptyset$, i.e., the second condition of Theorem 6.12 holds. Now, also the first condition holds since for all dependent local states s_j of all components $j \in \mathit{Comp}$ holds $\mathrm{NBRS}(s_j) \cap S_j^0 = \emptyset$—except for the initial states of all components, viz. s_s^0, $s_{m_i}^0$, and $s_{c_i}^0$ for $1 \leq i \leq n$, for which we have to show that the second term of the first condition holds, i.e., for all these states s_j, we have to find an interaction $\alpha \in \mathit{Int}(s_j)$ such that for all components $k \in \mathrm{compset}(\alpha) \setminus \{j\}$ holds $\left(\bigcup_{s_k \in \mathrm{PS}_k(s_j,\alpha)} \mathrm{NBRS}(s_k)\right) \cap S_k^0 = \emptyset$.

For state s_s^0, we can pick interaction $\{\mathit{reserve}_{m_i}, \mathit{mark}_s\}$ for any $1 \leq i \leq n$ because $\{\mathit{reserve}_{m_i}, \mathit{mark}_s\} \in \mathit{Int}(s_s^0)$, $\mathrm{compset}(\{\mathit{reserve}_{m_i}, \mathit{mark}_s\}) \setminus \{s\} = \{m_i\}$, and we have $\mathrm{PS}_{m_i}(s_s^0, \{\mathit{reserve}_{m_i}, \mathit{mark}_s\}) = \{s_{m_i}^4, s_{m_i}^5\}$ but both $\mathrm{NBRS}(s_{m_i}^4) \cap S_{m_i}^0 = \emptyset$ and $\mathrm{NBRS}(s_{m_i}^5) \cap S_{m_i}^0 = \emptyset$. Observe that for the states $s_{m_i}^0$ and $s_{c_i}^0$ with $1 \leq i \leq n$, all problematic state sets are empty, i.e., $\mathrm{PS}_k(s_{m_i}^0, \alpha) = \emptyset$ and $\mathrm{PS}_k(s_{c_i}^0, \alpha) = \emptyset$ for all suitable components k and interactions α. Thus, the first condition holds for all these states.

Now, we can conclude the deadlock-freedom of $\mathit{Sys}_{\text{Para-MMS}}(n)$ for all $n \in \mathbb{N} \setminus \{0\}$ because both conditions of Theorem 6.12 hold. Next, we take a look at the experimental evaluation of this system.



(a) Analysis of $\mathit{Sys}_{\text{Para-MMS}}(n)$, $1 \leq n \leq 1000$ **(b)** Size of $\mathit{Sys}_{\text{Para-MMS}}(n)$ for $1 \leq n \leq 1000$

Figure 6.2: Evaluation of the parametrized merchandise management system

Figure 6.2 depicts the experimental evaluation of $\mathit{Sys}_{\text{Para-MMS}}(n)$ for parameter n ranging from 1 to 1000. From the comparison of the three runtimes in Figure 6.2 (a), we see that our conditional approach without requiring strongly exclusive communication is the fastest. Observe that around $n = 770$, our evaluation system gets low on memory to compute and store the global behavior as a binary decision diagram. The global state space of $[\![\mathit{Sys}_{\text{Para-MMS}}(770)]\!]$ has approximately the size of 10^{235} states which demonstrates the power of

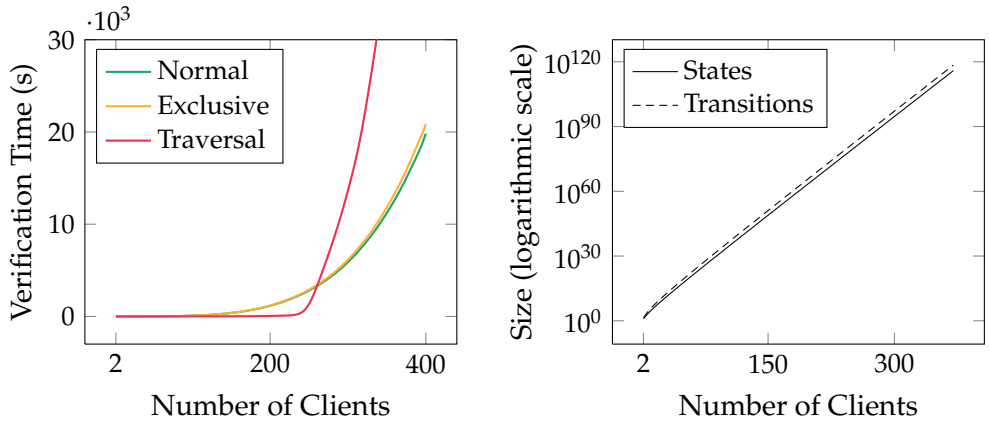
the symbolic representation, i.e., we cannot store such a huge number with an explicit representation of the states.

Nevertheless, our approach can verify the deadlock-freedom of the interaction system $Sys_{\text{Para-MMS}}(1000)$ in under 3 minutes where the global state space analysis needs approximately 2.86 hours. Note that the corresponding labeled transition system consists of more than 10^{304} states and more than 10^{307} transitions (cf. Figure 6.2 (b)).

We can also conclude that the property of strongly exclusive communication clearly influences the verification time, although the differences for $n = 1000$ are small, i.e., the non-exclusive version completed in 162.07 seconds whereas the strongly exclusive one needed 229.7 seconds. However, a trend is observable that this difference gets bigger for larger systems.

6.5.2 Database Interaction System

We already discussed our conditions for deadlock-freedom with respect to $Sys_{\text{DB}}(n)$ ($n \in \mathbb{N} \setminus \{0\}$): In Section 6.2.3 we treated Theorem 6.9, which failed, and in Section 6.3.2 we applied Theorem 6.12, which succeeded, i.e., $Sys_{\text{DB}}(n)$ is deadlock-free. Now, we also evaluate this system. Figure 6.3 depicts the verification time and global behavior size for $Sys_{\text{DB}}(n)$ for various numbers of clients, i.e., the parameter n, where we set the parameter to $w_i = i$.



(a) Analysis of $Sys_{\text{DB}}(n)$ for $2 \leq n \leq 400$ **(b)** Size of $Sys_{\text{DB}}(n)$ for $2 \leq n \leq 370$

Figure 6.3: Evaluation of the database system

Observe that we stopped to evaluate our global-behavior-based deadlock detection approach for $n = 370$ (which can be seen in Figure 6.3 (b) where the size is only plotted up to $n = 370$). For this system, the evolution already

took more than 24 hours, where our approach from this chapter completed in under 4 hours. For $n = 400$, the deadlock-freedom was established in 5.5 hours.

If we compare the size of the global behavior of $Sys_{\text{DB}}(n)$ as depicted in Figure 6.3 (b) with the size of the global behavior of the previous example $Sys_{\text{Para-MMS}}(n)$ (cf. Figure 6.2 (b)) there seems to be a curiosity: We can compute a labeled transition system of more than 10^{300} states in case of the latter system, but give up for $Sys_{\text{DB}}(n)$ for $n = 370$ although the corresponding global behavior only consists of approximately 10^{115} states which is several orders of magnitude smaller. The answer to this curiosity lies in the symbolic representation, i.e., the corresponding binary decision diagrams. It is well known that we can store more information symbolically, the better the variable ordering, e.g., consider the example given by Baier and Katoen [23, Example 6.73] which shows a linear- and an exponential-size version of a binary decision diagram because of different variable orderings. However, here we do not deepen the discussion of this issue since the variable ordering in the specification, i.e., the binary decision diagrams for the behavior of the components, is the same in all cases and not optimized with respect to a certain approach. But, we have to keep this phenomenon is mind if we compare our results with the literature—which we leave as future work.

Finally, Figure 6.3 (a) shows a slight performance increase if we do not transform the system into one with strongly exclusive communication beforehand. For instance for 400 clients, the difference between the non-exclusive, called "normal" in the figure, and the exclusive version amounts to approximately 16 minutes. Note that the database component d is the only component that needs adjustment. As for the previous example, this shows that the exclusive communication factor should not be underestimated.

6.5.3 Banking Interaction System

We consider an example that models a typical banking scenario: A clearing company interacts with n banks and each bank interacts with m automated teller machines (ATMs). This example was introduced by Baumeister et al. [31] and modeled as an interaction system by Majster-Cederbaum and Martens [179]. As usual, the banks operate the ATMs which receive requests by customers that consist of an inserted bank card and an associated personal identification number (PIN). The verification of the account information is then provided by a clearing company, which is the same for all banks. In case the verification succeeded, the customer can debit his account, i.e., the bank disburses the money, or cancel the operation. Otherwise, the customer is

informed about the wrong credentials; however, the number of trials is not limited.

Here, we define interaction system $Sys_{\text{Banks}}(n, m)$ where $n, m \in \mathbb{N} \setminus \{0\}$ as:

$$Comp = \{cc, b_1, \ldots, b_n\} \cup \bigcup_{1 \leq i \leq n} \{a_i^1, \ldots, a_i^m\},$$

$$A_{cc} = \{check_{cc}, correct_{cc}, incorrect_{cc}\},$$

$$A_{b_i} = \{cancel_{b_i}, check_{b_i}, correct_{b_i}, disburse_{b_i}, incorrect_{b_i}, request_{b_i}, rightpin_{b_i}, wrongpin_{b_i}\} \text{ for } 1 \leq i \leq n,$$

$$A_{a_i^j} = \{cancel_{a_i^j}, request_{a_i^j}, rightpin_{a_i^j}, takemoney_{a_i^j}, wrongpin_{a_i^j}\} \text{ for } 1 \leq i \leq n \text{ and } 1 \leq j \leq m,$$

$$Int = \bigcup_{1 \leq i \leq n} \{\{check_{cc}, check_{b_i}\}, \{correct_{cc}, correct_{b_i}\}, \{incorrect_{cc}, incorrect_{b_i}\}\} \cup \bigcup_{1 \leq i \leq n, 1 \leq j \leq m} \{\{request_{b_i}, request_{a_i^j}\}, \{rightpin_{b_i}, rightpin_{a_i^j}\}, \{cancel_{b_i}, cancel_{a_i^j}\}, \{wrongpin_{b_i}, wrongpin_{a_i^j}\}, \{disburse_{b_i}, takemoney_{a_i^j}\}\}, \text{ and}$$

$$Int_{\text{closed}} = \{\}.$$

Figure 6.4 depicts the interaction model for one bank and one ATM, i.e., $n = 1$ and $m = 1$.

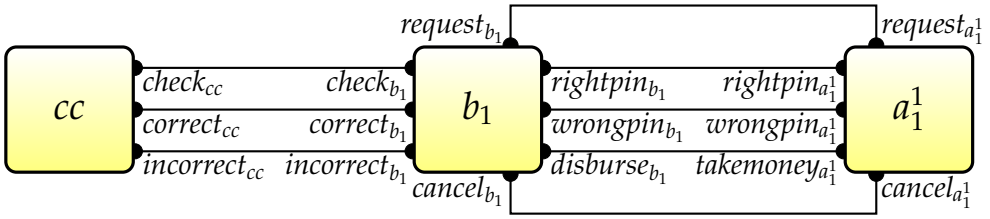
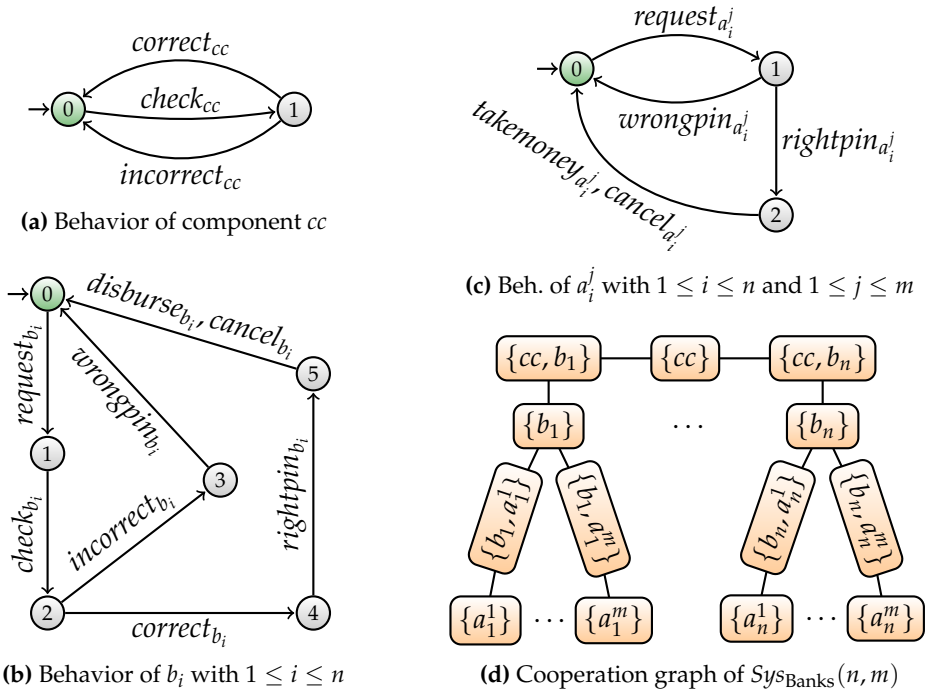


Figure 6.4: Interaction model of the banking example for $n = 1$ and $m = 1$

The behavioral model is depicted in Figure 6.5 on the following page. The behavior of the clearing company component *cc* is depicted in Figure 6.5 (a) together with the behavior of each bank component b_i for $1 \leq i \leq n$ in Figure 6.5 (b) and the behavior each ATM component a_i^j for $1 \leq i \leq n$ and $1 \leq j \leq m$ in Figure 6.5 (c).

The cooperation graph of $Sys_{\text{Banks}}(n, m)$ is depicted in Figure 6.5 (d) which shows that the system has a tree-like architecture and hence also a disjoint circular wait free one. Please note that the nested structure of the cooperation graph already indicates that many cooperations are not exclusive, i.e., the transformation of the system requires adjustments of many more components than of $Sys_{\text{DB}}(n)$.



(a) Behavior of component cc

(b) Behavior of b_i with $1 \leq i \leq n$

(c) Beh. of a_i^j with $1 \leq i \leq n$ and $1 \leq j \leq m$

(d) Cooperation graph of $Sys_{\text{Banks}}(n, m)$

Figure 6.5: Behavioral model and cooperation graph of $Sys_{\text{Banks}}(n, m)$

We analyze the banking example as well. We want to verify the deadlock-freedom of $Sys_{\text{Banks}}(n, m)$ and start by applying Theorem 6.9, i.e., we check whether problematic states exist. For all $1 \leq i \leq n$ and $1 \leq j \leq m$, we get (all other combinations are empty):

$$
\begin{aligned}
&\text{PS}_{cc}(s^1_{b_i}, \{check_{cc}, check_{b_i}\}) = \{s^1_{cc}\},\\
&\text{PS}_{cc}(s^2_{b_i}, \{correct_{cc}, correct_{b_i}\}) = \text{PS}_{cc}(s^2_{b_i}, \{incorrect_{cc}, incorrect_{b_i}\}) = \{s^0_{cc}\},\\
&\text{PS}_{b_i}(s^0_{cc}, \{check_{cc}, check_{b_i}\}) = \{s^2_{b_i}\},\\
&\text{PS}_{b_i}(s^1_{cc}, \{correct_{cc}, correct_{b_i}\}) = \text{PS}_{b_i}(s^1_{c}, \{incorrect_{cc}, incorrect_{b_i}\}) = \{s^1_{b_i}\},\\
&\text{PS}_{b_i}(s^0_{a_i^j}, \{request_{b_i}, request_{a_i^j}\}) = \{s^3_{b_i}, s^4_{b_i}, s^5_{b_i}\},\\
&\text{PS}_{b_i}(s^1_{a_i^j}, \{wrongpin_{b_i}, wrongpin_{a_i^j}\}) = \text{PS}_{b_i}(s^1_{a_i^j}, \{rightpin_{b_i}, rightpin_{a_i^j}\}) = \{s^0_{b_i}, s^5_{b_i}\},\\
&\text{PS}_{b_i}(s^2_{a_i^j}, \{cancel_{b_i}, cancel_{a_i^j}\}) = \text{PS}_{b_i}(s^2_{a_i^j}, \{disburse_{b_i}, takemoney_{a_i^j}\}) = \{s^0_{b_i}, s^3_{b_i}, s^4_{b_i}\},\\
&\text{PS}_{a_i^j}(s^0_{b_i}, \{request_{b_i}, request_{a_i^j}\}) = \{s^1_{a_i^j}, s^2_{a_i^j}\},\\
&\text{PS}_{a_i^j}(s^3_{b_i}, \{wrongpin_{b_i}, wrongpin_{a_i^j}\}) = \text{PS}_{a_i^j}(s^4_{b_i}, \{rightpin_{b_i}, rightpin_{a_i^j}\}) = \{s^0_{a_i^j}, s^2_{a_i^j}\},\\
&\text{PS}_{a_i^j}(s^5_{b_i}, \{cancel_{b_i}, cancel_{a_i^j}\}) = \text{PS}_{a_i^j}(s^5_{b_i}, \{disburse_{b_i}, takemoney_{a_i^j}\}) = \{s^0_{a_i^j}, s^1_{a_i^j}\}.
\end{aligned}
$$

We use one of these states to illustrate the intuition behind problematic states. Consider a bank b_i and its state $s^3_{b_i}$. In this state, the bank wants to cooperate with one of its associated ATMs, i.e., we have $\text{coopset}(s^3_{b_i}) = \{a_i^j \mid 1 \leq$

$j \leq m\}$. We fix one ATM a_i^j and since b_i wants to perform the interaction $\alpha = \{wrongpin_{b_i}, wrongpin_{a_i^j}\}$, we take a look at the set $\mathrm{PS}_{a_i^j}(s^3_{b_i}, \alpha)$, i.e., we ask which states of the ATM are problematic for the bank in state $s^3_{b_i}$ and α. Now, the only state that is not problematic is $s^1_{a_i^j}$ since we have $\alpha \in Int(s^1_{a_i^j})$. For the other two states, the ATM wants to cooperate with the bank, i.e., $b_i \in \mathrm{coopset}(s^k_{a_i^j})$ for $k \in \{0, 2\}$ and thus, we need to check their reachability in the subsystem consisting of the bank and the ATM component and whether an interaction β with $\mathrm{compset}(\beta) = \{b_i, a_i^j\}$ is enabled. Here, we get:

$$\mathrm{PS}_{a_i^j}(s^3_{b_i}, \{wrongpin_{b_i}, wrongpin_{a_i^j}\}) = \{s^0_{a_i^j}, s^2_{a_i^j}\}.$$

Thus, we cannot conclude deadlock-freedom and therefore refine the information by considering the reachability of problematic state combinations. First, we compute the NBRSs: For all components $k \in Comp$ and states $s_k \in S_k$, we have $\mathrm{NBRS}(s_k) = \{s_k\}$ except for the three states $s^1_{b_i}, s^3_{b_i}, s^4_{b_i}$ of every bank component b_i with $1 \leq i \leq n$ where we have:

$$\mathrm{NBRS}(s^1_{b_i}) = \{s^1_{b_i}, s^0_{b_i}, s^3_{b_i}, s^5_{b_i}, s^4_{b_i}\}, \qquad \mathrm{NBRS}(s^3_{b_i}) = \{s^3_{b_i}, s^2_{b_i}, s^1_{b_i}\},$$
$$\mathrm{NBRS}(s^4_{b_i}) = \{s^4_{b_i}, s^2_{b_i}, s^1_{b_i}\}.$$

Again, we consider state $s^3_{b_i}$ of a bank b_i: The states $s^1_{b_i}$ and $s^2_{b_i}$ are contained in the non-interfering backward reachable set because in each of these states, the bank only wants to cooperate with the clearing company component cc. Thus, the cooperation partners of b_i in state $s^3_{b_i}$, which are the associated ATMs, cannot influence that state $s^3_{b_i}$ is reached once state $s^1_{b_i}$ or $s^2_{b_i}$ is entered.

Then, we apply Theorem 6.12 and compute the entry interactions with respect to problematic state combinations. For all $1 \leq i \leq n$ and $1 \leq j \leq m$, we get (all other sets are empty):

$$\begin{aligned}
&\mathrm{PEI}(s^0_{cc}, \{check_{cc}, check_{b_i}\}) \\
&\qquad = \textstyle\bigcup_{1 \leq k \leq n, k \neq i} \{\{correct_{cc}, correct_{b_k}\}, \{incorrect_{cc}, incorrect_{b_k}\}\}, \\
&\mathrm{PEI}(s^1_{cc}, \{correct_{cc}, correct_{b_i}\}) = \mathrm{PEI}(s^1_{cc}, \{incorrect_{cc}, incorrect_{b_i}\}) \\
&\qquad = \textstyle\bigcup_{1 \leq k \leq n, k \neq i} \{\{check_{cc}, check_{b_k}\}\}, \\
&\mathrm{PEI}(s^0_{b_i}, \{request_{b_i}, request_{a_i^j}\}) = \textstyle\bigcup_{1 \leq k \leq m, k \neq j} \{\{wrongpin_{b_i}, wrongpin_{a_i^k}\}, \\
&\qquad\qquad \{cancel_{b_i}, cancel_{a_i^k}\}, \{disburse_{b_i}, takemoney_{a_i^k}\}\}, \\
&\mathrm{PEI}(s^3_{b_i}, \{wrongpin_{b_i}, wrongpin_{a_i^j}\}) = \mathrm{PEI}(s^4_{b_i}, \{rightpin_{b_i}, rightpin_{a_i^j}\}) \\
&\qquad = \textstyle\bigcup_{1 \leq k \leq m, k \neq j} \{\{request_{b_i}, request_{a_i^k}\}\}, \text{ and} \\
&\mathrm{PEI}(s^5_{b_i}, \{cancel_{b_i}, cancel_{a_i^j}\}) = \mathrm{PEI}(s^5_{b_i}, \{disburse_{b_i}, takemoney_{a_i^j}\}) \\
&\qquad = \textstyle\bigcup_{1 \leq k \leq m, k \neq j} \{\{rightpin_{b_i}, rightpin_{a_i^k}\}\}.
\end{aligned}$$

As an example, consider a bank b_i in state $s^3_{b_i}$, one of its associated ATMs a^j_i in state $s^0_{a^j_i}$, and interaction $\alpha = \{wrongpin_{b_i}, wrongpin_{a^j_i}\}$. We learned above that $s^0_{a^j_i} \in \mathrm{PS}_{a^j_i}(s^3_{b_i}, \alpha)$ holds and thus compute the entry interactions of these states. Here, we have $\mathrm{EI}(s^3_{b_i}, s^0_{a^j_i}) = \bigcup_{1 \leq k \leq m, k \neq j} \{\{request_{b_i}, request_{a^k_i}\}\}$ because if we consider the relevant NBRS combinations, which are $(s^2_{b_i}, s^0_{a^j_i})$ and $(s^1_{b_i}, s^0_{a^j_i})$, we can observe that the state $(s^1_{b_i}, s^0_{a^j_i})$ can be reached in the behavior of the subsystem consisting of the two components, viz. $[\![Sys_{\mathrm{Banks}}(n,m)[\{b_i, a^j_i\}]]\!]$, by executing the (partial) interaction $\{request_{b_i}\}$ which is used for cooperation with one of the other associated ATMs. Since each of these possibly offers the corresponding request action, we get the union of all these request interactions as the entry interactions of state $s^3_{b_i}$ and $s^0_{a^j_i}$. Observe that the interaction $\{request_{b_i}, request_{a^j_i}\}$ is not part of the entry interactions since its execution does not lead to a state combination where ATM a^j_i stays in its initial state, which is also the explanation why there is no interaction that lies in all intersections of the problematic entry interactions as we see next.

For all components $k \in Comp$ and states $s_k \in S_k$, we have $\mathrm{IPEI}(s_k) = \emptyset$, i.e., the second condition of Theorem 6.12 holds. Again, considering a particular bank b_i, the state $s^3_{b_i}$ is only a potential deadlock if all associated ATMs are in a problematic state. For instance, if $m = 2$, then the state combination $(s^3_{b_i}, s^0_{a^1_i}, s^0_{a^2_i})$ can only be part of a reachable global state if we have:

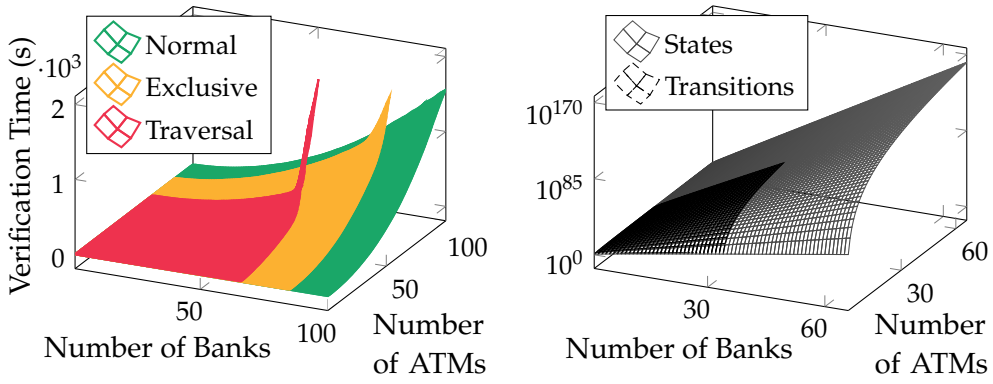
$$\mathrm{EI}(s^3_{b_i}, s^0_{a^1_i}) \cap \mathrm{EI}(s^3_{b_i}, s^0_{a^2_i}) \neq \emptyset.$$

We learned above that at least one of the ATMs has to cooperate with the bank b_i such that it can get to its local state $s^3_{b_i}$ and that no such common entry interaction exists. Here, this corresponds to the intuition about the banking system since either the state combination $(s^3_{b_i}, s^1_{a^1_i}, s^0_{a^2_i})$ or $(s^3_{b_i}, s^0_{a^1_i}, s^1_{a^2_i})$ can be reached after the execution of one of the request interactions.

As a further example, observe that if the clearing company cc wants to inform a bank b_i that the provided PIN is correct, i.e., cc wants to perform the interaction $\{correct_{cc}, correct_{b_i}\}$ in state s^1_{cc}, then a previously performed initial cooperation (a check interaction) with any other bank b_k with $i \neq k$ is problematic since the bank b_i could then not be ready to be informed anymore. But, since the check interaction for b_i is not problematic in this case, no such waiting situation can occur—which is shown by the emptiness of the intersections.

We come back to Theorem 6.12. We learned above that the second condition holds and now, also the first one holds since for all dependent local states s_k of all components $k \in Comp$ holds $\mathrm{NBRS}(s_k) \cap S^0_k = \emptyset$, except for the initial states of the components and state $s^1_{b_i}$ of every bank b_i but there the corresponding problematic state sets are empty. Thus, we can conclude the deadlock-freedom of $Sys_{\mathrm{Banks}}(n,m)$.

Next, we take a look at the experimental evaluation of this system. Figure 6.6 depicts our evaluation for different numbers of banks and ATMs.



(a) Analysis of $Sys_{\text{Banks}}(n, m)$, $1 \leq n, m \leq 100$ **(b)** Size of $Sys_{\text{Banks}}(n, m)$ for $1 \leq n, m \leq 66$

Figure 6.6: Evaluation of the banking system

Observe that the largest system we evaluated is $Sys_{\text{Banks}}(100, 100)$ whose deadlock-freedom was established in 1647.98 seconds (approximately 27.5 minutes). For the direct deadlock analysis, we already gave up for $n = 66$ and $m = 66$, where we needed 2146.4 seconds (approximately 35 minutes), because the trend of the verification time observable in Figure 6.6 (a) makes clear that larger systems need more and more time because our evaluation system starts to be running out of memory. With the approach of this chapter, we established the deadlock-freedom of $Sys_{\text{Banks}}(66, 66)$ in 262.08 seconds (approximately 4.4 minutes). This supports our statement in the introduction of this thesis that a system with ten thousand components is hard to analyze directly because $Sys_{\text{Banks}}(100, 100)$ consists of 10 101 components.

Finally, the exclusive communication factor is better visible since the verification of the systems with strongly exclusive communication requires much more time. For instance, for $n = 85$ banks and $m = 85$ ATMs for each bank, which yields an interaction system with 7311 components, the difference in the verification times amounts to approximately 14 minutes (for the exclusive case, this corresponds to the yellow peak visible in Figure 6.6 (a)). This lets us conclude that also from a performance point of view, the exclusive communication factor should not be underestimated and it is beneficial to have a technique such as our entry interactions to completely circumvent this requirement.

This ends our discussion of the experimental evaluation. In the next section, we summarize the chapter and address future work.

6.6 Summary and Future Work

In this chapter, we showed how component-based systems with multiway cooperation can be analyzed on the basis of an architectural constraint that goes beyond common acyclicity requirements. The presented analysis focuses on the property of deadlock-freedom of interaction systems and provides a polynomial-time checkable condition that ensures deadlock-freedom by exploiting a restriction of the architecture called disjoint circular wait freedom. Roughly speaking, this architectural constraint disallows any circular waiting situations among the components such that the reason of one waiting is independent from any other one.

We want to point out that we only derived a *sufficient condition* for deadlock-freedom. For instance, our approach fails in situations where a set of components blocks each other but other components not involved in this blocking are able to proceed globally. But, it should be clear that by only considering sets of components of size two—which yields a very efficient approach—not all such situations can be covered.

On the other hand, if our approach fails, the information provided by the entry interactions gives a hint of which components are involved in a potential deadlock. With this information, a software engineer can take a closer look at this potentially small set of components and either resolve the reason manually or encapsulate this set in a new composite component that has equivalent behavior, is verified deadlock-free with another technique, and now causes no problems in the remaining system.

Our approach can also be used as a design pattern to ensure that a system is correct by construction. If a software engineer sticks to the composition rule imposed by our architectural constraint, a subsequent application of our condition after each composition step facilitates a correct system design in an automatic and convenient way.

Moreover, our approach can be extended in the following direction. If we consider subsystems of fixed size $k > 2$ in the sufficient condition, we could obtain more information regarding the reachability of state combinations. Potentially, we then could, as already mentioned in Section 4.7 of Chapter 4, also admit simple cycles in the cooperation graph where up to k vertices that represent components lie on and derive a k-disjoint circular wait free architecture.

However, we leave these ideas for future work and instead further refine the model of interaction systems in the next chapter to allow for a gray-box view of the components.

Chapter 7

Gray-Box View and Protocols

In the previous chapters, we learned that restrictions on the architecture of interaction systems and the behavior of components (by requiring that certain equivalences hold) allow for property verification such as deadlock-freedom (cf. Chapter 6) or CTL*–X formulae (cf. Chapter 5) without exhaustively searching the global state space, which yields efficient approaches from a computational viewpoint.

A drawback of the original model of interaction systems (as specified in Chapter 2) is that the entity used for cooperation among the components, viz. the ports in the terminology of component-based development, is considered as a single action and thus no additional behavior can be specified for it, e.g., for any behavior-related questions we have to analyze the local component behavior. Additionally, if the local behavior of a component should not be fully disclosed, the natural question arises what kind of information needs to be disclosed by the component in order to still be able to verify properties. Up to now, we required the disclosure of the full behavior, which is usually called a white-box view of the components. The other extreme is a black-box view in which no behavior is accessible, i.e., the components only disclose their set of actions. Clearly, this is not useful in order to verify properties like deadlock-freedom, and we thus have to find a way in between these extremes: A so-called gray-box view.

We employ the following idea. We extend the model of interaction systems to also capture port behavior. We group several actions of one component and call this group a *port alphabet*, i.e., the set of actions is partitioned into several port alphabets. Additionally, every port is equipped with a labeled transition system over the port alphabet which we call the *port protocol*.

The idea behind this approach is that in verification steps we only want to

use the port protocols of involved components instead of their local behavior. This is more efficient since the behavior of the component is typically much larger (if we compare the number of states and transitions) than its port protocols. The verification of properties for the whole component and system respectively should then follow from the verification step that used only the port protocols.

Of course, this raises several questions. If we use a port protocol of a component in a verification step, its underlying labeled transition system has to be related to the behavior of the component or resemble parts of it. This relationship should also preserve properties, e.g., the satisfaction of the same logical formulae. Another question is how to obtain such port protocols, i.e., are they derived in an automatic manner or explicitly given by the component's author. In a typical scenario, a component processes data of one port internally and outputs it over another port. Obviously, the latter port must wait for the availability of the internal data. This step has to be incorporated in the port protocol, e.g., we could assume that this data is always available in verification steps. A certain unobservable action in the port protocol could model the exchange if this waiting influences the future behavior of the port. This connection of a port protocol's action (or a sequence of them) to the component's internal actions (or actions of another port) is also interesting for the first question, i.e., how are a component and a port related.

Another point for the introduction of such additional behavior is the paradigm of information hiding: Only relevant software parts should be accessible from outside. A component may be given only as a black box, i.e., the internal behavior is unknown in the development process, but the author provides the port protocols and guarantees that any property which is verifiable with the port protocols is also satisfied by the whole component. As mentioned above, this supports a gray-box view of the components that is desired in component-based development and related to information hiding [54].

In the following, we address the above mentioned questions and show how to answer them with a modified version of interaction systems which we call *protocol interaction system*.

7.1 Formalization of Protocol Interaction Systems

We formalize the idea for protocol interaction systems and port protocols as mentioned in the introduction to this chapter. We take the same route as in the introduction of interaction systems in Chapter 2, i.e., we first define a *protocol component system*.

Definition 7.1 (Protocol Component System): A *protocol component system PCS* is defined as a tuple $(Comp, \{P_i\}_{i \in Comp}, \{A_i^p\}_{i \in Comp \wedge p \in P_i})$ where *Comp* is a finite set of components, which are referred to as $i \in Comp$. The available *ports* of a component i are given by the finite set P_i, and the mapping $\text{ports}(i) := \{i{:}p \mid p \in P_i\}$ allows to refer to a port p of i as $i{:}p \in \text{ports}(i)$. The actions of each port $i{:}p$ are given by the finite set A_i^p, also denoted by *port alphabet* $A_{i:p}$, and are assumed to be disjoint, i.e., $\forall\, i, j \in Comp \;\; \forall\, p \in P_i \;\; \forall\, q \in P_j \colon i \neq j \;\vee\; p \neq q \implies A_i^p \cap A_j^q = \emptyset$. Similarly to a component system, all actions of a component i are contained in its action set $A_i := \bigcup_{p \in P_i} A_i^p$, and the union of all action sets is called the global action set $Act := \bigcup_{i \in Comp} A_i$. For convenience, we assume that $\tau \notin Act$ holds for a valid protocol component system.

Since we defined an action set for each component and the global action set above, we can now define interaction models for protocol component systems analogously to the definition for component systems. In the definition of an interaction (cf. Definition 2.2) and an interaction model (cf. Definition 2.3), we only need to substitute the term "component system" by "protocol component system". Thus, instead of defining protocol interaction models, we use Definitions 2.2 and 2.3 for protocol component systems. Similarly as for interactions and components, we put $i{:}p(\alpha) := A_{i:p} \cap \alpha$ for an interaction α and a port $i{:}p$ of a component i in a given interaction model based on a protocol component system and say that $i{:}p$ *participates in* α if $i{:}p(\alpha) \neq \emptyset$ holds.

Please note that all actions are assigned to a port in Definition 7.1. If some of these actions should be internal, they can be assigned to a special internal port, which we omitted here. Next, we define protocol interaction systems.

Definition 7.2 (Protocol Interaction System): Let *IM* be an interaction model based on a protocol component system *PCS*. A *protocol interaction system Sys* is defined as a tuple $(IM, \{LTS_i\}_{i \in Comp}, \{LTS_{i:p}\}_{i \in Comp \wedge i:p \in \text{ports}(i)})$. The *component behavioral model* $\{LTS_i\}_{i \in Comp}$ is a family of labeled transition systems with $LTS_i = (S_i, A_i, \{\xrightarrow{a}_i\}_{a \in A_i}, S_i^0)$, i.e., for each component exists a labeled transition system over its action set. Similarly, the *port behavioral model* $\{LTS_{i:p}\}_{i \in Comp \wedge i:p \in \text{ports}(i)}$ is a family of labeled transition systems with $LTS_{i:p} = (S_{i:p}, A_{i:p} \cup \{\tau\}, \{\xrightarrow{a}_{i:p}\}_{a \in A_{i:p} \cup \{\tau\}}, S_{i:p}^0)$, i.e., for each port exists a labeled transition system over its port alphabet (where τ-transitions are allowed) which we call *port protocol*. For convenience, we write $[\![i]\!]$ instead of LTS_i and $[\![i{:}p]\!]$ instead of $LTS_{i:p}$. Further, we assume that the components' and the port protocols' sets of states are disjoint, i.e., $\forall\, i, j \in Comp \colon i \neq j \implies S_i \cap S_j = \emptyset$ and $\forall\, i, j \in Comp \;\; \forall\, i{:}p \in \text{ports}(i) \;\; \forall\, j{:}q \in \text{ports}(j) \colon i \neq j \;\vee\; i{:}p \neq j{:}q \implies S_{i:p} \cap S_{j:q} = \emptyset$, and that all sets of states are nonempty, i.e., $\forall\, i \in Comp \colon |S_i| \geq |S_i^0| > 0$ and $\forall\, i \in Comp \;\; \forall\, i{:}p \in \text{ports}(i) \colon |S_{i:p}| \geq |S_{i:p}^0| > 0$ holds.

Observe that the special symbol τ is allowed as a transition label in the port protocols but not in the component behavior—we required that no action set contains τ in the definition of a protocol component system (cf. Definition 7.1).

We give an example to demonstrate protocol interaction systems. We use a simpler version of the merchandise management example Sys_{MMS}. We modify the original system such that the components do not cover the cancel/refund operations, i.e., we assume that all orders are successful, and such that no direct deliveries are offered. This leads to the following protocol component system where each component provides ports with respect to its cooperation partners. Please note that for the specification of the ports for a component i, we use the $\text{ports}(i)$ notation instead of the P_i notation. We set:

$$Comp = \{c, m, s\},$$
$$\text{ports}(c) = \{c{:}m\},\ \text{ports}(m) = \{m{:}c, m{:}p, m{:}s\},\ \text{ports}(s) = \{s{:}m\},$$
$$A_{c:m} = \{ask_c, buy_c\},$$
$$A_{m:c} = \{order_m, pay_m\},\ A_{m:p} = \{print_m\},\ A_{m:s} = \{deliver_m, reserve_m\}, \text{ and}$$
$$A_{s:m} = \{mark_s, ship_s\}.$$

The interaction model based on this protocol component system is similar to the original one, i.e., we connect the actions in the following way:

$$Int = \big\{\{ask_c, order_m\}, \{buy_c, pay_m\}, \{deliver_m, ship_s\}, \{print_m\}, \{reserve_m, mark_s\}\big\} \text{ and}$$
$$Int_{\text{closed}} = \big\{\{print_m\}\big\}.$$

Figure 7.1 illustrates the interaction model based on the protocol component system of the simplified version of the merchandise management example. We use a diamond shape to visualize the ports attached to a component.

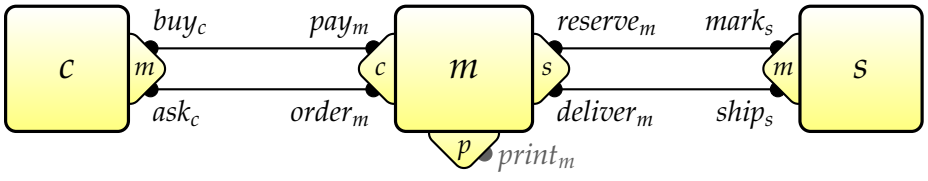
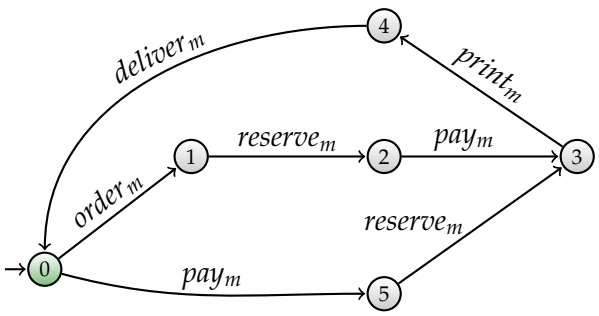


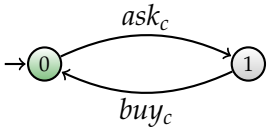
Figure 7.1: Interaction model based on a protocol component system

Now, in order to complete the specification for a protocol interaction system, we have to give the component and the port behavioral models. For the former, we use a similar behavior as for the original merchandise management system without the cancel/refund operations and the direct delivery branch.

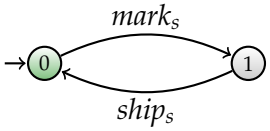
Figure 7.2 depicts the labeled transition systems of the component behavioral model.



(a) Behavior of component m



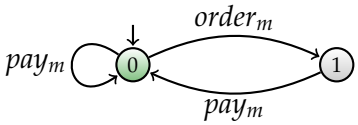
(b) Behavior of component c



(c) Behavior of component s

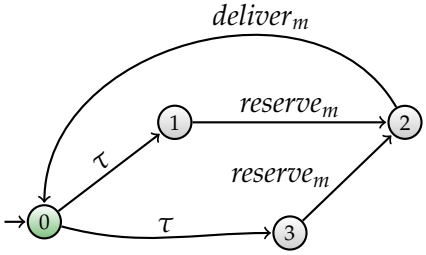
Figure 7.2: Behavior of the components of the protocol interaction system

Furthermore, we have to give the port behavioral model, i.e., define labeled transition systems over the corresponding port alphabets. We start with the (simple) management component where we have to give three labeled transition systems for the three ports, which are depicted in Figure 7.3.



(a) Port protocol $[\![m{:}c]\!]$

(b) Port protocol $[\![m{:}p]\!]$



(c) Port protocol $[\![m{:}s]\!]$

Figure 7.3: Port protocols of the simple management component m

We derived the port protocols as follows (up to now, we have not addressed an automatic procedure for this task). Consider the cooperations that are possible over the port $m{:}c$ with respect to the local behavior $[\![m]\!]$ (cf. Figure 7.2 (a)). A customer can either order something and then has to pay, or directly pay. Afterwards, this process can be repeated which leads to the port protocol as depicted in Figure 7.3 (a). Similarly, a (potential) printer component could observe consecutive print actions over the port $m{:}p$ which results in the port protocol depicted in Figure 7.3 (b). For the port $m{:}s$ and a storage component, the reservation and the delivery actions are triggered after the customer has chosen to either order or pay directly. We model the initial unobservable choice as two τ-transitions leading to different states which then results in the port protocol depicted in Figure 7.3 (c).

The customer and the storage component have only one port for cooperating with the management component. Thus, we can use the behavior of the component as the protocol of the respective port. However, to complete the specification, Figure 7.4 depicts the corresponding port protocols.

(a) Port protocol $[\![c{:}m]\!]$ of component c **(b)** Port protocol $[\![s{:}m]\!]$ of component s

Figure 7.4: Port protocols of the simple customer and storage components

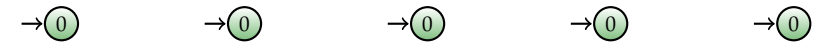
Observe that this completes the specification of a protocol interaction system according to Definition 7.2.

Now, the global behavior of a protocol interaction system is defined analogously as for an interaction system (cf. Definition 2.6). Clearly, a protocol interaction system can be understood as an interaction system by ignoring the port alphabets and protocols. Thus, we can use the operators for interaction systems defined in Chapter 3 and moreover, define corresponding ones for protocol interaction systems. However, since these operators do not modify the components, we skip such definitions here. For instance, a subsystem construction operator for protocol interaction systems works the same way as the one for interaction systems (cf. Definition 3.18), but has to include all ports of the components in the resulting subsystem in order to yield a valid protocol interaction system.

We already mentioned that we have not yet required any relation between port protocols and component behavior. Next, we discuss this issue.

7.1.1 Relating Port Protocols and Component Behavior

In the introduction to this chapter, we stated that for a component gray-box view, the access to the behavior of the components, i.e., the component behavioral model, is restricted and cannot be used in verification steps. Instead, the port protocols should be used. But up to now, it is sufficient to assign a labeled transition system with a single initial state and no transitions for each port since the only requirement in Definition 7.2 is that no set of states is empty, e.g., the port protocols depicted in Figure 7.5 on the facing page can be used for the protocol merchandise management example instead of the ones specified in Figures 7.3 and 7.4.



(a) $[\![c{:}m]\!]$ **(b)** $[\![m{:}c]\!]$ **(c)** $[\![m{:}p]\!]$ **(d)** $[\![m{:}s]\!]$ **(e)** $[\![s{:}m]\!]$

Figure 7.5: Alternative port protocols for the example

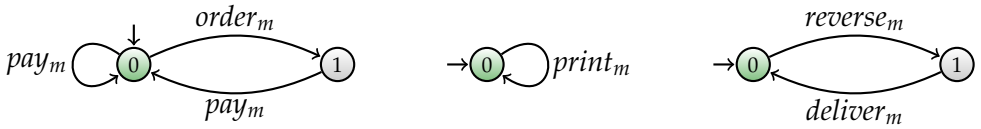
However, since our original plan is to use the port protocols instead of the components' local behavior in verification steps, we need to investigate on the necessary relation of a port and its corresponding component. When we introduced the port protocols in the previous section, we already addressed this issue in a vague way: A port protocol should indicate what kind of (partial) behavior of the component can be observed over the port. The protocol thus acts as a window to the component that allows for seeing some of its internals.

In Chapter 2, we introduced a behavioral equivalence for interaction systems and labeled transition systems. Thus, it is natural to use this equivalence, viz. branching bisimilarity with explicit divergence, as the relation between a port protocol and the behavior of the corresponding component. Here, the behavior of the component that is observable through the port should be restricted to the port alphabet and be as small as possible. The following notion of *port conformance* formalizes this idea where we require that the local behavior of the component restricted to the alphabet of the port is branching bisimilar (with explicit divergence) to the port protocol.

Definition 7.3 (Port Conformance): Let *Sys* be a protocol interaction system. A port $i{:}p$ of a component i is said to be *conform* to the component if $[\![i{:}p]\!] \approx_b^\Delta f_{i:p}([\![i]\!])$ holds where $f_{i:p}$ is a relabeling function that replaces all labels and transitions not contained in the port alphabet $A_{i:p}$ with the label τ and with a τ-transition respectively, i.e., for $[\![i]\!] = (S_i, A_i, \{\xrightarrow{a}_i\}_{a\in A_i}, S_i^0)$ we have $f_{i:p}([\![i]\!]) = (S_i, A_{i:p} \cup \{\tau\}, \{\xrightarrow{a}_i\}_{a\in A_{i:p}\cup\{\tau\}}, S_i^0)$ with $\xrightarrow{\tau}_i = \bigcup_{a\in A_i\setminus A_{i:p}} \xrightarrow{a}_i$. Furthermore, we assume that the port protocol is minimal with respect to branching bisimilarity with explicit divergence if it is conform, i.e., we have $[\![i{:}p]\!] = [\![i{:}p]\!]_{\approx_b^\Delta}$.

Thus, if we use the port protocols depicted in Figure 7.5 for the protocol merchandise management example, i.e., the labeled transition systems each consisting of a single initial state without transitions, it holds that no port is conform to its associated component since all components offer an outgoing transition in their initial states (cf. Figure 7.2). Thus, we take a look at the original specification of the example, i.e., the port protocols specified in Figures 7.3 and 7.4. Since the components c and s have only one port and the port protocol is identical to the component's behavior in each case, these ports are conform.

For component m, we proceed as follows: We take the original labeled transition system $[\![m]\!]$ (cf. Figure 7.2 (a)), replace for each port all transitions whose label is not contained in the port alphabet with τ (which corresponds to the relabeling function given in Definition 7.3), and compute the quotient with respect to branching bisimilarity with explicit divergence (cf. Definition 2.16) of the resulting system. Figure 7.6 depicts the three quotients.



(a) Quotient of $f_{m:c}([\![m]\!])$ **(b)** Quotient of $f_{m:p}([\![m]\!])$ **(c)** Quotient of $f_{m:s}([\![m]\!])$

Figure 7.6: Quotients of the behavior of component m

Now, if we compare these labeled transition systems, i.e., the one in Figure 7.3 (a) with Figure 7.6 (a), Figure 7.3 (b) with Figure 7.6 (b), and Figure 7.3 (c) with Figure 7.6 (c), we learn that the ports $m{:}c$ and $m{:}p$ are conform to component m but port $m{:}s$ is not because its port protocol is not minimal with respect to branching bisimilarity with explicit divergence. Thus, we replace the port protocol $[\![m{:}s]\!]$ with the labeled transition system depicted in Figure 7.6 (c) and gain a protocol interaction system where all ports are conform to their associated components.

We already mentioned our goal of using the port protocols instead of the local component behavior in verification steps. In order to approach this goal, we need a way to combine the port protocols of two cooperating components such that we can examine their joint behavior similar to interaction systems created with the subsystem construction operator of Chapter 3. The following definition of the *port behavior* allows for this examination.

Definition 7.4 (Port Behavior): Let *Sys* be a protocol interaction system and $P \subseteq \bigcup_{i \in Comp} \text{ports}(i)$ a set of ports of *Sys*'s components. The *port behavior* of *Sys* with respect to P is defined as the labeled transition system $[\![P]\!] := (S_P, Int_P \cup \{\tau\}, \{\xrightarrow{\alpha}_P\}_{\alpha \in Int_P \cup \{\tau\}}, S_P^0)$ where $S_P = \prod_{i:p \in P} S_{i:p}$, $S_P^0 = \prod_{i:p \in P} S_{i:p}^0$, and $Int_P = \{\alpha \cap (\bigcup_{i:p \in P} A_{i:p}) \mid \alpha \in Int\} \setminus \{\emptyset\}$. For all $\alpha \in Int_P$ and all $s, t \in S_P$ we have $s \xrightarrow{\alpha}_P t$ if and only if $\forall\, i{:}p \in P$: if $i{:}p(\alpha) = \{a\}$ then $s_{i:p} \xrightarrow{a}_{i:p} t_{i:p}$ and if $i{:}p(\alpha) = \emptyset$ then $s_{i:p} = t_{i:p}$. Additionally, for $s, t \in S_P$ we have $s \xrightarrow{\tau}_P t$ if and only if $\exists\, i{:}p \in P\colon s_{i:p} \xrightarrow{\tau}_{i:p} t_{i:p} \ \wedge\ (\forall\, j{:}q \in P \setminus \{i{:}p\}\colon s_{j:q} = t_{j:q})$, i.e., only one unobservable local step is allowed to happen during an unobservable step in the port behavior. If all reachable states of $[\![P]\!]$ have at least one successor, i.e., $\forall\, s \in S_P \ \ \forall\, s^0 \in S_P^0\colon s^0 \longrightarrow_P^* s \implies \text{Suc}(s) \neq \emptyset$ holds with $\longrightarrow_P = \bigcup_{\alpha \in Int_P \cup \{\tau\}} \xrightarrow{\alpha}_P$, then we call the port behavior *deadlock-free*.

Please note that we do not distinguish between open and closed interactions in Definition 7.4. This is reasonable since we currently only analyze a single protocol interaction system. An extension with respect to several protocol interaction systems is left for future work.

We want to mention that an interaction system can be transformed into a protocol interaction system with conform ports in a straightforward way—the other direction is even simpler since, as already mentioned above, we just have to ignore the ports and take the union of the port alphabets as action sets. The following corollary fixes the former transformation.

Corollary 7.5 (Relation of Protocol and "Normal" Interaction Systems): Let *Sys* be an arbitrary interaction system with a set *Comp* of components, a set *Int* of interactions over an action set $Act = \bigcup_{i \in Comp} A_i$, a set of closed interactions Int_{closed}, and a family $\{[\![i]\!]\}_{i \in Comp}$ of labeled transition systems describing the local behavior of the components. Introduce for every component $i \in Comp$ a port $i{:}p$ with the port alphabet $A_{i:p} = A_i$. The port protocol of each such port corresponds to the local behavior of the component, i.e., we set $[\![i{:}p]\!] = [\![i]\!]$. The interaction sets need not to be modified. Now, we gained a protocol interaction system where all ports are conform to their corresponding component.

We omit a proof of Corollary 7.5 since it directly follows from the definition that the transformation yields a valid protocol interaction system with conform ports. We want to point out that these transformations imply that all definitions such as deadlock-freedom (cf. Definition 2.8) or livelock-freedom (cf. Definition 2.9) are also valid for protocol interaction systems. Moreover, the behavioral equivalence between interaction systems (cf. Definition 2.15) carries over to protocol interaction systems. Next, we introduce architectural constraints for protocol interaction systems.

7.1.2 Architectures of Protocol Interaction Systems

We define the architecture of a protocol interaction system as a special graph similar to the component graph for interaction systems (cf. Definition 4.1). In order to restrict this architecture as in Chapter 4, we require that certain properties of this graph hold which we call a *tree-like protocol architecture*.

Definition 7.6 (Protocol Component Graph and Architecture): The *protocol component graph* $G_{\text{prot}} := (V, E)$ of a protocol interaction system *Sys* is defined by the set of vertices $V = Comp \cup (\bigcup_{i \in Comp} \text{ports}(i))$ and the set of edges $E = \{\{i, i{:}p\} \mid i \in Comp \ \wedge \ i{:}p \in \text{ports}(i)\} \cup \{\{i{:}p, j{:}q\} \mid i, j \in Comp \ \wedge \ i{:}p \in \text{ports}(i) \ \wedge \ j{:}q \in \text{ports}(j) \ \wedge \ (\exists\, \alpha \in Int\colon i{:}p(\alpha) \neq \emptyset \ \wedge \ j{:}q(\alpha) \neq \emptyset)\}$. Two

ports are *connected* if they are related by an edge in G_{prot}. The *port connectivity* of a port $i{:}p$ is defined as the number of ports to which $i{:}p$ is connected. If the port connectivity of a port is less than two, we say that the port is *uniquely connected*. If G_{prot} forms a tree in the graph-theoretical sense (cf. Definition 4.3), we say *Sys* has a *tree-like protocol architecture*.

Since it is more convenient to have a graphical representation of the protocol component graph, we use the following notation: Components are depicted as rectangles (as in the component graph, cf. Definition 4.1) and ports as diamonds. Figure 7.7 depicts the protocol component graph of the protocol merchandise management example. Observe that the graph is a tree in the graph-theoretical sense and the port connectivity of every port is less than two, i.e., the example has a tree-like protocol architecture and all ports are uniquely connected.

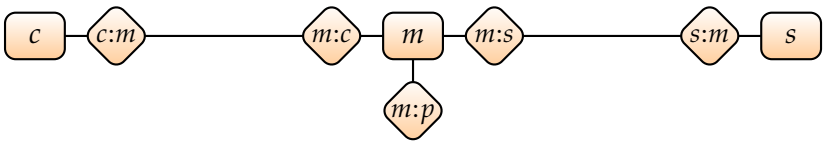


Figure 7.7: Protocol component graph of the protocol merchandise management example, where we use a diamond shape for the vertices representing the ports.

We omit an algorithm for deciding whether a protocol interaction system has a tree-like protocol architecture because such an algorithm is analogously to the one for tree-like architectures discussed in Section 4.3.1.

Next, we show how this architectural constraint can be used to verify deadlock-freedom of protocol interaction systems without accessing the behavior of the components.

7.2 Deadlock Detection with Port Protocols

In order to exploit the compositional information and the information obtained by combining the port protocols via the port behavior (cf. Definition 7.4), we put restrictions on the architecture as above and on the local behaviors, i.e., on the existence of unobservable behavior in the port protocols. We then establish an efficiently checkable condition for deadlock-freedom in protocol interaction systems, which can be verified in time polynomial in the number and size of the port protocols without accessing the behavior of the components.

Before we consider deadlock-freedom of a protocol interaction system, we analyze the deadlock information that can be derived due to the port conformance (cf. Definition 7.3). The following lemma addresses this issue. We want to mention that the property of deadlock-freedom is only defined for protocol interaction systems (as for interaction system, cf. Definition 2.8) and port behaviors (cf. Definition 7.4), i.e., we have to be careful which structures we call deadlock-free.

Lemma 7.7: Let *Sys* be a protocol interaction system, $i \in Comp$ one of its components, and $i{:}p \in \text{ports}(i)$ one of i's ports that is conform to i. The port behavior $[\![\{i{:}p\}]\!]$ is deadlock-free if and only if $Sys[\{i\}]$ is deadlock-free.

A formal proof of Lemma 7.7 can be found in Appendix F on page 267.

Thus, we can exclude the possibility that a deadlock is introduced into a system by a component that has a state without outgoing transitions by requiring that all ports are conform—which is reasonable as discussed above—and checking the deadlock-freedom of one port behavior consisting of a single port for each component.

Now, in order to verify deadlock-freedom of a whole protocol interaction system, we use the following idea: An unobservable step in a port protocol is only present if the component's future behavior can be influenced by the cooperation with its environment. If no τ-transition is present, the component's behavior visible through the port protocol is inevitable unless the component gets stuck, i.e., involved in a deadlock. Here, the requirements of a tree-like protocol architecture and uniquely connected ports allow that it is sufficient to check pairs of port protocols for deadlock-freedom, because due to their τ- and deadlock-freedom, no cyclic waiting relation is possible.

Next, we formalize this idea as a theorem.

Theorem 7.8: Let *Sys* be a protocol interaction system. Assume that *Sys* has a tree-like protocol architecture and that every port is uniquely connected and conform to its corresponding component and that all port protocols are free of τ-transitions and states without successors. If for all connected ports $i{:}p \in \text{ports}(i)$ and $j{:}q \in \text{ports}(j)$ of all components $i, j \in Comp$ it holds that $[\![\{i{:}p, j{:}q\}]\!]$ is deadlock-free, then $[\![Sys]\!]$ is deadlock-free.

A formal proof of Theorem 7.8 can be found in Appendix F on page 268.

We apply the theorem to the protocol merchandise management example where we use the port protocols specified in Figure 7.3 (a), Figure 7.3 (b), and Figure 7.6 (c) for component m. As mentioned above, all ports are con-

form to their associated component and, clearly, all port protocols are free of τ-transitions and states without successors. Thus, the assumptions of Theorem 7.8 are satisfied because, as discussed in Section 7.1.2, the system has a tree-like protocol architecture and all ports are uniquely connected.

Now, we have to compute the port behaviors of all connected ports, viz. $[\![\{c{:}m, m{:}c\}]\!]$ and $[\![\{m{:}s, s{:}m\}]\!]$, which are depicted in Figure 7.8.

(a) Port behavior $[\![\{c{:}m, m{:}c\}]\!]$ **(b)** Port behavior $[\![\{m{:}s, s{:}m\}]\!]$

Figure 7.8: Port behaviors of the connected ports

Obviously, both port behaviors $[\![\{c{:}m, m{:}c\}]\!]$ and $[\![\{m{:}s, s{:}m\}]\!]$ are deadlock-free. Thus, the deadlock-freedom of the protocol merchandise management example is implied by Theorem 7.8.

As already mentioned, the application of Theorem 7.8 is very efficient since we only have to compute the port behaviors of uniquely connected ports in a system that has a tree-like protocol architecture, i.e., we can perform these computations in time polynomial in the number and size of the port protocols. Moreover, we do not need accessing the behavior of the components—if we assume that the port conformance is established beforehand—and our approach thus supports the gray-box view mentioned in the introduction to this chapter.

In the next section, we take a look at the potential savings that our approach allows for since the port protocols are usually much smaller than the behavior of the components.

7.2.1 Potential Savings

We address how deadlock analyses in interaction systems can benefit from port protocols despite the limited information due to a gray-box view, i.e., restricted access to the behavior of the components.

Consider a protocol interaction system *Sys* with one master component m that is surrounded by client components $1, \ldots, n$ for a $n \in \mathbb{N} \setminus \{0\}$. The master component offers one port containing a single action for each client and each client has two ports, one for the master component containing a single action

and another one—called r—containing $n-1$ actions. Thus, we have (where i ranges over $1, \ldots, n$):

$$Comp = \{m, 1, 2, \ldots, n\},$$
$$\text{ports}(m) = \{m{:}1, \ldots, m{:}n\} \text{ and } \text{ports}(i) = \{i{:}m, i{:}r\},$$
$$A_{m:i} = \{a_m^i\},\ A_{i:m} = \{a_i^i\}, \text{ and } A_{i:r} = \{a_i^1, \ldots, a_i^n\} \setminus \{a_i^i\}.$$

Now, the actions of the ports are used for synchronizing the components in the following way:

$$Int = \left\{ \{a_m^i, a_i^i\}, \{a_i^k\} \mid i \in \{1, \ldots, n\} \ \wedge\ k \in \{1, \ldots, n\} \setminus \{i\} \right\} \text{ and}$$
$$Int_{\text{closed}} = \{\}.$$

Figure 7.9 illustrates the resulting interaction model.

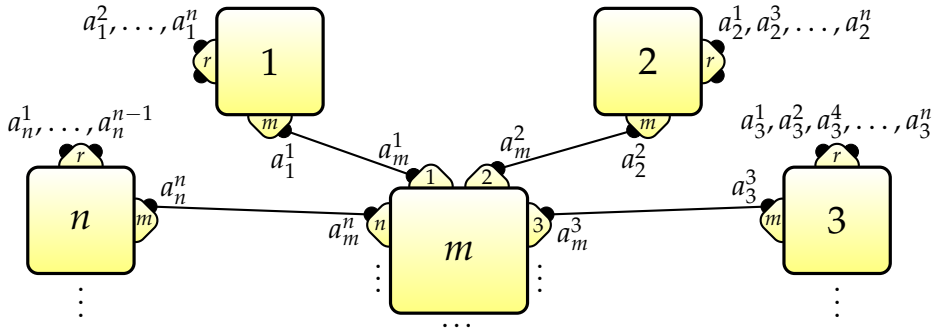
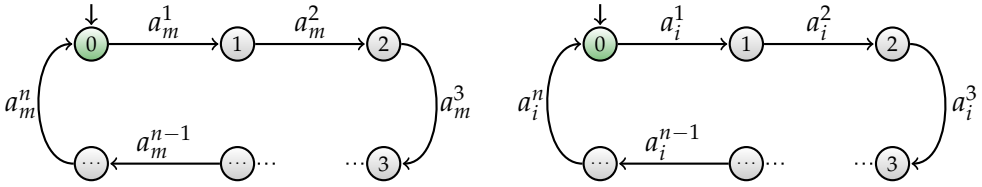


Figure 7.9: Interaction model of the protocol interaction system

Next, we specify the component and the port behavioral model. Observe that each component has n actions that can be numbered from 1 to n. For the component behavior, we consecutively execute these actions in their natural order. Figure 7.10 (a) depicts the labeled transition system for the master component m, and Figure 7.10 (b) the one for the clients.



(a) Behavior of component m **(b)** Behavior of component i, $1 \leq i \leq n$

Figure 7.10: Component behavior of the protocol interaction system

For the port behavioral model, we only address the connected ports. Figure 7.11 on the following page depicts these port protocols: Figure 7.11 (a)

depicts the port protocol of each port $m{:}i$ of the master component m where $i \in Comp \setminus \{m\}$ denotes a client. Similarly, Figure 7.11 (b) shows the port protocols of the ports $i{:}m$ of all client components $i \in Comp \setminus \{m\}$. Clearly, the port protocols are conform (cf. Definition 7.3) and free of τ-transitions and states without successors. The conformance holds since if we replace all transitions in a component's behavior except the one contained in the port alphabet in question with τ-transitions, then the resulting system consists of one observable transition preceded and succeeded respectively by τ-transitions. The quotient with respect to branching bisimilarity with explicit divergence of such a system clearly is a single initial state with a self-loop as depicted in Figure 7.11 (cf. Definition 2.16).

(a) Port protocol $[\![m{:}i]\!]$, $1 \leq i \leq n$

(b) Port protocol $[\![i{:}m]\!]$, $1 \leq i \leq n$

Figure 7.11: Port protocols of the protocol interaction system

Please note that the port protocols of the ports $i{:}r$ of all components $i \in Comp \setminus \{m\}$ can be similarly derived in a conform and τ-free way by taking the corresponding component behavior, replacing the action and transition respectively that is not contained in the port alphabet $A_{i:r}$ with τ, and computing the quotient with respect to branching bisimilarity with explicit divergence (cf. Definition 2.16). These systems are also free of states without successors.

Now, assume we want to prove the deadlock-freedom of this example. Since the interaction model does not put strong restrictions on the cooperation of the components, all state combinations are possible and reachable, i.e., if we compute the global behavior of the example, we quickly face the state space explosion problem since the global state space consists of all n^{n+1} state combinations where n is the number of clients.

However, we can use our approach from Chapter 6 since the example clearly is disjoint circular wait free (ignoring the port protocols and treating the example as an interaction system). If we want to apply Theorem 6.9 (cf. page 154), we have to compute the problematic states. Without going into the details of this computation here, we find out that no problematic states exists in the example and thus its deadlock-freedom is implied by Theorem 6.9. But, while performing this computation, we need to analyze all subsystems of cooperating components of size two that consist of the master component m and a client component i for $i \in Comp \setminus \{m\}$ in each case, i.e., a total number of n labeled transition systems, and all these systems are of size n^2 where n is the number of clients.

As an alternative to this analysis, we can apply the approach from this chapter, i.e., Theorem 7.8 (cf. page 193). We already mentioned above that all port protocols are conform and free of τ-transitions and states without successors. The protocol component graph of the example is a tree in the graph-theoretical sense—hence, the system has a tree-like protocol architecture—and every port is uniquely connected (cf. Definition 7.6 and Figure 7.9—we here omit to depict the graph since it looks similar to the system's interaction model). Thus, the requirements of Theorem 7.8 are satisfied and we have to analyze the port behavior (cf. Definition 7.4) of all connected ports. We can see in Figure 7.11 that all resulting labeled transition systems consist of one initial state with a self-loop labeled by the corresponding interaction of the port behavior in question. Since all these port behaviors are deadlock-free, Theorem 7.8 implies deadlock-freedom of the example. Here, we need to analyze all port behaviors of connected ports of size two that consist of the master component's port $m{:}i$ and a client component's port $i{:}m$ for $i \in \mathit{Comp} \setminus \{m\}$ in each case, i.e., a total number of n labeled transition systems, and all these systems are of constant size.

The difference in the size of the labeled transition systems under analysis of the approaches of Chapter 6 and this chapter corresponds to a quadratic factor that we can save in the asymptotical costs. Thus, we can benefit from port protocols in special situations as demonstrated by the example, although we do not have full access to all information of the protocol interaction system if we take a gray-box view as mentioned in the introduction of this chapter.

7.2.2 Conjectured Extension

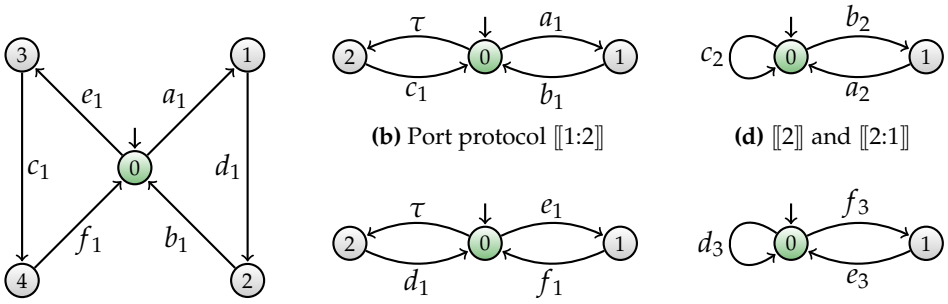
In the paper where we introduced port protocols for interaction systems [161], we conjectured that we can extend Theorem 7.8 by weakening one of its assumption. To quote from our paper [161]: "We conjecture that it is sufficient that the protocol behavior of combined port protocols is τ-free instead of requiring the τ-freedom of all port protocols." Note that we call the protocol behavior of combined port protocols simply port behavior in this thesis, but in our former paper, we additionally required that the combined behavior is minimal with respect to branching bisimilarity, i.e., we consider the quotient of the port behavior in the following.

Unfortunately, this conjecture is false as we show by the following counterexample. Consider the following protocol interaction system *Sys* with:

$$\mathit{Comp} = \{1, 2, 3\},$$
$$\text{ports}(1) = \{1{:}2, 1{:}3\},\ \text{ports}(2) = \{2{:}1\},\ \text{ports}(3) = \{3{:}1\},$$

$A_{1:2} = \{a_1, b_1, c_1\}$, $A_{1:3} = \{d_1, e_1, f_1\}$, $A_{2:1} = \{a_2, b_2, c_2\}$, $A_{3:1} = \{d_3, e_3, f_3\}$,
$Int = \{\{a_1, a_2\}, \{b_1, b_2\}, \{c_1, c_2\}, \{d_1, d_3\}, \{e_1, e_3\}, \{f_1, f_3\}\}$, and
$Int_{\text{closed}} = \{\}$.

The behavior of the components and the port protocols are given in Figure 7.12 where use the same labeled transition system for $[\![2]\!]$ and $[\![2{:}1]\!]$ and for $[\![3]\!]$ and $[\![3{:}1]\!]$.

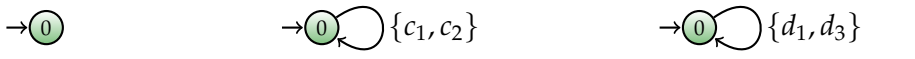


(a) Comp. behavior $[\![1]\!]$ **(b)** Port protocol $[\![1{:}2]\!]$ **(d)** $[\![2]\!]$ and $[\![2{:}1]\!]$ **(c)** Port protocol $[\![1{:}3]\!]$ **(e)** $[\![3]\!]$ and $[\![3{:}1]\!]$

Figure 7.12: Component behaviors and port protocols of the counterexample

Observe that all ports are conform to their associated component. Clearly, the architecture of this system is tree-like because the only ports that are connected are 1:2 with 2:1 and 3:2 with 3:1. However, not all port protocols are free of τ-transitions, viz. $[\![1{:}2]\!]$ and $[\![1{:}3]\!]$ (cf. Figure 7.12 (b) and (c)). Thus, we cannot apply Theorem 7.8 to show deadlock-freedom of the example.

But what about our conjecture? First, we compute the global behavior of the system which is depicted in Figure 7.13 (a). Clearly, the system is not deadlock-free—this can also be seen from the specification of the protocol interaction system (cf. Figure 7.12): Observe that component 1 wants to execute interactions $\{a_1, a_2\}$ or $\{e_1, e_3\}$ in its initial state where its partners, components 2 and 3, want to execute interactions $\{b_1, b_2\}$ or $\{c_1, c_2\}$ and $\{d_1, d_3\}$ or $\{f_1, f_3\}$ respectively. This mismatch results in the deadlock that is depicted in Figure 7.13 (a).



(a) Global behavior **(b)** Port behavior $[\![\{1{:}2, 2{:}1\}]\!]_{\approx_b^{\Delta}}$ **(c)** Port behavior $[\![\{1{:}3, 3{:}1\}]\!]_{\approx_b^{\Delta}}$

Figure 7.13: Global behavior and port behaviors of the counterexample

Now, we apply our conjecture from above where we have to compute the labeled transition systems $[\![\{1{:}2, 2{:}1\}]\!]_{\approx_b^{\Delta}}$ and $[\![\{1{:}3, 3{:}1\}]\!]_{\approx_b^{\Delta}}$, i.e., the quotients of the port behaviors of the connected ports. The quotients are depicted in

Figure 7.13 (b) and (c). Observe that both quotients are free of τ-transitions and states without successors, i.e., all assumptions of our conjecture hold. However, the deadlock-freedom of the whole protocol interaction system is a false implication now because the global initial state is a deadlock. This ends our discussion of the conjectured extension of Theorem 7.8.

In the next section, we take a look at related approaches from the literature that employ similar ideas as our port protocols.

7.3 Related Approaches and Discussion

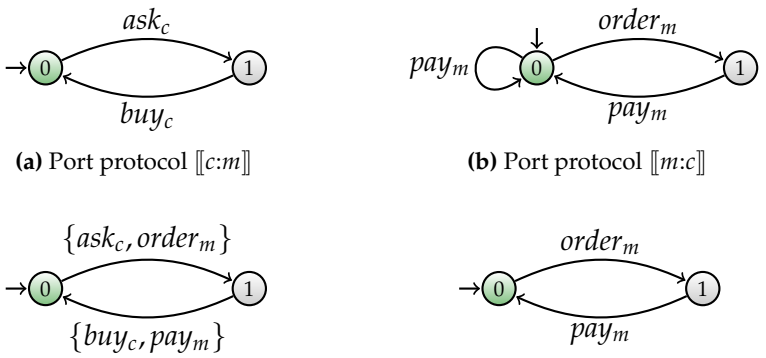
Hennicker et al. [137] and Mota et al. [207] study similar ideas as our protocol interaction systems, which we discuss and compare in this section. In both works, the authors define port protocols on the basis of a formal model for component-based systems.

The approach of Hennicker et al. [137] introduces port protocols in the way we stated our goals formulated at the beginning of this chapter. In their component model, each component's port provides a protocol which is in the following correctness relation to the behavior of the component. A component is called correct with respect to one of its ports, if the behavior of the component restricted to the actions in the port alphabet (i.e., any other action becomes an unobservable τ-action) is weakly bisimilar to the port protocol. Of course, it is assumed that each component is correct with respect to all of its ports. The authors do not provide an automatic generation of such port protocols, i.e., they assume that they are given by the component's author. But, obviously, the correctness relation between a component and one of its ports can be used to derive a suitable port protocol.

Then, a notion called "neutrality" allows to apply a reduction strategy such that properties need only to be verified on the reduced part of the system. Note that this notion is also defined for the components (in the absence of port protocols) which we discussed in Section 5.5 of Chapter 5. Here, neutrality of a port q for a port p means that the composition of the protocols of p and q restricted to the alphabet of p is weakly bisimilar to the protocol of p, i.e., it is sufficient to consider only p. This protocol composition works similar to the port behavior we defined above (cf. Definition 7.4) and we can thus apply the approach of Hennicker et al. [137] to protocol interaction systems.

However, the port neutrality seems to be a strong restriction as can be seen by means of the protocol merchandise management example. We showed in Section 7.1.1 that the port protocols are conform to their components and

thus, they are also correct in the notion of Hennicker et al. [137] since labeled transition systems that are branching bisimilar (with explicit divergence) are also weak bisimilar. Now, Figure 7.14 depicts the combined behavior of the respective port protocols of the customer and the management component, i.e., the port behavior $[\![\{c{:}m, m{:}c\}]\!]$. The latter is depicted in Figure 7.14 (c), and Figure 7.14 (d) depicts the restricted version.



(a) Port protocol $[\![c{:}m]\!]$ **(b)** Port protocol $[\![m{:}c]\!]$

(c) Port behavior $[\![\{c{:}m, m{:}c\}]\!]$ **(d)** $[\![\{c{:}m, m{:}c\}]\!]$ restricted to $A_{m:c}$

Figure 7.14: Port protocols and port behavior of the protocol merchandise management example

Clearly, the two systems are not weakly bisimilar because only one of them has a pay_m-self-loop in its initial state—compare Figure 7.14 (b) and Figure 7.14 (d).

Moreover, weak bisimilarity is in our opinion not suited for the relationship between components and their ports. Any information about the branching structure is typically lost and with the additional requirement by Hennicker et al. [137, Section 4.2] of weakly deterministic port protocols and the implied existence of a weakly bisimilar τ-free port protocol, any information about the unobservable but important internal behavior of the component is lost in the port protocol. Of course, we made a similar assumption by requiring the absence of τ-transitions in Theorem 7.8, and the necessity of this assumption for the verification of deadlock-freedom is a drawback of the approach. However, with weak bisimilarity this information is lost a priori which indicates a loss of information due to the definition of the model.

For instance, if a τ-self-loop exists in a port protocol at a state which has another non-τ outgoing transition, the τ-self-loop can simply be left out, because the behavior of this new port protocol is weakly bisimilar to the former behavior. But, an important information is lost during this step: The port protocol indicates at this state that the component could forever cooperate through other ports with its environment, and thus this port is never used for

cooperation. However, since a non-τ outgoing transition exists, the port may eventually be used for cooperation.

A similar observation exists with inevitable τ-divergent behavior, which indicates at a port protocol that the corresponding component has reached a state in which it never uses the port for cooperation again. This could indicate a design flaw of the component, but also situations are imaginable in which a port offers a special action which serves as a safety seal in order to disable the functionality of this port completely. Or, similarly, it is possible to disable this port over another port with the execution of a transition leading into a τ-divergent state. When using weak bisimilarity, all of this possible interesting information is lost in the port protocol.

In the work by Mota et al. [207], a similar idea is called "compatibility" of two ports and requires that all sequences of actions of one port are also possible in the other one if we abstract from input and output actions. In their component model, the behavior of a component is given as a CSP [141] process that offers channels for the component's cooperation with its environment which can be understood as the ports of the component. These channels are related by connectors that are also modeled as CSP processes. A protocol is defined as a projection of a component's behavior over a channel where this projection is done by concealing every symbol that is not contained in the channel's alphabet. If it now holds for two protocols that "all possible sequences of output values in one channel are accepted by the corresponding input channel" [207] and vice versa, they are called compatible. However, our protocol merchandise management example discussed around Figure 7.14 above shows that the depicted protocols are not compatible in this sense if we abstract from I/O operations, i.e., the approach of Mota et al. [207] fails for our example.

The name "protocol" has, of course, a very broad meaning and has been used in quite different settings. We mention one occurrence with respect to component-based systems that we already discussed in Chapter 2: The SOFA component model which features so-called behavior protocols [228]. A behavior protocol extends a SOFA component with a regular expression like protocol, which represents traces of the component's behavior that can be observed on the interface of the component. In the following, we use the terminology for behavior protocols introduced by Ježek et al. [148]. The authors call the compliant behavior of a component with its ports "horizontal protocol compatibility" of the components' behavior protocols. This compatibility check also features (by interconnecting two behavior protocols) the detection of bad activity, i.e., the request of a component cannot be answered by the other one, no activity, i.e., local deadlock of the interconnection, and

infinite activity, i.e., the interconnection forms a communication loop which is never exited. Additionally, a "vertical protocol compatibility" called "protocol compliance" is available which allows for checking whether the behavior protocol of a composite component corresponds to the interconnection of the behavior protocols of its subcomponents.

However, as a behavior protocol abstracts the behavior of a whole component, the idea behind behavior protocols is different from the questions we focus on; thus, although it is an interesting way of checking the compliance of two components, the behavior protocols are not comparable to our approach. Note that in interaction systems, a behavior protocol is comparable to the behavior of the whole component, i.e., the underlying labeled transition system.

Finally, Graf and Steffen [125] show how interface specifications, that are supersets of the set of sequences which can be observed at a certain interface of a process and that are provided by the designer of the system, can be used to minimize finite state systems in a compositional way. This potentially indicates an opportunity for future work, e.g., whether we can apply compositional reduction as discussed in Chapter 5 to protocol interaction systems where we only use the port protocols in equivalence checks. We want to mention that interface generation of software components, i.e., the interface specifications are not provided, is also an active field of research, e.g., a recent approach is discussed by Giannakopoulou and Păsăreanu [111].

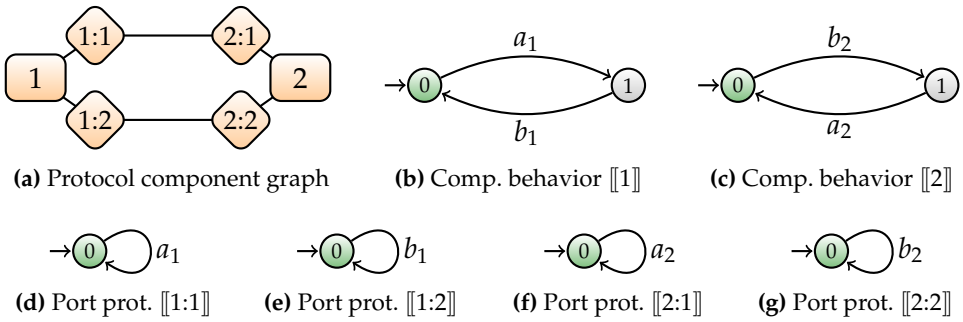
This ends our discussion of related approaches. In the next section, we address some limitations of the port protocols approach.

7.4 Limitations of the Port Protocols Approach

We take a closer look at the acyclicity requirement of the protocol component graph, that we used to establish a condition for deadlock-freedom in Section 7.2, and what information can be derived from the port protocols in the absence this requirement. As we see in the following, this reveals some limitations of the port protocols approach.

We consider a mutual blocking scenario in a protocol interaction system. Such a mutual blocking of two components is the smallest deadlock—measured in the number of involved components—that can occur in a protocol interaction system if we assume that all behaviors of the components are "locally deadlock-free". Of course, such "mutual" deadlocks can be efficiently detected, because we only need to analyze all subsystems of size two for deadlocks which is possible in polynomial time, i.e., if $|S_{\max}|$ denotes the largest local state space, then this detection can be performed in $O(|Comp|^2 \cdot |S_{\max}|^2)$.

However, the introduction of port protocols and the reliance on only considering these protocols in a deadlock analysis is flawed if mutual blocking is present in a system. The following example gives a first indication why a protocol-only approach is not sufficient. Figure 7.15 depicts the protocol component graph and behavioral models of the following protocol interaction system *Sys* with $Comp = \{1, 2\}$, $\text{ports}(1) = \{1{:}1, 1{:}2\}$, and $\text{ports}(2) = \{2{:}1, 2{:}2\}$. Each port alphabet contains a single action, we have $A_{1:1} = \{a_1\}$, $A_{1:2} = \{b_1\}$, $A_{2:1} = \{a_2\}$, and $A_{2:2} = \{b_2\}$. The interactions set is given by $Int = \{\{a_1, a_2\}, \{b_1, b_2\}\}$, i.e., the components cooperate over their a- and b-actions, and the set of closed interactions is empty. Clearly, all ports are conform to their associated components.



(a) Protocol component graph **(b)** Comp. behavior $[\![1]\!]$ **(c)** Comp. behavior $[\![2]\!]$

(d) Port prot. $[\![1{:}1]\!]$ **(e)** Port prot. $[\![1{:}2]\!]$ **(f)** Port prot. $[\![2{:}1]\!]$ **(g)** Port prot. $[\![2{:}2]\!]$

Figure 7.15: Protocol interaction system modeling a mutual blocking situation

Although the protocol interaction system is already deadlocked in its global initial state—observe that component 1 can only execute interaction $\{a_1, a_2\}$ and component 2 only interaction $\{b_1, b_2\}$ in the global initial state (s_1^0, s_2^0)—any protocol-only analysis (without additional assumptions) is unable to detect the mutual blocking because the port protocols model that any action is always executable. The port behaviors of the connected ports, viz. $[\![\{1{:}1, 2{:}1\}]\!]$ and $[\![\{1{:}2, 2{:}2\}]\!]$, consist of a single initial state labeled by the respective interaction, i.e., they are deadlock-free. Thus, no deadlock information can be derived from them.

Similarly to the mutual blocking in the previous scenario, any larger cycle of waiting situations among components in a protocol interaction system causes problems if only the protocols of the system are considered in an analysis. Here, we solved this problem by requiring the acyclicity of the protocol component graph that excluded such scenarios.

This ends our discussion of protocol interaction systems. We summarize the chapter in the next section and provide some directions for future work.

7.5 Summary and Future Work

We discussed how a white-box view of the components, that we assumed in the previous chapters, can be turned into a gray-box view by the introduction of port protocols. Such a view is reasonable with the advent of prefabricated components that do not fully disclose their internal details. With further assumptions, these port protocols allow for the verification of deadlock-freedom without looking at the component behavior and thus they potentially make verification steps more efficient since the labeled transition systems that need to be considered can be much smaller. However, we learned that also some drawbacks exist and the architecture of a systems needs to be restricted in order to detect cooperation mismatches that lead to deadlocks.

Despite these drawbacks, the available information could even be more restricted as the following scenario illustrates, which shows some directions for future work—of course, weaker assumptions that lie in between the ones of Theorem 7.8 and the (non-working) extension of Section 7.2.2 are also of interest. Additionally, the proof of Theorem 7.8 shows an application for a correctness-by-construction approach.

7.5.1 Partial Specification of Components via Port Protocols

Imagine a situation where a software architect ordered a prefabricated component at a software company, but the development time of this particular component lasts. Since the software architect wants to implement and test the behavior of the future component, the software company and the architect agree on a specification of the future component.

This specification is only partial because the full behavior is not known yet, and thus the protocol obtainment technique mentioned in Section 7.1.1 is not applicable. Now, logic is used for this contract, e.g., CTL*–X to specify the desired properties, i.e., the company provides the architect with a set of CTL*–X formulae that the later component is guaranteed to satisfy.

This view is more restricted than the gray-box view we discussed in this chapter, because the behavior is not specified completely. Here, it is only known that each port guarantees the validity of its associated set of formulae. However, we currently do not know how such a restricted view can be incorporate in (protocol) interaction systems. Please note that these are only early ideas for future work on the topic.

In the following chapter, we conclude the thesis.

Chapter 8

Conclusion

In this final chapter, we conclude the thesis by giving pointers to the sections where we discussed related work and by addressing additional directions for future work. But first, we shortly address our goals formulated at the beginning of the thesis (cf. Section 1.2.1): Have we accomplished our mission?

One of our goals is to guarantee *efficiency* which we ensured by showing upper bounds that are polynomial in the size of an interaction system's specification for all presented techniques—left aside the direct approaches based on a global behavior analysis discussed in Chapter 2. We derived *automatic* property verification techniques, which is our second goal, i.e., all techniques can be applied in a convenient way for a given interaction system and do not require any further user input, and moreover, these techniques allow for correctness by construction. Instead of building systems at will and then use, say, model checking to establish properties, we followed the philosophy to develop and investigate design patterns or architectural constraints, which we introduced in Chapter 4, that are amenable to the formulation of efficiently checkable conditions for the properties in question. We followed this line for deadlock-freedom in Chapter 6 and for CTL*–X formulae in Chapter 5. Thus, although such conditions indicate that not all systems can be verified by using them, we achieved our goals for an interesting class of systems.

8.1 Related Work

We addressed related work in each chapter: We reviewed various formal models for software components in Section 2.1.4 of Chapter 2, analyzed related (de-)composition operators in Section 3.6 of Chapter 3, dealt with architectures and restrictions of the cooperation structure in related formalisms in

Section 4.6 of Chapter 4, treated compositional reduction in other component-based settings in Section 5.5 of Chapter 5, discussed related work on efficient deadlock analysis in Section 6.4 of Chapter 6, and showed how various authors introduced ideas similar to port protocols in Section 7.3 of Chapter 7.

8.2 Future Work

We already gave some directions for future work at the end of each chapter. We particularly want to highlight the extension of our compositional reduction approach discussed in Section 5.6 of Chapter 5 and the extension of our approach for deadlock-freedom discussed in Section 6.6 of Chapter 6. Furthermore, there are some directions that we have not addressed yet and mention in the following paragraphs.

A dependable system has the ability to provide its services under severe conditions such as failure of system parts or presence of malicious cooperation partners. With respect to interaction systems, Majster-Cederbaum and Martens [178] address how a deadlock-free interaction systems preserves its deadlock-freedom if a certain set of actions is not available anymore, i.e., all interactions where such an action is part of are not executable any longer. The authors call deadlock-freedom in a given interaction system robust with respect to such a set of actions if the constrained system is deadlock-free as well. Moreover, Martens [190, Chapter 5] extended the deadlock detection approach for tree-like interaction systems [180] to check in polynomial time whether deadlock-freedom is robust with respect to a certain set of actions. Here, it is an interesting question whether our approach (cf. Chapter 6) can also be extended in this way to show that deadlock-freedom is robust in interaction systems with a disjoint circular wait free architecture.

Similar investigations can be found in the field of fault tolerance and reliable distributed computing where more detailed failure models such as crash-stop, crash-recovery, or byzantine are studied. Here, it is a challenging question whether such failure models can be incorporated in a component-based setting such as interaction systems. Our work on consensus algorithms [105, 106] already indicates how processes could be encapsulated as components with ports. Recently, Bonakdarpour et al. [44] studied such ideas in the context of BIP [30] for which interaction systems are a theoretical model.

Last but not least, a combination of the ideas in Chapter 5, where we investigated under which conditions a component can be removed from the analysis, and the ideas of Chapter 6, where we showed how deadlock-freedom can be established in polynomial time, is promising.

Appendix A

Preliminaries and Notation

In this appendix, we include various definitions for self-containedness of the thesis.

A.1 Labeled Transition Systems

We use labeled transition systems for any behavioral description in this thesis. Formally, they are defined as follows.

Definition A.1 (Labeled Transition System): A *labeled transition system LTS* is a tuple $(S, \Sigma, \{\xrightarrow{a}\}_{a\in\Sigma}, S^0)$ where S is a finite set of *states* which is also called the *state space*, Σ is an alphabet[1] containing the *labels*, $\{\xrightarrow{a}\}_{a\in\Sigma}$ is a family of *transition relations* where $\xrightarrow{a} \subseteq S \times S$ for each $a \in \Sigma$, and $S^0 \subseteq S$ is a set of *initial states*. Whenever $(s,t) \in \xrightarrow{a}$ for two states $s, t \in S$ and a label $a \in \Sigma$, we write $s \xrightarrow{a} t$ instead and say that there is an *a-transition* from s to t. A label $a \in \Sigma$ is called *enabled* in a state $s \in S$ if a state $t \in S$ exists such that $s \xrightarrow{a} t$. A transition $s \xrightarrow{a} s$ for a state $s \in S$ and a label $a \in \Sigma$ is called an *a-self-loop*, and in general, transitions relating the same state are called *self-loops*.

Since we often need to deal with the future or past behavior (as seen from a particular state), the following definition of two functions allows us to access this state information in a convenient way.

Definition A.2 (Predecessor and Successor): Let $LTS = (S, \Sigma, \{\xrightarrow{a}\}_{a\in\Sigma}, S^0)$ be a labeled transition system. For a state $s \in S$ and a set of labels $L \subseteq \Sigma$, we define the *L-predecessors* of s as $\mathrm{Pre}(s, L) := \{t \in S \mid \exists\, a \in L\colon t \xrightarrow{a} s\}$ and

[1] An alphabet is a finite set of symbols as in automata theory [145].

the *predecessors* of s as $\mathrm{Pre}(s) := \mathrm{Pre}(s, \Sigma)$. Similarly, the *L-successors* of s are defined as $\mathrm{Suc}(s, L) := \{t \in S \mid \exists\, a \in L\colon s \xrightarrow{a} t\}$ and the *successors* of s as $\mathrm{Suc}(s) := \mathrm{Suc}(s, \Sigma)$. Note that $\mathrm{Pre}(s, \emptyset) = \mathrm{Suc}(s, \emptyset) = \emptyset$ holds.

The second last of the preliminary definitions introduces the notion of a "path" that is used in various contexts throughout this thesis. The most common usage of this notion is in the area of graph theory (cf. for instance the book of Diestel [89, Section 1.3]).

Definition A.3 (Path and Maximal Path): A *path* over a tuple $(S, \mathcal{R})$, where S is a finite set and $\mathcal{R} \subseteq S \times S$ is a binary relation over S, is a finite or infinite, consecutively numbered sequence $\pi := \langle s_0, s_1, s_2, \ldots \rangle$ such that $\forall\, i \in \mathbb{N}\colon s_i \in \pi \implies s_i \in S$ and $\forall\, i \in \mathbb{N}\colon s_i \in \pi \wedge s_{i+1} \in \pi \implies (s_i, s_{i+1}) \in \mathcal{R}$. The length of a finite path, i.e., the number of its elements, is denoted by $|\pi|$; if π is an infinite path we write $|\pi| = |\mathbb{N}|$. By $\pi[i]$ we denote the ith element of the path—observe that a zeroth element exists—and by $\pi[i\,..]$ the suffix path starting with element s_i (for $i \in \mathbb{N}$ with $i < |\pi|$). We call a path *maximal* either if it is infinite, i.e., $|\pi| = |\mathbb{N}|$, or if $|\pi| > 0$ and for the last element no successor element exists, i.e., if $\pi = \langle s_0, \ldots, s_n \rangle$ for some $n \in \mathbb{N}$ and no $s_{n+1} \in S$ exists with $(s_n, s_{n+1}) \in \mathcal{R}$. By $\mathrm{MaxPaths}(s)$ for $s \in S$ we denote the set of all maximal paths π over $(S, \mathcal{R})$ with $\pi[0] = s$. A finite path π is called *simple*, if no element occurs twice, i.e., $\forall\, i, j \in \{0, \ldots, |\pi| - 1\}\colon i \neq j \implies \pi[i] \neq \pi[j]$.

Finally, the notion of paths allows us to define the well-known notion of reachability of states in labeled transition systems.

Definition A.4 (Reachable States): Let $LTS = (S, \Sigma, \{\xrightarrow{a}\}_{a \in \Sigma}, S^0)$ be a labeled transition system. A state $s \in S$ is called *reachable* in LTS if an initial state $s^0 \in S^0$ and a finite path π over $(S, \bigcup_{a \in \Sigma} \xrightarrow{a})$ exist such that $\pi[0] = s^0$ and $\pi[|\pi|-1] = s$ holds.

A.2 Graph Theory

The following definitions are common notation in graph theory. A more comprehensive introduction and overview can be found in the book of Diestel [89, Chapter 1].

Definition A.5 (Graph): A *graph* G is a tuple (V, E) where V is a finite set of *vertices* and E a set of *edges* which are 2-element subsets of V, i.e., $E \subseteq \{e \in 2^V \mid |e| = 2\}$. Two vertices $v, w \in V$ are called *adjacent* if an edge $e \in E$ exists with $e = \{v, w\}$. In this case, edge e and vertex v (and w respectively) are

called *incident*. Furthermore, the set of *neighbors* of a vertex v in G is denoted by $\text{nb}_G(v)$, i.e., $\text{nb}_G(v) := \{w \in V \mid \{v, w\} \in E\}$.

The definition of a path that we gave in the previous section is also valid for graphs as defined above because we can consider the set of edges as a symmetric binary relation over the set of vertices. Thus, if we reason about paths in a graph in the following, we think of paths as introduced with Definition A.3.

Definition A.6 (Connected Graph): A graph $G = (V, E)$ is said to be *connected* if for any two vertices $v, w \in V$ a finite path π over G with $\pi[0] = v$ and $\pi[|\pi| - 1] = w$ exists.

For connected graphs, we define the following properties.

Definition A.7 (Graph Properties): Let $G = (V, E)$ be a connected graph. The *distance* $\text{dist}_G(v, w)$ between two vertices v and w in G is the length of a shortest path between them minus one (since in graph theory, as opposed to our definition of a path, the edges are counted), i.e., $\text{dist}_G(v, w) := \min\{|\pi| - 1 \mid \pi \text{ is a finite path over } G \text{ with } \pi[0] = v \text{ and } \pi[|\pi| - 1] = w\}$. The *eccentricity* $\text{ecc}_G(v)$ of a vertex v is the maximum distance from v to any other vertex, i.e., $\text{ecc}_G(v) := \max\{\text{dist}_G(v, w) \mid w \in V\}$. The *radius* $\text{rad}(G)$ is the minimum eccentricity over all vertices in G, i.e., $\text{rad}(G) := \min\{\text{ecc}_G(v) \mid v \in V\}$. The *diameter* $\text{diam}(G)$ is the maximum eccentricity over all vertices in G, i.e., $\text{diam}(G) := \max\{\text{ecc}_G(v) \mid v \in V\}$. The *center* $\text{center}(G)$ contains all vertices v whose eccentricity equals the radius of G, i.e., $\text{center}(G) := \{v \in V \mid \text{ecc}_G(v) = \text{rad}(G)\}$. The *periphery* $\text{periphery}(G)$ contains all vertices v whose eccentricity equals the diameter of G, i.e., $\text{periphery}(G) := \{v \in V \mid \text{ecc}_G(v) = \text{diam}(G)\}$.

Finally, we define the notion of simple cycles in graphs.

Definition A.8 (Simple Cycle): A *simple cycle* in a graph $G = (V, E)$ is a finite path π of length at least four, starting and ending in a vertex $v \in V$, i.e., $\pi[0] = v$ and $\pi[|\pi| - 1] = v$ and $|\pi| > 3$, with the property that no vertex other than v occurs twice: $\forall\, i, j \in \{1, \ldots, |\pi| - 1\} \colon i \neq j \implies \pi[i] \neq \pi[j]$, i.e., the suffix path $\pi[1..]$ is a simple path. The set of vertices that lie on such a simple cycle is denoted by $\text{cycle}(G) \subseteq V$.

Appendix B

Pseudocode Algorithms

In this appendix, we give some of the algorithms mentioned throughout the thesis together with runtime analyses. The first algorithm initializes the information that we need in following algorithms.

Algorithm B.1 Initialization for all algorithms based on interaction systems

INITIALIZATION(Sys) // defines all variables as attributes of objects

```
1  for each component i ∈ Comp
2      for each state s ∈ S_i
3          for each action a ∈ A_i
4              s.Suc_a = ∅ // for setting Suc(s, a)
5              s.Pre_a = ∅ // for setting Pre(s, a)
6  for each component i ∈ Comp
7      for each state s ∈ S_i
8          for each action a ∈ A_i
9              for each state t ∈ S_i
10                 if (s, t) ∈ -a->_i // or: s -a->_i t
11                     s.Suc_a = s.Suc_a ∪ {t}
12                     t.Pre_a = t.Pre_a ∪ {s}
13 for each interaction α ∈ Int
14     α.compset = ∅ // for setting compset(α)
15     for each component i ∈ Comp
16         if not A_i ∩ α == ∅ // if i participates in α
17             α.compset = α.compset ∪ {i}
18             α.i = a  where {a} = A_i ∩ α // setting i(α) as action
19     α.closed = FALSE
20     if α ∈ Int_closed
21         α.closed = TRUE
```

Please note that we here consider algorithms that operate on interaction systems where we assume that the input is given as specified in Definition 2.5, i.e., we assume, as already done in Algorithm B.1 on the preceding page, that we have access to the sets *Comp*, *Act*, *Int*, Int_{closed}, and Int_{open} and for each component $i \in Comp$ we can access the labeled transition system corresponding to its local behavior $[\![i]\!] = (S_i, A_i, \{\xrightarrow{a}_i\}_{a \in A_i}, S_i^0)$. Observe that Algorithm B.1 initializes the following information: The functions Suc() and Pre() for every state and action (cf. Definition A.2) and the functions compset() and $i()$ for every interaction and component i (cf. Definition 2.2). An upper bound for the runtime of this algorithm is $O(|Comp| \cdot |S_{\max}|^2 \cdot |A_{\max}| + |Int| \cdot |Comp|)$ where $S_{\max}$ is the largest set of states among the components and $A_{\max}$ the largest action set. This bound is polynomial in the size of the input, which we highlighted as an important goal in the introduction of the thesis, i.e., we can use this algorithm in the following and can safely assume that the initialization is carried in polynomial time. Next, we take a look at the computation of the reachable global behavior of an interaction system.

Global Behavior Traversal

Algorithm B.2 on the facing page computes the reachable global behavior (cf. Definition 2.6) of an interaction system by traversing through the global state space. Here, we are not interested in a particular order of the components with respect to global state tuples. However for an algorithm that is executable by a computer, we have to use an operator that allows for considering an "order independent" Cartesian product, i.e., we cannot use the Cartesian product operator that is available in many programming languages.

Here, we denote such an "order independent" Cartesian product operator by "$\breve{\times}$" and use it to construct global states as in Definition 2.6—cf. its usage in line 4, line 24, and line 25 of Algorithm B.2. From an implementation point of view, we could for instance always use the same order of the components or use a set representation. Furthermore, we assume that an appropriate $\breve{\times}$-operator corresponds to an "unfolded" Cartesian product, i.e., the operator does not introduce the typical nested set structure that comes with a consecutive application of a Cartesian product. As an example, consider the sets $S_1 = \{s_1\}$, $S_2 = \{s_2\}$, and $S_3 = \{s_3, t_3\}$. If we now compute $S = S_1 \times S_2 = \{(s_1, s_2)\}$ and afterwards $S \times S_3$, we get $\{((s_1, s_2), s_3), ((s_1, s_2), t_3)\}$. For our new operator, we require that $(S_1 \breve{\times} S_2) \breve{\times} S_3$ equals $\{(s_1, s_2, s_3), (s_1, s_2, t_3)\}$ and that the order inside the tuples is not important but always the same. Note that such a computation is typically carried out in a loop, e.g., consider line 4 of Definition 2.6 where we compute the set of global initial states.

Algorithm B.2 Global behavior traversal

BEHAVIOR-TRAVERSAL(Sys)

```
1  INITIALIZATION(Sys) // cf. Algorithm B.1 on page 211
2  S^0 = ∅ // for setting the global initial states
3  for each component i ∈ Comp
4      S^0 = S^0 ⨯̆ S_i^0 // An "order independent", "unfolded" Cartesian product
5  S = ∅ // for setting the global states
6  --τ--> = ∅ // for setting the global transition relations
7  for each interaction α ∈ Int_open
8      --α--> = ∅
9  current = S^0
10 repeat // traverse the global state space
11     S = S ∪ current
12     found = ∅ // for saving the global states found in the current step
13     for each global state s ∈ current
14         s.Suc = ∅ // we only compute Suc(s) here (cf. Definition A.2)
15         s.Suc_τ = ∅ // for setting Suc(s, {τ})
16         s.Suc_ϯ = ∅ // for setting Suc(s, Int_open)
17         for each interaction α ∈ Int
18             successors = ∅
19             for each i ∈ Comp and i's local state s_i of s = (..., s_i, ...)
20                 if i ∈ α.compset
21                     if s_i.Suc_α.i == ∅
22                         successors = ∅
23                         break for loop over components (line 19)
24                     else successors = successors ⨯̆ s_i.Suc_α.i
25                 else successors = successors ⨯̆ {s_i}
26             found = found ∪ successors
27             if not successors == ∅
28                 if α.closed == TRUE
29                     --τ--> = --τ--> ∪ ({s} × successors)
30                     s.Suc_τ = s.Suc_τ ∪ successors
31                 else --α--> = --α--> ∪ ({s} × successors)
32                     s.Suc_ϯ = s.Suc_ϯ ∪ successors
33                 s.Suc = s.Suc ∪ successors
34     current = found \ S // keep only the new states for the next step
35 until current == ∅
36 return (S, Int_open^τ, {--α-->}_{α∈Int_open^τ}, S^0) // returns labeled transition system
```

Next, we demonstrate a few steps of Algorithm B.2 by means of the merchandise management example Sys_{MMS}. After the first for loop we have $S^0 = \{(s_c^0, s_m^0, s_s^0)\}$ (after line 4). Observe that this is the only global initial state, thus when the repeat loop in line 10 starts the set *current* contains only this global state. When the for loop in line 13 treats this global state and the for loop in line 17 treats the interaction $\alpha = \{ask_c, order_m\}$, the set *successors* is set to $s_c^0.Suc_{ask_c} \times s_m^0.Suc_{order_m} \times \{s_s^0\} = \{(s_c^1, s_m^1, s_s^0)\}$. Since this set is nonempty, we get for the α-transition relation $\xrightarrow{\alpha} = \{((s_c^0, s_m^0, s_s^0), (s_c^1, s_m^1, s_s^0))\}$ after line 31. Note that this is the only interaction that is enabled in the global initial state, i.e., for all other interactions the for loop of line 19 is terminated in line 23 because of a component whose action is not enabled in its current state.

We turn to the question of the runtime of Algorithm B.2. Since we check for every global state which interactions are enabled in the corresponding local states of the components, we have $O(|S| \cdot |Int| \cdot |Comp|)$ as an upper bound. This bound holds since each reachable global state is only considered once in the repeat loop in line 10 and in the for loop in line 13 respectively because such a state is part of the set *current* exactly once—this is also the reason why the successors $\text{Suc}(s)$ of a global state s are set correctly in line 14 and line 33. We already discussed in Section 2.2 that for the size of the global state space S we have $|S| \leq |S_{\max}|^{|Comp|}$ where $|S_{\max}|$ denotes the size of the largest local state space, i.e., $|S_{\max}| = \max\{|S_i| \mid i \in Comp\}$. However, this boils down to $O(|S_{\max}|^n)$ with $n = |Comp|$ as the upper runtime bound of Algorithm B.2, which clearly is not in polynomial time in the size of the input—if we neglect the factors $|Int|$ and $|Comp|$. Thus, this algorithm is only feasible for small parameters (such as the merchandise management example Sys_{MMS}).

Livelock Detection

The following algorithm inspects an interaction system with respect to livelock-freedom: Algorithm B.3 on the facing page works in two phases. First, the reachable states of the global behavior of the given interaction system are determined in line 1 which are then used to construct a directed graph where the vertices correspond to the states that have an outgoing τ-transition and no outgoing non-τ-transition (cf. lines 2–8). If we detect a state that possesses only a τ-self-loop (cf. line 5), we can answer the question of livelock-freedom negative since we already detected a reachable livelock. Afterwards, the edges are stored as an adjacency list in lines 9–12 where two vertices u, v are adjacent if there is a τ-transition from u to v. In this step, we can neglect the τ-self-loops since we already checked them while constructing the set of vertices. Then, the second phase begins where we have to check whether there is a

Algorithm B.3 Livelock detection

```
LIVELOCK-FREE(Sys)
 1  (S, Σ, {—a→}_{a∈Σ}, S^0) = BEHAVIOR-TRAVERSAL(Sys) // cf. Algorithm B.2
 2  Vertices = ∅                                       // on page 213
 3  for each global state s ∈ S
 4      if not s.Suc_τ == ∅ and s.Suc_τ' == ∅ // s has τ- and no non-τ-success.
 5          if s.Suc_τ == {s} // if s has only a τ-self-loop, s is a livelock
 6              return FALSE
 7          Vertices = Vertices ∪ {s}
 8          s.indegree = 0
 9  for each global state s ∈ Vertices // check for a cycle among the vertices
10      s.Adjacency = (s.Suc_τ ∩ Vertices) \ {s} // keep only relevant τ-tran.
11      for each global state t ∈ s.Adjacency
12          t.indegree = t.indegree + 1
13  Sources = CREATESTACK() // Sources is a common stack data structure
14  for each global state s ∈ Vertices
15      if s.indegree == 0
16          PUSH(Sources, s)
17  while not ISEMPTY(Sources) // successively remove vertices
18      s = POP(Sources)       // with indegree == 0
19      for t ∈ s.Adjacency
20          t.indegree = t.indegree − 1
21          if t.indegree == 0
22              PUSH(Sources, t)
23  for each global state s ∈ Vertices // If there is still a vertex with nonzero
24      if s.indegree > 0              // indegree, then also a cycle exists.
25          return FALSE
26  return TRUE
```

directed cycle among the vertices (cf. lines 13–22). Observe that we stored the indegree, i.e., the number of incoming edges, for every vertex. Now, as long as there is a vertex with indegree zero, we decrement the indegrees of all adjacent vertices and re-check (we keep a stack of non-checked vertices with indegree zero to avoid double checks). Afterwards, if there still is a vertex with an indegree greater than zero, then there must be a cycle in the original graph which corresponds to a cycle of τ-transitions in the global behavior (cf. lines 23–25 where we inspect the indegree of all vertices). This lets us decide the livelock-freedom of the interaction system in question.

The runtime of the second phase of Algorithm B.3 is linear in the number of vertices and edges of the graph constructed in the first phase because we

check each of these entities at most once. But, the construction of the graph is directly connected to the construction of the global state space (with a call of Algorithm B.2 in line 1 of Algorithm B.3). Since the size of the corresponding labeled transition system can be exponential in the number of components, hence also the runtime of Algorithm B.3.

Composition Information Validity

Algorithm B.4 checks whether the set I^+ is valid with respect to Definition 3.4. The comments provide an explanation of this check.

Algorithm B.4 Validity check of the set I^+ of a composition information

CHECK-NEW-INTERACTIONS$(I^+, Sys^{(1)}, Sys^{(2)}, \ldots, Sys^{(n)})$

```
1  for each new interaction α ∈ I+
2      counter = 0 // Counts the number of participating systems
3      for i = 1 to n
4          β = α ∩ Act(i) // The actions of Sys(i) that are contained in α
5          α = α \ β // Update α to contain only non-processed actions
6          valid = TRUE
7          if not β == ∅ // Sys(i) participates in the new interaction α
8              counter = counter + 1
9              valid = FALSE
10             for each interaction γ ∈ Int(i)
11                 if β ⊆ γ // Check whether β ∈ ⋃_{γ∈Int(i)} 2^γ holds
12                     valid = TRUE
13                     break for loop over interactions (line 10)
14         if not valid
15             return FALSE
16         if α == ∅ // Sys(i) is the last system participating in α
17             if counter ≤ 1 // Check whether α ∈ ⋃_{1≤i≤n} ⋃_{γ∈Int(i)} 2^γ
18                 return FALSE
19             break for loop over interaction systems (line 3)
20     if not α == ∅ // An element is contained in α that is not an action
21         return FALSE
22 return TRUE
```

The runtime bound of Algorithm B.4 is polynomial in the size of the input since we loop for all new interactions through the set of interactions of the n input interaction systems, i.e., the bound is $O(|I^+| \cdot n \cdot |Int^{(\max)}|)$ where $Int^{(\max)}$ denotes the largest interaction set.

Computing the Cooperation Graph

Algorithm B.5 yields the cooperation graph G_{coop} of an interaction system *Sys* (cf. Definition 4.4).

Algorithm B.5 Computation of the cooperation graph

COOPERATION-GRAPH(*Sys*)

```
 1  INITIALIZATION(Sys) // cf. Algorithm B.1 on page 211
 2  Vertices = ∅
 3  Edges = ∅
 4  for each interaction α ∈ Int // Add each set of participating components
 5      Vertices = Vertices ∪ {α.compset} // to the set of vertices.
 6  Tmp = ∅
 7  for each vertex u ∈ Vertices // We consider all pairs of sets of compo-
 8      for each vertex v ∈ Vertices // nents participating in an interaction.
 9          Tmp = Tmp ∪ {u ∩ v}
10  Vertices = Vertices ∪ (Tmp \ {∅})
11  for each component i ∈ Comp // Add each component as a singleton to
12      Vertices = Vertices ∪ {{i}} // the set of vertices.
13  for each vertex v ∈ Vertices
14      v.nb = ∅
15  for each vertex u ∈ Vertices
16      for each vertex v ∈ Vertices
17          if u ⊂ v // Is u adjacent to v?
18              adjacent = TRUE
19              for each vertex w ∈ Vertices // Does a proper subset of v
20                  if u ⊂ w and w ⊂ v      // strictly contain u?
21                      adjacent = FALSE
22                      break for loop over vertices (line 19)
23              if adjacent == TRUE
24                  Edges = Edges ∪ {{u, v}}
25                  u.nb = u.nb ∪ {v}
26                  v.nb = v.nb ∪ {u}
27  return (Vertices, Edges)
```

The most time consuming part of this algorithm is in lines 15–26 where we loop three times through the set of vertices in order to determine the edges. Thus, the runtime bound of the algorithm is $O(|V|^3)$ where V denotes the set of vertices of the cooperation graph. As we learned in Section 4.3, we here have $|V| \in O(|Int|^2 + |Comp|)$, i.e., the runtime of this algorithm is polynomial in the size of the interaction system given as input.

Disjoint Circular Wait Free Architectures

Algorithm B.6 implements Theorem 4.9 which checks whether a given interaction system has disjoint circular wait free architecture.

Algorithm B.6 Check for a disjoint circular wait free architecture

```
DCWF(Sys)
 1  (V, E) = COOPERATION-GRAPH(Sys) // cf. Algorithm B.5 on page 217
 2  if not GRAPH-IS-CONNECTED(V, E) // cf. [76, Section 22.3]
 3      return FALSE
 4  V' = ∅
 5  E' = ∅
 6  for each vertex v ∈ V
 7      V' = V' ∪ {v_in, v_out}
 8      E' = E' ∪ {(v_in, v_out)} // directed edges are ordered pairs
 9  for each edge {v, w} ∈ E // edges are 2-element subsets of vertices
10      E' = E' ∪ {(v_out, w_in), (w_out, v_in)}
11  CAPACITY = λ x, y: (x, y) ∈ E' ? 1 : 0 // λ yields a function
12  for each component i ∈ Comp with v = {i}, v ∈ V, and v_in, v_out ∈ V'
13      if |v.nb| ≥ 2
14          CAPACITY-V = λ x, y: (x, y) == (v_in, v_out) ? 2 : CAPACITY(x, y)
15          for each component j ∈ Comp with w = {j}, w ∈ V, w_in ∈ V'
16              if not i == j and |w.nb| ≥ 2
17                  if MAXIMUM-FLOW(V', E', v_in, w_in, CAPACITY-V) ≥ 2
18                      return FALSE // ↪ network, source, sink, capac.
19  return TRUE
```

We already addressed the runtime of Algorithm B.6 in the discussion of an application of Theorem 4.9 in Section 4.3.2.

Computation of the Cycle Components

Algorithm B.7 on the facing page computes the set of cycle components in an efficient way.

In line 4, we use an algorithm by Tarjan [251] for finding all bridges in time $O(|V| + |E|)$. The runtime of this algorithm is linear in the number of vertices and edges of the cooperation graph—if we neglect its construction—since each of these entities is at most considered once in a loop and the classification in line 4 can also be carried out in linear time.

Algorithm B.7 Computation of the cycle components

CYCLE-COMPONENTS(Sys)

```
1  (V, E) = COOPERATION-GRAPH(Sys) // cf. Algorithm B.5 on page 217
2  for each vertex v ∈ V
3      v.degree = |v.nb|
4  Classify each edge e ∈ E as bridge: e.bridge is set to TRUE or FALSE [251]
5  for each edge e ∈ E with e = {u, v} and u, v ∈ V
6      if e.bridge == TRUE
7          u.degree = u.degree − 1
8          v.degree = v.degree − 1
9  Cycle-Comp = ∅
10 for each component i ∈ Comp with v = {i} and v ∈ V
11     if v.degree > 1
12         Cycle-Comp = Cycle-Comp ∪ {i}
13 return Cycle-Comp
```

Ensuring Exclusive Communication

Algorithm B.8 on the following page ensures the property of exclusive communication (cf. Definition 5.1) for interaction systems as defined in this thesis (cf. Definition 2.5).

We omit a detailed cost analysis of Algorithm B.8 but it obviously runs in polynomial time in the size of the input since every interaction is considered once and this consideration possibly adds new labels and transitions for the behavior of components whose number is bounded by the size of the currently considered interaction.

Algorithm B.8 Ensuring exclusive communication

EXCLUSIVE(Sys)

1 INITIALIZATION(Sys) // cf. Algorithm B.1 on page 211
2 **for** each component $i \in Comp$
3 $\quad A_i^{\text{new}} = \emptyset$
4 $\quad LTS_i^{\text{new}} = (S_i, \emptyset, \emptyset, S_i^0)$ // where $[\![i]\!] = (S_i, A_i, \{\xrightarrow{a}_i\}_{a \in A_i}, S_i^0)$
5 $Int^{\text{new}} = \emptyset$
6 $Int_{\text{closed}}^{\text{new}} = \emptyset$
7 **for** each interaction $\alpha \in Int$
8 $\quad \alpha^{\text{new}} = \emptyset$
9 $\quad$ **for** each component $i \in \alpha.compset$
10 $\quad\quad a^{\text{new}} = \alpha.i^{\alpha.compset}$ // each action $a \in \alpha$ becomes $a^{\text{compset}(\alpha)}$
11 $\quad\quad A_i^{\text{new}} = A_i^{\text{new}} \cup a^{\text{new}}$
12 $\quad\quad \alpha^{\text{new}} = \alpha^{\text{new}} \cup a^{\text{new}}$
13 $\quad\quad$ **for** each transition $s_i \xrightarrow{\alpha.i}_i t_i$ of $[\![i]\!]$
14 $\quad\quad\quad$ add a label a^{new} to LTS_i^{new}
15 $\quad\quad\quad$ add a transition $s_i \xrightarrow{a^{\text{new}}}_i t_i$ to LTS_i^{new}
16 $\quad Int^{\text{new}} = Int^{\text{new}} \cup \alpha^{\text{new}}$
17 $\quad$ **if** $\alpha.closed$ == TRUE
18 $\quad\quad Int_{\text{closed}}^{\text{new}} = Int_{\text{closed}}^{\text{new}} \cup \alpha^{\text{new}}$
19 **return** $(((Comp, \{A_i^{\text{new}}\}_{i \in Comp}), Int^{\text{new}}, Int_{\text{closed}}^{\text{new}}), \{LTS_i^{\text{new}}\}_{i \in Comp})$

Appendix C

Polynomial-Space Algorithms for Freedom from Deadlock and Freedom from Livelock

In this appendix, we want to address the issue of detecting deadlocks and livelocks *deterministically* in polynomial space. Note that we already know that this is possible for deadlocks since, as discussed in Chapter 2, the decision problem can be solved nondeterministically in polynomial space and together with Savitch's famous theorem [240], PSPACE = NPSPACE, the claim follows. However, it is interesting to ask whether we can use polynomial-space algorithms in practice, and we have to give a polynomial-space algorithm for the deciding livelock-freedom to establish its PSPACE-completeness (cf. Section 2.3.2).

According to Lipton [174, Chapter 29], Savitch's theorem [240] uses a key idea that was introduced in a proof by Lewis II et al. [172] about the recognition of a context-free language in logarithmic space. However, it was Savitch who realized the important consequence of the idea for the collapse of the complexity classes PSPACE and NPSPACE.

In the following, we give several algorithms that all work with respect to a given interaction system *Sys*. In some cases, we omit to pass *Sys* as a parameter for better readability, i.e., only the two following Algorithms C.5 and C.7 are called with *Sys* as a parameter and all other algorithms are called from inside these and thus have access to *Sys* as a global parameter. We also assume that interaction system *Sys* is initialized with respect to Algorithm B.1 on page 211. Observe that this initialization does not occupy more than polynomial space. We begin with a polynomial-space algorithm for deadlock-freedom.

C.1 Freedom from Deadlock

The first item we have to determine is whether a certain global state is a deadlock. Algorithm C.1 realizes this check in the most space efficient way by checking wether an interaction is enabled, which answers the question of deadlock-freedom for the state in question affirmative.

Algorithm C.1 Determines whether a given global state is a deadlock

```
DEADLOCK(s) // global state s
 1  for each interaction α ∈ Int
 2      enabled = TRUE
 3      for each i ∈ Comp and i's local state s_i of s
 4          if i ∈ α.compset
 5              if s_i.Suc_{α.i} == ∅
 6                  enabled = FALSE
 7                  break for loop over components (line 3)
 8      if enabled == TRUE
 9          return FALSE
10  return TRUE
```

The next step is to apply Algorithm C.1 to all reachable global states. Here, we need to be able to compute the set of global states in a space efficient way. The following algorithm realizes this with the concept of an iterator that only stores one global state at a time.

Algorithm C.2 Global state iterator

```
GLOBAL-STATE-ITERATOR()
 1  current = 0
 2  max = 1
 3  for each component i ∈ Comp
 4      max = max · |S_i|
 5  while current < max
 6      global-state = ∅ // we use an ordered set to construct an n-tuple
 7      divisor = 1
 8      for each component i ∈ Comp
 9          global-state = global-state ∪ S_i[⌊current/divisor⌋ mod |S_i|]
10          divisor = divisor · |S_i| // ↪ S_i[k] yields the (k+1)th state of i
11      yield the global state that corresponds to global-state
12      current = current + 1
```

The main idea of Algorithm C.2 is that we do not compute the Cartesian

product that corresponds to the global state space directly, which would waste too much space, but instead compute its elements sequentially in a lazy way. Here, we assume that each set of local states has a fixed order and is accessible as in typical arrays, i.e., $S_i[k]$ yields the $(k+1)$th local state of component i for $0 \leq k \leq |S_i| - 1$. We yield each global state in line 11 of Algorithm C.2.

Since a global state is reachable if there is a path starting in a global initial state, we need to iterate over all global initial states in a space efficient way. We use the same idea as for Algorithm C.2 to derive the following algorithm.

Algorithm C.3 Global initial state iterator

GLOBAL-INITIAL-STATE-ITERATOR()

```
1  current = 0
2  max = 1
3  for each component i ∈ Comp
4      max = max · |S_i^0|
5  while current < max
6      global-initial-state = ∅
7      divisor = 1
8      for each component i ∈ Comp
9          global-initial-state = global-initial-state ∪ S_i^0[⌊current/divisor⌋ mod |S_i^0|]
10         divisor = divisor · |S_i^0| // ↪ S_i^0[k] yields the (k+1)th init. state
11     yield the global state that corresponds to global-initial-state
12     current = current + 1
```

Next, we can address the reachability question. In Algorithm C.4 on the following page, we use the typical idea as in Savitch's theorem [240]: A global state t is reachable from another global state s in at most k transitions if there is a global state r that is reachable from s in at most $\lfloor \frac{k}{2} \rfloor$ transitions and t is reachable from r in at most $\lceil \frac{k}{2} \rceil$ transitions. The key insight for the space consumption of Algorithm C.4 is that the recursion depth is logarithmic in the number of global states since each call halves the parameter k. Moreover, we only need to store three global states and an integer in each recursion level and, only on the lowest level, iterators over the interactions and components respectively. Clearly, all these variables occupy only logarithmic space. Thus, the overall space requirement is squared logarithmical in the number of global states, which meets our polynomial-space bound.

Finally, we can derive a polynomial-space algorithm for deadlock-freedom, which is given as Algorithm C.5 on the following page. Thereby, we use the overall number of global states as parameter k for the reachability analysis since at most all global states are intermediate states on any path.

Algorithm C.4 Polynomial-space reachability analysis

REACHABLE(s, t, k) // global states s, t and integer k

```
1  if k ≤ 1
2      for each interaction α ∈ Int
3          executable = TRUE
4          for each i ∈ Comp and i's local state s_i of s and t_i of t
5              if i ∈ α.compset
6                  if t_i ∉ s_i.Suc_{α.i}
7                      executable = FALSE
8                      break for loop over components (line 4)
9              elseif not t_i == s_i
10                 executable = FALSE
11                 break for loop over components (line 4)
12         if executable == TRUE
13             return TRUE
14     return s == t
15 else for each global state r in GLOBAL-STATE-ITERATOR()
16         if REACHABLE(s, r, ⌊k/2⌋) and REACHABLE(r, t, ⌈k/2⌉)
17             return TRUE
18     return FALSE
```

Algorithm C.5 Polynomial-space deadlock detection

PSPACE-DEADLOCK-FREE(Sys)

```
1  max = 1
2  for each component i ∈ Comp
3      max = max · |S_i|
4  for each global initial state s^0 in GLOBAL-INITIAL-STATE-ITERATOR()
5      for each global state s in GLOBAL-STATE-ITERATOR()
6          if DEADLOCK(s) and REACHABLE(s^0, s, max)
7              return FALSE
8  return TRUE
```

C.2 Freedom from Livelock

For a polynomial-space livelock detection, we can proceed similarly as above. A certain global state is a livelock (cf. Definition 2.9), if it is reachable from a global initial state, if in all global states that are reachable over a sequence of τ-transitions no open interaction is enabled, and if the global state itself is reachable (again) by a nonempty sequence of τ-transitions. We can already

answer the first part with Algorithm C.4 in polynomial space and thus need a space-efficient way for the two latter parts.

Algorithm C.6 answers the question whether a global state t is reachable from a global state s in at most k steps and polynomial space by only using τ-transitions. Observe that we meet our polynomial-space bound for $k \leq 1$ by looping through all closed interactions and sets of local states of the participating components. For larger k, the halving of the parameter ensures that the recursion depth stays logarithmic in k—analogously as discussed above for Algorithm C.4.

Algorithm C.6 Polynomial-space τ reachability analysis

```
TAU-REACH(s, t, k) // global states s, t and integer k
 1  if k ≤ 1
 2      tau-executable = FALSE
 3      for each closed interaction α ∈ Int_closed
 4          executable = TRUE
 5          for each i ∈ Comp and i's local state s_i of s and t_i of t
 6              if i ∈ α.compset
 7                  if t_i ∉ s_i.Suc_{α.i}
 8                      executable = FALSE
 9                      break for loop over components (line 5)
10              elseif not t_i == s_i
11                  executable = FALSE
12                  break for loop over components (line 5)
13          if executable == TRUE
14              tau-executable = TRUE
15              break for loop over interactions (line 3)
16      return s == t or tau-executable
17  else for each global state r in GLOBAL-STATE-ITERATOR()
18          if TAU-REACH(s, r, ⌊k/2⌋) and TAU-REACH(r, t, ⌈k/2⌉)
19              return TRUE
20      return FALSE
```

Now, in order to decide whether a given interaction is livelock-free in polynomial space, we need to consider each reachable global state, ensure that it has no (global) successor that is reachable over a sequence of τ-transitions where an open interaction is enabled, and whether the global state in question is reachable (again) from its successors by τ-transitions only (which ensures that there is at least one such transition). Note that we employ the same idea as for Algorithm C.5 to keep the space bound by checking all paths up to a certain length k. Algorithm C.7 on the following page implements this task.

Algorithm C.7 Polynomial-space livelock detection

```
PSPACE-LIVELOCK-FREE(Sys)
 1  max = 1
 2  for each component i ∈ Comp
 3      max = max · |S_i|
 4  for each global initial state s^0 in GLOBAL-INITIAL-STATE-ITERATOR()
 5      for each global state s in GLOBAL-STATE-ITERATOR()
 6          if REACHABLE(s^0, s, max)
 7              livelock = TRUE
 8              reachable-again = FALSE
 9              for each global state t in GLOBAL-STATE-ITERATOR()
10                  if TAU-REACH(s, t, max) // i.e., if s -τ->* t, check
11                      non-tau-enabled = FALSE // Suc(t, Int_open) = ∅
12                      for each open interaction α ∈ Int_open
13                          enabled = TRUE
14                          for each i ∈ Comp and i's local state t_i of t
15                              if i ∈ α.compset
16                                  if t_i.Suc_{α.i} == ∅
17                                      enabled = FALSE
18                                      break for loop in line 14
19                          if enabled == TRUE
20                              non-tau-enabled = TRUE
21                              break for loop in line 12
22                      if non-tau-enabled == TRUE
23                          livelock = FALSE
24                          break for loop in line 9
25                      if reachable-again == FALSE // check s -τ->+ s
26                          if s == t  // check for a τ-self-loop
27                              Insert lines 2–15 of Algorithm C.6
28                              reachable-again = tau-executable
29                          else reachable-again = TAU-REACH(t, s, max)
30              if livelock and reachable-again
31                  return FALSE
32  return TRUE
```

Note that we have to ensure that the currently considered global state has a τ-self-loop in lines 26–28 because a global state is always reachable in at most k steps from itself and we thus cannot use Algorithm C.6 as in line 29.

Together with the PSPACE-hardness established in Section 2.3.2 of Chapter 2, we can thus conclude that livelock detection is PSPACE-complete.

Appendix D

Computation Tree Logic

In this appendix, we give the formal definition for the syntax and semantics of Extended Computation Tree Logic. Note that a detailed introduction, discussion, and references for this topic are given in Section 2.3.3 of Chapter 2. Some further remarks can be found in Section E.4 of the following appendix.

Definition D.1 (Syntax of CTL*): Let Σ be an alphabet with $\tau \notin \Sigma$ called *atomic propositions*. A *CTL* state formula* Φ over Σ, also briefly called *CTL* formula*, is syntactically defined via the grammar:

$$\Phi ::= \top \mid p \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathsf{E}\,\phi$$

where $p \in \Sigma$, Φ_1, Φ_2 are CTL* state formulae over Σ, and ϕ is a *CTL* path formula* over Σ which is syntactically defined via the grammar (where ϕ_1, ϕ_2 are CTL* path formulae over Σ):

$$\phi ::= \Phi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}\,\phi \mid \phi_1\,\mathsf{U}\,\phi_2$$

Definition D.2 (Auxiliary Notation for CTL*): As usual, we define the following operators (with the operator precedence $\neg, \mathsf{E}, \mathsf{A}, \mathsf{U}, \mathsf{F}, \mathsf{G}, \wedge, \vee, \Rightarrow, \Leftrightarrow$) and symbols for CTL* path formulae ϕ, ϕ_1, ϕ_2 and CTL* formulae Φ, Φ_1, Φ_2:

- $\bot := \neg\top$
- $\phi_1 \vee \phi_2 := \neg(\neg\phi_1 \wedge \neg\phi_2)$
- $\Phi_1 \vee \Phi_2 := \neg(\neg\Phi_1 \wedge \neg\Phi_2)$
- $\phi_1 \Rightarrow \phi_2 := \neg\phi_1 \vee \phi_2$
- $\Phi_1 \Rightarrow \Phi_2 := \neg\Phi_1 \vee \Phi_2$
- $\phi_1 \Leftrightarrow \phi_2 := \phi_1 \Rightarrow \phi_2 \wedge \phi_2 \Rightarrow \phi_1$
- $\Phi_1 \Leftrightarrow \Phi_2 := \Phi_1 \Rightarrow \Phi_2 \wedge \Phi_2 \Rightarrow \Phi_1$
- $\mathsf{A}\,\phi := \neg\mathsf{E}\,\neg\phi$
- $\mathsf{F}\,\phi := \top\,\mathsf{U}\,\phi$
- $\mathsf{G}\,\phi := \neg\mathsf{F}\,\neg\phi$
- $\bigvee_{i\in\{1,\ldots,n\}} \phi_i := \phi_1 \vee \ldots \vee \phi_n$
- $\bigvee_{i\in\{1,\ldots,n\}} \Phi_i := \Phi_1 \vee \ldots \vee \Phi_n$

- $\bigwedge_{i\in\{1,\ldots,n\}} \phi_i := \phi_1 \wedge \ldots \wedge \phi_n$
- $\bigwedge_{i\in\{1,\ldots,n\}} \Phi_i := \Phi_1 \wedge \ldots \wedge \Phi_n$
- $\bigvee_{i\in\emptyset} \phi_i := \bot$
- $\bigvee_{i\in\emptyset} \Phi_i := \bot$
- $\bigwedge_{i\in\emptyset} \phi_i := \top$
- $\bigwedge_{i\in\emptyset} \Phi_i := \top$

For the empty disjunction and conjunction, we have $\bigvee_{i\in\emptyset} \phi_i = \bot$ whereas $\bigwedge_{i\in\emptyset} \phi_i = \top$ holds. This is reasonable because a disjunction corresponds to an existential quantification whereas a conjunction to a universal one.

Definition D.3 (Kripke Structure): A *Kripke structure* is defined as a tuple $KS := (S, \longrightarrow, S^0, \Sigma, \mathcal{L})$ where S is a finite sets of *states*, $\longrightarrow \subseteq S \times S$ is a *transition relation*, $S^0 \subseteq S$ is the set of *initial states*, Σ is an alphabet containing the *atomic propositions*, and $\mathcal{L}\colon S \to 2^\Sigma$ is a function that labels each state with a set of atomic propositions. Whenever $(s,t) \in \longrightarrow$ for two states $s, t \in S$ we write $s \longrightarrow t$ instead.

Definition D.4 (Satisfaction Relation for CTL*): Let $KS = (S, \longrightarrow, S^0, \Sigma, \mathcal{L})$ be a Kripke structure and let $p \in \Sigma^\prime$ be an atomic proposition, $s \in S$ be a state, ϕ, ϕ_1, ϕ_2 be CTL* path formulae over $\Sigma^\prime$ and Φ, Φ_1, Φ_2 be CTL* state formulae over $\Sigma^\prime$. The satisfaction relation $\models$ for CTL* state formulae is inductively defined as (where $KS, s \models \top$ always holds):

$$
\begin{array}{lll}
KS, s \models p & \text{iff} & p \in \mathcal{L}(s) \\
KS, s \models \neg\Phi & \text{iff} & KS, s \not\models \Phi \\
KS, s \models \Phi_1 \wedge \Phi_2 & \text{iff} & KS, s \models \Phi_1 \ \wedge \ KS, s \models \Phi_2 \\
KS, s \models \mathsf{E}\,\phi & \text{iff} & \exists\, \pi \in \mathrm{MaxPaths}(s)\colon KS, \pi \models \phi
\end{array}
$$

For any maximal path π over KS (cf. Definition A.3), satisfaction relation $\models$ for CTL* path formulae is defined as

$$
\begin{array}{lll}
KS, \pi \models \Phi & \text{iff} & KS, \pi[0] \models \Phi \\
KS, \pi \models \neg\phi & \text{iff} & KS, \pi \not\models \phi \\
KS, \pi \models \phi_1 \wedge \phi_2 & \text{iff} & KS, \pi \models \phi_1 \ \wedge \ KS, \pi \models \phi_2 \\
KS, \pi \models \mathsf{X}\,\phi & \text{iff} & |\pi| > 1 \ \wedge \ KS, \pi[1\,..] \models \phi \\
KS, \pi \models \phi_1\,\mathsf{U}\,\phi_2 & \text{iff} & \exists\, i \in \mathbb{N}\colon i < |\pi| \ \wedge \ KS, \pi[i\,..] \models \phi_2 \\
 & & \quad \wedge\ (\forall\, j \in \mathbb{N}\colon j < i \implies KS, \pi[j\,..] \models \phi_1)
\end{array}
$$

Definition D.5 (Equivalence of CTL* Formulae): Two CTL* formulae Φ_1 and Φ_2 over an alphabet $\Sigma^\prime$ are *equivalent*, denoted by $\Phi_1 \equiv \Phi_2$, if for every Kripke structure $KS = (S, \longrightarrow, S^0, \Sigma, \mathcal{L})$ and every state $s \in S$ it holds that

$$KS, s \models \Phi_1 \text{ if and only if } KS, s \models \Phi_2.$$

Appendix E

Bisimilarities

In this appendix, we formally define the five equivalences mentioned in Section 2.4. There, they are only briefly discussed and after a comparison branching bisimilarity with explicit divergence is chosen to be our "working" behavioral equivalence for interaction systems (cf. Definition 2.15). Here, we give formal definitions based on labeled transition systems (in the notation of this thesis, cf. Definition A.1), shortly present their historical origin, and discuss the asymptotic runtime of their computation. We also address their expressive power, i.e., what kind of properties are preserved in equivalent systems, by means of a logical characterization.

We want to mention that for their computation, i.e., an algorithm that decides whether two systems are equivalent, the known algorithms usually compute the largest/coarsest auto-equivalence of a single labeled transition system, i.e., they relate the states in *one* given system. If we are now interested in deciding the equivalence of two systems, the first step is to construct a "union system" which serves as the input for the auto-equivalence algorithm.

This union is defined in the natural way: Let LTS_1 and LTS_2 be labeled transition systems over the same alphabet, i.e., $LTS_i = (S_i, \Sigma, \{\xrightarrow{a}_i\}_{a\in\Sigma}, S_i^0)$ for $i = 1, 2$. The union of the two systems is the labeled transition system:

$$LTS = (S_1 \cup S_2, \Sigma, \{\xrightarrow{a}_1 \cup \xrightarrow{a}_2\}_{a\in\Sigma}, S_1^0 \cup S_2^0).$$

Now, the labeled transition systems LTS_1 and LTS_2 are equivalent if there is a corresponding auto-equivalence relation $\mathcal{R}$ for LTS such that $\forall\, s_1^0 \in S_1^0 \;\; \exists\, s_2^0 \in S_2^0 \colon (s_1^0, s_2^0) \in \mathcal{R}$ and, vice versa, $\forall\, s_2^0 \in S_2^0 \;\; \exists\, s_1^0 \in S_1^0 \colon (s_2^0, s_1^0) \in \mathcal{R}$—this works because $\mathcal{R}$ is symmetric. Additionally, since these algorithms usually compute the largest/coarsest equivalence relation, they can also be used to compute quotients of labeled transition systems, as introduced in our setting for branching bisimilarity with explicit divergence by Definition 2.16. We refer

the reader to the book of Baier and Katoen [23, Chapter 7] for a more detailed treatment.

Typically, the number of states and the number of transitions of the input labeled transition system are the influencing parameters for the runtime costs of such algorithms, which is in our case the union of the two systems in question. For the sake of better readability, we use the abbreviations $n = |S_1| + |S_2|$ and $m = \sum_{a \in \Sigma}(|\xrightarrow{a}_1| + |\xrightarrow{a}_2|)$ in the following.

E.1 Strong Bisimilarity

First, we give a formal definition of *strong bisimilarity* with respect to labeled transition systems.

Definition E.1 (Strong Bisimilarity): Let $LTS_i = (S_i, \Sigma, \{\xrightarrow{a}_i\}_{a \in \Sigma}, S_i^0)$, $i = 1, 2$, be labeled transition systems over the same alphabet. The systems are *strongly bisimilar*, denoted by $LTS_1 \approx_s LTS_2$, if a symmetric relation $\mathcal{R} \subseteq S_1 \times S_2 \cup S_2 \times S_1$ exists such that for $i = 1, 2$ and $j = (i \bmod 2) + 1$ holds

1. for all states $s_i^0 \in S_i^0$ exists a state $s_j^0 \in S_j^0$ such that $(s_i^0, s_j^0) \in \mathcal{R}$ and
2. for all $(s_i, s_j) \in \mathcal{R}$ with $s_i \in S_i$ and $s_j \in S_j$, all $t_i \in S_i$, and all $a \in \Sigma$, if $s_i \xrightarrow{a}_i t_i$ then there is a state $t_j \in S_j$ with $s_j \xrightarrow{a}_j t_j$ and $(t_i, t_j) \in \mathcal{R}$.

The notion of bisimilarity was independently invented by Park [222] and Milner [197] where Milner applied the notion under the name strong equivalence in the context of his CCS [200]. The two ideas are the same if image finiteness is assumed [86], i.e., the number of successors of all states is finite, and later, Milner [197] adopted Park's notation. Milner was also involved in the first logical characterization of his equivalence where Hennessy-Milner logic [135] was found to characterizes strong bisimilarity [136]. Hennessy-Milner logic can directly be interpreted over labeled transition systems, i.e., we do not need to translate labeled transition systems into Kripke structures as for CTL* (cf. Section 2.3.3), however, it is weaker than CTL* which means that there are properties that can be expressed in CTL* but not in Hennessy-Milner logic—note that there is no fixed-point operator in the latter logic [247, Remark 2]. Browne et al. [53] give a logical characterization of strong bisimilarity in CTL*, where they apply Milner's notion of strong bisimilarity in the context of Kripke structures. Note that with the translation of De Nicola and Vaandrager [85], this applies also for labeled transition systems—in the cited work, the authors only talk about CTL*–X but the translation is also valid for CTL* because it preserves strong bisimilarity [238].

For the computation of strong bisimilarity, the first algorithm goes back to Kanellakis and Smolka [152]—according to Baier and Katoen [23, Section 7.10]. The main idea behind this algorithm is partition refinement: Start with an initial partition of the state space and try to refine these partitions, usually called blocks, until all equivalent states are in the same block. The refinement is done by so-called splitters that separate the states in a block by their distinguishable behavior. This technique has its origins in automata theory in the context of minimization, e.g., the algorithm by Hopcroft [143], where equivalent states with respect to the relation by Myhill [209] and Nerode [211] are computed. For strong bisimilarity, the partition refinement algorithm of Paige and Tarjan [219] improved the one by Kanellakis and Smolka [152] and settled the upper bound of $O(m \cdot \log n)$ if the number of labels is fixed and $m \geq n$ holds—a very detailed overview can be found in the book of Baier and Katoen [23, Section 7.3.4]. Dovier et al. [93] discuss improvements and symbolic implementations of this algorithm. The question of the runtime behavior of a strong bisimilarity checking algorithm without the assumption that $|\Sigma|$ is constant was recently addressed by Valmari [257]. The author shows that the $O(m \cdot \log n)$ bound is also valid in this case.

E.2 Weak Bisimilarity

The notion of strong bisimilarity has the drawback that all behavior is considered as observable. For instance, a sequence of τ-transition can be fully observed and each single τ step can be counted. The existence of unobservable behavior thus needs a special treatment that was introduced in the broad work of Robin Milner [135, 136, 195–200]. Before we further discuss the historical origin, we give a formal definition of *weak bisimilarity* with respect to labeled transition systems.

Definition E.2 (Weak Bisimilarity): Let $LTS_i = (S_i, \Sigma^\tau, \{\xrightarrow{a}_i\}_{a \in \Sigma^\tau}, S_i^0)$, $i = 1, 2$, be labeled transition systems over the same alphabet. The systems are *weakly bisimilar*, denoted by $LTS_1 \approx_w LTS_2$, if a symmetric relation $\mathcal{R} \subseteq S_1 \times S_2 \cup S_2 \times S_1$ exists such that for $i = 1, 2$ and $j = (i \bmod 2) + 1$ holds

1. for all states $s_i^0 \in S_i^0$ exists a state $s_j^0 \in S_j^0$ such that $(s_i^0, s_j^0) \in \mathcal{R}$ and
2. for all $(s_i, s_j) \in \mathcal{R}$ with $s_i \in S_i$ and $s_j \in S_j$, all $t_i \in S_i$, and all $a \in \Sigma^\tau$, if $s_i \xrightarrow{a}_i t_i$ then
   (a) either $a = \tau$ and a state $t_j \in S_j$ exists with $s_j \xrightarrow{\tau}_j^* t_j$ and $(t_i, t_j) \in \mathcal{R}$
   (b) or there are states $q_j, r_j, t_j \in S_j$ with $s_j \xrightarrow{\tau}_j^* q_j$, $q_j \xrightarrow{a}_j r_j$, $r_j \xrightarrow{\tau}_j^* t_j$, and $(t_i, t_j) \in \mathcal{R}$.

Originally, the notion of weak bisimilarity was introduced as observational equivalence in the presence of unobservable actions by Hennessy and Milner [135, 136]; however, Milner [198] also seems to be the first to use the name weak bisimilarity. In their original definition, the authors define this equivalence over sequences of actions, although Milner [199, Proposition 8.4] shows that it is possible to restrict these sequences to be of length one, which then coincides with Definition E.2. Similar concepts were discussed by Bergstra and Klop [37] and by Baeten and Van Glabbeek [22]. The failure equivalence of Brookes et al. [51] in the context of CSP [141] also specially treats unobservable behavior although it is coarser than weak bisimilarity. We refer the reader to Van Glabbeek [113, Section 4] for a good overview. A logical characterization of weak bisimilarity can be found in Milner's book on CSS [200, Chapter 10], where he introduces weak modalities for Hennessy-Milner logic [135].

We turn to the question of computing weak bisimilarity. Typical algorithms first compute the reflexive transitive closure of the τ-transition relation, which yields the double arrow relation of CCS [197], and then use the algorithm for strong bisimilarity, e.g., the algorithm derived by Kanellakis and Smolka [152]. The former part can be achieved by the Floyd–Warshall algorithm [100, 261] in runtime $O(n^3)$, or for a better worst case upper bound, by an algorithm by Munro [208] that exploits fast matrix multiplication, which in turn can be computed in runtime $O(n^{2.376})$ with an algorithm by Coppersmith and Winograd [75]. The closure computation then dominates the runtime over the strong bisimilarity computation. More recently, this over twenty years old bound was improved independently by Stothers [248] to $O(n^{2.3737})$ and later by Vassilevska Williams [259] to $O(n^{2.3727})$. In the latter work, the author also conjectures that more tighter estimates are very likely. However, these algorithms are only of theoretical interest although they yield the best worst case upper bound that is known today. We refer the reader to the thesis of Nuutila [214] for efficient algorithms addressing this problem. However, as pointed out by Groote and Vaandrager [130, Section 6.2], the above runtime is only valid if the number of labels is fixed, otherwise the upper bound becomes $O(m \cdot n^{2.3727})$.

E.3 Branching Bisimilarity

One drawback of weak bisimilarity is that it does not preserve the branching structure of labeled transition systems. Figure 2.11 on page 53 already illustrated this non-preservation: The depicted systems are (pairwise) weakly bisimilar but in the first system, there is a computation path that always offers α before β is executed whereas in the other systems such a path does not exists.

A better example is given by Van Glabbeek and Weijland [115, Figure 1] where the same computation paths in three different (weakly bisimilar) systems offer different in-between options. These kind of observations, which origin from Van Glabbeek and Weijland [114], led to the introduction of branching bisimilarity, which we define next.

Definition E.3 (Branching Bisimilarity): Let $LTS_i = (S_i, \Sigma^\tau, \{\xrightarrow{a}_i\}_{a\in\Sigma^\tau}, S_i^0)$, $i = 1, 2$, be labeled transition systems over the same alphabet. The systems are *branching bisimilar*, denoted by $LTS_1 \approx_b LTS_2$, if a symmetric relation $\mathcal{R} \subseteq S_1 \times S_2 \cup S_2 \times S_1$ exists such that for $i = 1, 2$ and $j = (i \bmod 2) + 1$ holds

1. for all states $s_i^0 \in S_i^0$ exists a state $s_j^0 \in S_j^0$ such that $(s_i^0, s_j^0) \in \mathcal{R}$ and
2. for all $(s_i, s_j) \in \mathcal{R}$ with $s_i \in S_i$ and $s_j \in S_j$, all $t_i \in S_i$, and all $a \in \Sigma^\tau$, if $s_i \xrightarrow{a}_i t_i$ then
   (a) either $a = \tau$ and $(t_i, s_j) \in \mathcal{R}$
   (b) or there are states $r_j, t_j \in S_j$ with $s_j \xrightarrow{\tau}_j^* r_j$, $r_j \xrightarrow{a}_j t_j$, $(s_i, r_j) \in \mathcal{R}$, and $(t_i, t_j) \in \mathcal{R}$.

We want to mention that a proof that branching bisimilarity is an equivalence relation was addressed by Basten [29] after some "obvious" proofs turned out to be incorrect. De Nicola and Vaandrager [85] studied logical characterizations of branching bisimilarity and show that CTL*–X interpreted over all paths instead of maximal paths (as introduced in Appendix D) is suitable for such a characterization. Their work is based on stuttering equivalence of Kripke structures studied by Browne et al. [53].

Computational issues were addressed by Groote and Vaandrager [130] which derive an algorithm that works similarly to the approach for strong bisimilarity mentioned in Section E.1: After an initial partition of the state space into blocks, the blocks are refined by splitters (with respect to a certain label) until no more refinements are possible. Such a splitter with respect to a label a for a block B is the set of states that can be reached from the states in B with an a-transition after performing a finite number of τ-transitions that enter only states in B as intermediate states. We refer the reader to the detailed discussion in the work by Groote and Vaandrager [130] that also addresses implementation issues. The runtime of this algorithm is $O(m \cdot \log m + m \cdot n)$ or $O((|\Sigma| + m) + m \cdot n)$ depending on whether one uses heapsort or bucket sort respectively in the construction of the initial partition—note that the set of labels needs not to be fixed. Otherwise, the runtime is $O(m \cdot n)$. The authors conjecture that this bound can be improved, but best to our knowledge, no better worst case upper bound is known today.

E.4 Divergence Sensitive Branching Bisimilarity

Although the interpretation of CTL* formulae over all paths of Kripke structures was originally used by its inventors, Emerson and Halpern [95], this interpretation is only reasonable for total Kripke structures, i.e., structures in which all states have at least one successor. An example that illustrates this issue is given by De Nicola and Vaandrager [85, Example 3.2.8]. Nowadays, the maximal path interpretation became accepted, e.g., Baier and Katoen [23, Remark 6.11, page 329] allow it, although, for instance, Clarke et al. [72] require totality of the transition relation but then use an infinite path interpretation—remember that maximal paths and infinite paths coincide in this case. An additional reason for using the maximal path interpretation is the ability to describe properties like fairness [85]. If one now wants to use CTL*–X interpreted over maximal paths (as we do in Appendix D) to serve as a logical characterization, branching bisimilarity needs to be strengthened to be sensitive to divergent paths, i.e., maximal paths where eventually only τ-transitions occur, which lead to the introduction of divergence sensitive branching bisimilarity. This new bisimilarity was invented by De Nicola and Vaandrager [85] where they originally introduced a special state in both labeled transition systems in question such that every state in one of the original systems, that lies on a cycle of τ-transitions or has no outgoing transition, is equipped with a transition to the new state. This translation allows to relate livelocks in the systems since De Nicola and Vaandrager [85] define the new bisimilarity as: If the new systems are branching bisimilar, the original systems are divergence sensitive branching bisimilar. Here, we modify this definition to skip the addition of the special state and give a more "direct" definition.

Definition E.4 (Divergence Sensitive Branching Bisimilarity): Let $LTS_i = (S_i, \Sigma^\tau, \{\xrightarrow{a}_i\}_{a\in\Sigma^\tau}, S_i^0)$, $i = 1, 2$, be labeled transition systems over the same alphabet. The systems are *divergence sensitive branching bisimilar*, denoted by $LTS_1 \approx_b^\lambda LTS_2$, if a symmetric relation $\mathcal{R} \subseteq S_1 \times S_2 \cup S_2 \times S_1$ exists such that for $i = 1, 2$ and $j = (i \bmod 2) + 1$ holds

1. for all states $s_i^0 \in S_i^0$ exists a state $s_j^0 \in S_j^0$ such that $(s_i^0, s_j^0) \in \mathcal{R}$ and
2. for all $(s_i, s_j) \in \mathcal{R}$ with $s_i \in S_i$ and $s_j \in S_j$ it holds that
   - 2.1. for all $t_i \in S_i$ and all $a \in \Sigma^\tau$, if $s_i \xrightarrow{a}_i t_i$ then
     - (a) either $a = \tau$ and $(t_i, s_j) \in \mathcal{R}$
     - (b) or there are states $r_j, t_j \in S_j$ with $s_j \xrightarrow{\tau}_j^* r_j$, $r_j \xrightarrow{a}_j t_j$, $(s_i, r_j) \in \mathcal{R}$, and $(t_i, t_j) \in \mathcal{R}$ and
   - 2.2. if there is an infinite path π over $(S_i, \xrightarrow{\tau}_i)$ such that $\pi[0] = s_i$,

then there exists a state $t_j \in S_j$ such that $s_j \xrightarrow{\tau}_j t_j$ or $t_j = s_j$ if $\mathrm{Suc}(s_j) = \emptyset$ and $(t_i, t_j) \in \mathcal{R}$ where $t_i = \pi[k]$ for some $k \in \mathbb{N}$.

We omit a formal proof that Definition E.4 coincides with the original one mentioned above, since Part 2.2 exactly captures those states that are treated in the translation of De Nicola and Vaandrager [85]. Finally with this new bisimilarity, De Nicola and Vaandrager [85, Theorem 3.4.6] show that divergence sensitive branching bisimilarity is logically characterized by CTL*–X interpreted over maximal paths.

The computation of divergence sensitive branching bisimilarity can be performed with a slight modification of the algorithm by Groote and Vaandrager [130] that computes branching bisimilarity as addressed in the previous section. We shortly address this modification here since we did not find it in the literature. Before the initial partition is computed, the algorithm [130] computes the strongly connected components of the input transition system restricted to inert τ-transitions (interpreted as a directed graph where the edges correspond to the τ-transitions). Then, the transition system is condensed with respect to these strongly connected components, i.e., the states in such a connected component become a single new state where all incoming and outgoing transitions are redirected to the new state. Since branching bisimilarity does not preserve divergence, no τ-self-loop is added to a new state. If we now want to preserve divergence, we just have to add such a self-loop (if the state has at least one successor, cf. Part 2.2 of Definition E.4) as a non-inert transition in the terminology of the implementation of Groote and Vaandrager [130]. Note that this modification does not influence the asymptotic runtime of the algorithm, which thus yields the same runtime as discussed in the previous section.

E.5 Branching Bisimilarity with Explicit Divergence

In Chapter 3, we discussed the desired behavior that a behavioral equivalence in interaction systems is a congruence with respect to the composition and closing operator. Otherwise, the composition of two systems is not guaranteed to have the expected outcome if one of the systems is replaced by a behavioral equivalent one. For divergence sensitive branching bisimilarity such an undesired behavior was observed by Van Glabbeek et al. [117, Section 5] where they introduce a merge operator for labeled transition systems that corresponds to process algebraic parallel operators. An example [117, Figure 4] then illustrates the undesired behavior, where two states of a labeled transition system are auto-equivalent with respect to divergence sensitive branching

bisimilarity, but after the merge operator is applied, this equivalence vanishes. The authors show that a stronger equivalence, viz. branching bisimilarity with explicit divergence, overcomes this problem. This equivalence was defined by Van Glabbeek and Weijland [114] in the spirit of Bergstra et al. [38]. Next, we define this equivalence in our setting.

Definition E.5 (Branching Bisimilarity with Explicit Divergence): Let $LTS_i = (S_i, \Sigma^\tau, \{\xrightarrow{a}_i\}_{a\in\Sigma^\tau}, S_i^0)$, $i = 1, 2$, be labeled transition systems over the same alphabet. The systems are *branching bisimilar with explicit divergence*, denoted by $LTS_1 \approx_b^\Delta LTS_2$, if a symmetric relation $\mathcal{R} \subseteq S_1 \times S_2 \cup S_2 \times S_1$ exists such that for $i = 1, 2$ and $j = (i \bmod 2) + 1$ holds

1. for all states $s_i^0 \in S_i^0$ exists a state $s_j^0 \in S_j^0$ such that $(s_i^0, s_j^0) \in \mathcal{R}$ and
2. for all $(s_i, s_j) \in \mathcal{R}$ with $s_i \in S_i$ and $s_j \in S_j$ it holds that
   - 2.1. for all $t_i \in S_i$ and all $a \in \Sigma^\tau$, if $s_i \xrightarrow{a}_i t_i$ then
     - (a) either $a = \tau$ and $(t_i, s_j) \in \mathcal{R}$
     - (b) or there are states $r_j, t_j \in S_j$ with $s_j \xrightarrow{\tau}_j^* r_j$, $r_j \xrightarrow{a}_j t_j$, $(s_i, r_j) \in \mathcal{R}$, and $(t_i, t_j) \in \mathcal{R}$ and
   - 2.2. if there is an infinite path π over $(S_i, \xrightarrow{\tau}_i)$ such that $\pi[0] = s_i$, then there exists a state $t_j \in S_j$ such that $s_j \xrightarrow{\tau}_j t_j$ and $(t_i, t_j) \in \mathcal{R}$ where $t_i = \pi[k]$ for some $k \in \mathbb{N}$.

One drawback of the switch from the divergence sensitive version to the one with explicit divergence lies in the logical characterization. This was also addressed by Van Glabbeek et al. [117], where the authors discuss a modification of the translation from labeled transition systems to Kripke structures or alternatively, a modification of the logic CTL*–X such that it is able to distinguish deadlocks and livelocks. As we discussed in Section 2.4.1, this can also be achieved by ensuring that the labeled transition system in question is deadlock-free.

The computation of branching bisimilar with explicit divergence goes along the lines of the modification of the algorithm by Groote and Vaandrager [130] described in the previous section. The only difference is that we add τ-self-loops to all condensed states instead of only to the ones that have at least one successor. Thus, we also get the same runtime as for computing branching bisimilarity.

Appendix F

Formal Proofs

F.1 Proofs from Chapter 2

Proof of Lemma 2.10: First observe that the global behaviors of the two systems are "nearly identical": The additional component x in Sys' introduces for every global state of $[\![Sys']\!]$ a self-loop that is labeled with τ—since $\{live_x\}$ is a closed interaction—and all τ-transitions of $[\![Sys]\!]$ are labeled with the corresponding open interactions in $[\![Sys']\!]$. All other transitions can be found in both systems. Now, assume that Sys is deadlock-free. Then, for all reachable global states s of $[\![Sys]\!]$ holds $\mathrm{Suc}(s) \neq \emptyset$ (cf. Definition 2.8). Thus, for all corresponding global states s' of $[\![Sys']\!]$ also holds $\mathrm{Suc}(s') \neq \emptyset$ and that they are reachable. Since we have $\mathrm{Suc}(s', \{\tau\}) = \{s'\}$, i.e., the only τ-transition is the one by component x, we know that $\mathrm{Suc}(s', Int'_{\mathrm{open}}) \neq \emptyset$ holds, i.e., there is a successor that is reachable via an open interaction—otherwise s would be a deadlock. But this means that no such state s' can be a livelock, thus Sys' is livelock-free (cf. Definition 2.9). Analogously, we can show the other direction, i.e., assume that Sys' is livelock-free and conclude the deadlock-freedom of Sys. Finally, the translation itself can clearly be carried out in constant time if we assume that we work on a given copy of Sys or directly modify the system. ∎

Proof of Lemma 2.17: Let $LTS = (S, \Sigma^\tau, \{\xrightarrow{a}\}_{a\in\Sigma^\tau}, S^0)$ be the given system. We distinguish LTS and its quotient by $(S_1, \Sigma_1^\tau, \{\xrightarrow{a}_1\}_{a\in\Sigma_1^\tau}, S_1^0) = LTS_{\approx_b^\Delta}$ and $(S_2, \Sigma_2^\tau, \{\xrightarrow{a}_2\}_{a\in\Sigma_2^\tau}, S_2^0) = LTS$ in the following. Observe that the alphabets of the two systems are equal by definition, i.e., $\Sigma_1^\tau = \Sigma_2^\tau =: \Sigma^\tau$. We now have to show that a relation $\mathcal{R}' \subseteq S_1 \times S_2 \cup S_2 \times S_1$ exists such that $LTS_{\approx_b^\Delta} \approx_b^\Delta LTS$ holds. The definition of the quotient $LTS_{\approx_b^\Delta}$ (cf. Definition 2.16) already pro-

vides a symmetric relation $\mathcal{R} \subseteq S \times S$—observe that $S = S_2$ holds—that establishes $LTS \approx_b^\Delta LTS$. We define $\mathcal{R}' := \{([s_2]_\mathcal{R}, s_2) \mid s_2 \in S_2\}^\leftrightarrow$ and show that $\mathcal{R}'$ is indeed a relation that establishes the branching bisimilarity with explicit divergence of $LTS_{\approx_b^\Delta}$ and LTS—remember that S_1 consists of equivalence classes of the states in S_2 with respect to $\mathcal{R}$. Clearly, $\mathcal{R}'$ is symmetric since we applied the symmetric closure. Now, for all initial states $s_i^0 \in S_i^0$ exists a state $s_j^0 \in S_j^0$ such that $(s_i^0, s_j^0) \in \mathcal{R}'$ for $i = 1, 2$ and $j = (i \bmod 2) + 1$ because each initial state of LTS is contained in an equivalence class by definition. This shows the first part of Definition E.5. Next, we consider the second part. For $i = 1, 2$, $j = (i \bmod 2) + 1$, and all $(s_i, s_j) \in \mathcal{R}'$ with $s_i \in S_i$ and $s_j \in S_j$ holds

2.1. for all $t_i \in S_i$ and all $a \in \Sigma^\tau$, if $s_i \xrightarrow{a}_i t_i$ then $s \xrightarrow{a} t$ holds for two states $s, t \in S$ with $s \in s_i$ and $t \in t_i$ if $i = 1$ and $s = s_i$ and $t = t_i$ if $i = 2$.

   (a) If $a = \tau$ and $(s, t) \in \mathcal{R}$, then we have $(t_i, s_j) \in \mathcal{R}'$ by definition.

   (b) Otherwise, we know that a state $r \in S$ exists such that $s \xrightarrow{\tau}{}^* r$, $r \xrightarrow{a} t$, and $(s, r) \in \mathcal{R}$ holds. But then, we also find $r_j, t_j \in S_j$ with $r_j \ni r$ and $t_j \ni t$ if $j = 1$ and $r_j = r$ and $t_j = t$ if $j = 2$ such that $s_j \xrightarrow{\tau}{}_j^* r_j$, $r_j \xrightarrow{a}_j t_j$, and $(s_i, r_j) \in \mathcal{R}'$ holds because s and r are in the same equivalence class. Thus, also $(t_i, t_j) \in \mathcal{R}'$ holds by definition of $\mathcal{R}'$ and because $\mathcal{R}$ establishes $LTS \approx_b^\Delta LTS$.

2.2. if there is an infinite path π over $(S_i, \xrightarrow{\tau}_i)$ such that $\pi[0] = s_i$, then we find this path also over $(S, \xrightarrow{\tau})$. Thus, we know that a state $t \in S$ exists with $s \xrightarrow{\tau}{}^+ t$, $(s, t) \in \mathcal{R}$, and $t \xrightarrow{\tau}{}^* s$ since the state space is finite. Now, we also find a state $t_j \in S_j$ such that $s_j \xrightarrow{\tau}_j t_j$ and $(t_i, t_j) \in \mathcal{R}'$ where $t_i \in S_i$ is a state with $t_i = [t]_\mathcal{R}$ if $i = 1$ and $t_i = t$ if $i = 2$ (cf. Definition 2.16).

Thus, the relation $\mathcal{R}'$ establishes the branching bisimilarity with explicit divergence of the systems in question, i.e., $LTS_{\approx_b^\Delta} \approx_b^\Delta LTS$ holds. ∎

F.2 Proofs from Chapter 3

Proof of Proposition 3.6: We have to show that the interaction model constructed by the composition operator, $\bigotimes_{(I^+, I^-)}\{IM^{(1)}, \ldots, IM^{(n)}\}$, is indeed a valid interaction model. Observe that this is sufficient—for the proof that the tuple is an interaction system—since for all other entities of an interaction system such as the sets of components, actions, and labeled transition systems, the validity is guaranteed by the disjointness of the single systems—because these entities are interrelated to each other by set union. Since the composition operator joins the closed interactions by set union and no closed interaction

is removed from the overall interaction set, we can focus on this set of interactions. We have to show two things: First, that every interaction contains at most one action of every component, and second, that any action is contained in at least one interaction (cf. Definitions 2.2 and 2.3). For the first claim, we have to consider only the set I^+ since the claim already holds for all $Int^{(i)}$ with $1 \leq i \leq n$. The set I^+ is constructed via the powerset interjoin operator which combines sets from its operands by set union. Since the first claim already holds for all of these operands, the set union of the powerset interjoin does not add actions of one component twice. For the second claim, we have to consider the set I^- which removes interactions from the combined interaction set. But for this removal, we have the requirement that $I^- \sqsubseteq I^+$ holds, i.e., any interaction that is removed is contained in at least one new interaction. This guarantees the validity of the second claim. Thus, the composition operator yields a valid interaction system. ∎

Proof of Proposition 3.7: In order to show the proposition, we have to prove two claims: First, that each (binary) composition step is indeed a valid composition according to Definition 3.4, and second, that the resulting interaction sets on both sides of the equation of the proposition are equal. Note that the latter claim is sufficient for the equality, since the sets of components, actions, closed interactions, and labeled transition systems are not altered by the composition operator (cf. Definition 3.4).

We prove the first claim via induction over the current composition step with system $Sys^{(i)}$ for $2 \leq i \leq n$. Consider an arbitrary composition step where the next binary composition is between $Sys^{(i)}$ with $2 \leq i \leq n$ and the composed system of the previous steps. We can assume that all previous composition steps are valid. Let $Int^{(1,\ldots,i-1)}$ denote the resulting interaction set of the previous compositions. The current step is valid if (cf. Definition 3.4):

1. $I^+_{1,\ldots,i} \subseteq Int^{(1,\ldots,i-1)}_{\text{open}} \bowtie Int^{(i)}_{\text{open}}$,
2. $I^-_{1,\ldots,i} \subseteq Int^{(1,\ldots,i-1)}_{\text{open}} \cup Int^{(i)}_{\text{open}}$, and
3. $I^-_{1,\ldots,i} \sqsubseteq I^+_{1,\ldots,i}$ holds.

The first item follows because $I^+_{1,\ldots,i}$ contains only interactions α where components of $Sys^{(i)}$ participate in—since $\alpha \cap Act^{(i)} \neq \emptyset$ is required—but not exclusively, i.e., there is at least one component from the systems $1, \ldots, i-1$ that also participates—since $\alpha \nsubseteq Act^{(i)}$ is required. This is exactly the requirement for a powerset interjoin of two sets (cf. Definition 3.2). The second item is also true because $I^-_{1,\ldots,i}$ contains interactions α either from $Int^{(1,\ldots,i-1)}_{\text{open}}$ since $\exists\, \beta \in I^+_{1,\ldots,i} \colon \alpha = \beta \setminus Act^{(i)}$, or from $Int^{(i)}_{\text{open}}$ since $\exists\, \beta \in I^+_{1,\ldots,i} \colon \alpha = \beta \cap Act^{(i)}$, i.e.,

$\alpha \in Int_{\text{open}}^{(1,\ldots,i-1)} \cup Int_{\text{open}}^{(i)}$. The last item follows because, as for the second item, any interaction in $I_{1,\ldots,i}^{-}$ is contained in an interaction in $I_{1,\ldots,i}^{+}$: The α is always included in the quantified β in $I_{1,\ldots,i}^{-}$. This proves that any binary composition step is indeed a valid one.

For the second claim, we have to show that

$$\begin{aligned}&\left(I^+ \cup Int^{(1)} \cup \ldots \cup Int^{(n)}\right) \setminus I^- \stackrel{?}{=} \\ &\quad I_{1,\ldots,n}^{+} \cup \left[\left(\ldots\left[\left(I_{1,2}^{+} \cup Int^{(1)} \cup Int^{(2)}\right) \setminus I_{1,2}^{-}\right]\ldots\right] \cup Int^{(n)}\right) \setminus I_{1,\ldots,n}^{-} \qquad (\star)\end{aligned}$$

where the left-hand side is the interaction set resulting from the composition of all systems, and the right-hand side corresponds to the set of interactions resulting from the single binary compositions applied in the order given by the parentheses on the right-hand side of the equation of the proposition.

We first show that the resulting set of interactions of the binary compositions, i.e., the right-hand side of Equation ($\star$), can be rewritten as:

$$\begin{aligned}&I_{1,\ldots,n}^{+} \cup \left[\left(\ldots\left[\left(I_{1,2}^{+} \cup Int^{(1)} \cup Int^{(2)}\right) \setminus I_{1,2}^{-}\right]\ldots\right] \cup Int^{(n)}\right) \setminus I_{1,\ldots,n}^{-} \\ &\quad = \Big(\bigcup_{2\leq i\leq n} I_{1,\ldots,i}^{+} \cup Int^{(1)} \cup \ldots \cup Int^{(n)}\Big) \setminus \bigcup_{2\leq i\leq n} I_{1,\ldots,i}^{-} \qquad (*)\end{aligned}$$

This rewriting is valid because any interaction that is removed via set difference on the left-hand side of Equation ($*$) is not included in later (with respect to the order induced by the parentheses) unions, i.e., for any index i with $2 \leq i \leq n$, and any interaction $\alpha \in I_{1,\ldots,i}^{-}$ it holds that for all k with $i+1 \leq k \leq n$ holds $\alpha \notin I_{1,\ldots,k}^{+} \wedge \alpha \notin Int^{(k)}$. This statement holds because any such α is not included in an $I_{1,\ldots,k}^{+}$ with a higher index $k > i$ since these sets only consist of interactions where components from $Sys^{(k)}$ participate in which is not the case for sets with smaller indices. Analogously, this holds for any set $Int^{(k)}$ which only adds interactions of a system with a higher index (that could not have been removed earlier) for any such k.

Next, we prove two auxiliary equations which we apply to Equation ($*$) afterwards. It holds that

$$\bigcup_{2\leq i\leq n} I_{1,\ldots,i}^{+} \setminus \bigcup_{2\leq i\leq n} I_{1,\ldots,i}^{-} = I^+ \quad \text{and} \qquad (\dagger)$$

$$\left(Int^{(1)} \cup \ldots \cup Int^{(n)}\right) \setminus \bigcup_{2\leq i\leq n} I_{1,\ldots,i}^{-} = \left(Int^{(1)} \cup \ldots \cup Int^{(n)}\right) \setminus I^-. \qquad (\ddagger)$$

We prove Equation ($\dagger$): First, observe that $I^+ \subseteq \bigcup_{2\leq i\leq n} I_{1,\ldots,i}^{+}$ holds because for any index i, the composite interactions where components of interaction system $Sys^{(i)}$ and of all systems with smaller indices participate in are included in $I_{1,\ldots,i}^{+}$. We have to show that the union of the parametrized old interactions

($I^-_{1,\dots,i}$) contains all interactions that are not contained in I^+ but in the union of the parametrized new interactions ($I^+_{1,\dots,i}$). Note that no interaction from I^+ is removed because for any such α, that is removed, it is required that either $\alpha \notin I^+$, or $\alpha \in I^-$ and $I^+ \cap I^- = \emptyset$ holds. Assume that α is an arbitrary interaction that is contained in $\bigcup_{2\leq i\leq n} I^+_{1,\dots,i}$ but not in I^+. Observe that any such α is present in the union because an index i exists such that components from $Sys^{(i)}$ participate in α and also an index k with $k > i$ exists such that components from $Sys^{(k)}$ participate in an interaction β with $\alpha \subset \beta$, because otherwise $\alpha \in I^+$ (if no such β exists). Assume that k is the largest of such indices with the property that no index j with $i < j < k$ exists such that components from $Sys^{(j)}$ participate in β. Note that we can always find these two indices i and k with no index j in between, because $\alpha \notin I^+$ and the components from $Sys^{(k)}$ belong to the last system (with respect to the order induced by the parentheses) that need α for cooperation—if an j in between exists, then another set that strictly contains α is used for this cooperation, i.e., this index j is already the largest one. We have $\beta \in I^+_{1,\dots,k}$ since components of $Sys^{(k)}$ participate in it. Now, we also have $\alpha \in I^-_{1,\dots,k}$, since $\alpha = \beta \setminus Act^{(k)}$, $\alpha \notin I^+$, and $\forall\, \gamma \in I^+ \colon \alpha \subseteq \gamma \implies \beta \subseteq \gamma$ since k is the largest of such indices. Note that also $\alpha \notin Act^{(j)}$ holds for any $j < k$, since otherwise, α is not contained in $\bigcup_{2\leq i\leq n} I^+_{1,\dots,i}$. Thus, any such interaction α is contained in $\bigcup_{2\leq i\leq n} I^-_{1,\dots,i}$ which proves Equation (†).

For a proof of Equation (‡), observe that any interaction α that is removed from $Int^{(1)} \cup \ldots \cup Int^{(n)}$ by $\bigcup_{2\leq i\leq n} I^-_{1,\dots,i}$ is also contained in I^- because for any index i such an α is contained in either $I^-_{1,\dots,i}$ if $\alpha \subseteq Act^{(j)}$ and $\alpha \in I^-$ for any index $j < i$, or $\alpha \subseteq Act^{(i)}$ and $\alpha \in I^-$. Since also $I^- \subseteq \bigcup_{2\leq i\leq n} I^-_{1,\dots,i}$ holds—because $I^- \sqsubseteq I^+$, $I^+ \subseteq \bigcup_{2\leq i\leq n} I^+_{1,\dots,i}$, and $\exists\, \beta \in I^+_{1,\dots,i} \colon \alpha = \beta \cap Act^{(i)} \wedge \alpha \in I^-$ for all i—the claim of Equation (‡) follows.

We apply Equations (†) and (‡) to Equation (∗), which yields:

$$
\begin{aligned}
&\Big(\bigcup_{2\leq i\leq n} I^+_{1,\dots,i} \cup Int^{(1)} \cup \ldots \cup Int^{(n)}\Big) \setminus \bigcup_{2\leq i\leq n} I^-_{1,\dots,i} \\
&= \Big(\bigcup_{2\leq i\leq n} I^+_{1,\dots,i} \setminus \bigcup_{2\leq i\leq n} I^-_{1,\dots,i}\Big) \cup \big(Int^{(1)} \cup \ldots \cup Int^{(n)}\big) \setminus \bigcup_{2\leq i\leq n} I^-_{1,\dots,i} \\
&= \big(I^+\big) \cup \big(Int^{(1)} \cup \ldots \cup Int^{(n)}\big) \setminus I^- \\
&= \big(I^+ \setminus I^-\big) \cup \big(Int^{(1)} \cup \ldots \cup Int^{(n)}\big) \setminus I^- && \text{(since } I^+ \cap I^- = \emptyset\text{)} \\
&= \big(I^+ \cup Int^{(1)} \cup \ldots \cup Int^{(n)}\big) \setminus I^-
\end{aligned}
$$

This proves Equation (⋆), i.e., the set of interactions of both systems in the equation of Proposition 3.7 are equal, which proves the statement. ∎

Proof of Proposition 3.9: The composition information only consists of sets that have no order information, i.e., it holds that $Int^{(1)}_{\text{open}} \bowtie Int^{(2)}_{\text{open}} = Int^{(2)}_{\text{open}} \bowtie Int^{(1)}_{\text{open}}$ and $Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}} = Int^{(2)}_{\text{open}} \cup Int^{(1)}_{\text{open}}$. ■

Proof of Proposition 3.10: First, we show that the two binary compositions on the left-hand side of the equation of the proposition can be contracted into a single composition, i.e., that

$$\left(Sys^{(1)} \underset{(I^+_{1,2}, I^-_{1,2})}{\otimes} Sys^{(2)}\right) \underset{(I^+_{1,2,3}, I^-_{1,2,3})}{\otimes} Sys^{(3)} = \bigotimes_{(I^+, I^-)} \{Sys^{(1)}, Sys^{(2)}, Sys^{(3)}\} \quad (\star)$$

holds. Consider the set of interactions resulting on the left-hand side of Equation ($\star$). According to Definition 3.4, there we have:

$$\begin{aligned}
&\left(I^+_{1,2,3} \cup \left((I^+_{1,2} \cup Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}}) \setminus I^-_{1,2}\right) \cup Int^{(3)}_{\text{open}}\right) \setminus I^-_{1,2,3} \\
&= \left(I^+_{1,2,3} \cup I^+_{1,2} \cup Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}} \cup Int^{(3)}_{\text{open}}\right) \setminus \left(I^-_{1,2,3} \cup I^-_{1,2}\right) \\
&= \left(\left((I^+_{1,2} \cup I^+_{1,2,3}) \setminus I^-_{1,2,3}\right) \cup Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}} \cup Int^{(3)}_{\text{open}}\right) \setminus \left((I^-_{1,2} \cup I^-_{1,2,3}) \setminus I^+_{1,2}\right) \\
&= \left(I^+ \cup Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}} \cup Int^{(3)}_{\text{open}}\right) \setminus I^- \qquad (*)
\end{aligned}$$

where $I^+ = (I^+_{1,2} \cup I^+_{1,2,3}) \setminus I^-_{1,2,3}$ and $I^- = (I^-_{1,2} \cup I^-_{1,2,3}) \setminus I^+_{1,2}$ are defined in the proposition. Note that the penultimate line is correct because $I^-_{1,2} \cap I^+_{1,2} = \emptyset$ holds. Now, we showed that the last line of Equation ($*$) equals the set of interactions after the contracted composition step above, i.e., the set of interactions resulting on the right-hand side of Equation ($\star$). Additionally, we have to show that this composition is valid, i.e., that

1. $I^+ \subseteq Int^{(1)}_{\text{open}} \bowtie Int^{(2)}_{\text{open}} \bowtie Int^{(3)}_{\text{open}}$,
2. $I^- \subseteq Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}} \cup Int^{(3)}_{\text{open}}$, and
3. $I^- \sqsubseteq I^+$ holds.

Case 1 directly follows from the definition of I^+ because $I^+ = (I^+_{1,2} \cup I^+_{1,2,3}) \setminus I^-_{1,2,3}$ and the sets $I^+_{1,2}$ and $I^+_{1,2,3}$ are valid powerset interjoins: $I^+_{1,2} \subseteq Int^{(1)}_{\text{open}} \bowtie Int^{(2)}_{\text{open}}$ and $I^+_{1,2,3} \subseteq Int^{(1,2)}_{\text{open}} \bowtie Int^{(3)}_{\text{open}}$ where $Int^{(1,2)}_{\text{open}}$ contains interactions from $Int^{(1)}_{\text{open}}$, $Int^{(2)}_{\text{open}}$, and from a valid powerset interjoin of these two. Next, we consider Case 2:

$$\begin{aligned}
I^- &= (I^-_{1,2} \cup I^-_{1,2,3}) \setminus I^+_{1,2} \\
&\subseteq \left((Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}}) \cup (Int^{(1,2)}_{\text{open}} \cup Int^{(3)}_{\text{open}})\right) \setminus I^+_{1,2} \\
&= \left(Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}} \cup ((I^+_{1,2} \cup Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}}) \setminus I^-_{1,2}) \cup Int^{(3)}_{\text{open}}\right) \setminus I^+_{1,2}
\end{aligned}$$

$$= \left(Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}} \cup I^{+}_{1,2} \cup Int^{(3)}_{\text{open}}\right) \setminus I^{+}_{1,2}$$
$$= Int^{(1)}_{\text{open}} \cup Int^{(2)}_{\text{open}} \cup Int^{(3)}_{\text{open}}.$$

Thus, also this case holds. For $I^- \sqsubseteq I^+$, an interaction α contained in $I^- = (I^-_{1,2} \cup I^-_{1,2,3}) \setminus I^+_{1,2}$ that has no appropriate superset interaction β contained in $I^+ = (I^+_{1,2} \cup I^+_{1,2,3}) \setminus I^-_{1,2,3}$, i.e., $\alpha \subseteq \beta$, would violate either $I^-_{1,2} \sqsubseteq I^+_{1,2}$ or $I^-_{1,2,3} \sqsubseteq I^+_{1,2,3}$. Thus, we know that the contraction of the composition is correct, i.e., Equation ($\star$) holds.

Second, we expand the system of the right-hand side of Equation ($\star$) in a different order. The numbering of the interaction systems is not important, i.e., applying the renaming $1 = 3'$, $2 = 1'$, and $3 = 2'$ of the proposition, we get:

$$\bigotimes_{(I^+,I^-)} \{Sys^{(1)}, Sys^{(2)}, Sys^{(3)}\} = \bigotimes_{(I^+,I^-)} \{Sys^{(3')}, Sys^{(1')}, Sys^{(2')}\}.$$

Since $\{Sys^{(3')}, Sys^{(1')}, Sys^{(2')}\} = \{Sys^{(1')}, Sys^{(2')}, Sys^{(3')}\}$, we can apply Proposition 3.7 and Definition 3.4, which yields

$$\bigotimes_{(I^+,I^-)} \{Sys^{(1')}, Sys^{(2')}, Sys^{(3')}\} = \left(Sys^{(1')} \underset{\mathcal{I}_{1',2'}}{\otimes} Sys^{(2')}\right) \underset{\mathcal{I}_{1',2',3'}}{\otimes} Sys^{(3')}.$$

Now, we rename the interaction systems (but not the composition information) and apply Proposition 3.9:

$$\left(Sys^{(1')} \underset{\mathcal{I}_{1',2'}}{\otimes} Sys^{(2')}\right) \underset{\mathcal{I}_{1',2',3'}}{\otimes} Sys^{(3')} = Sys^{(1)} \underset{\mathcal{I}_{1',2',3'}}{\otimes} \left(Sys^{(2)} \underset{\mathcal{I}_{1',2'}}{\otimes} Sys^{(3)}\right). \qquad \blacksquare$$

Proof of Proposition 3.12: Let $(S_1, \Sigma_1, \{\xrightarrow{\alpha}_1\}_{\alpha\in\Sigma_1}, S^0_1) = [\![Sys^{(1)}]\!]$ with $\Sigma_1 = Int^{(1)}_{\text{open}} \cup \{\tau\}$, $(S_2, \Sigma_2, \{\xrightarrow{\alpha}_2\}_{\alpha\in\Sigma_2}, S^0_2) = [\![Sys^{(2)}]\!]$ with $\Sigma_2 = Int^{(2)}_{\text{open}} \cup \{\tau\}$, and $(S_3, \Sigma_3, \{\xrightarrow{\alpha}_3\}_{\alpha\in\Sigma_3}, S^0_3) = [\![Sys^{(3)}]\!]$ with $\Sigma_3 = Int^{(3)}_{\text{open}} \cup \{\tau\}$ be the corresponding global behaviors of the given interaction systems. Since $Sys^{(1)} \approx^{\Delta}_b Sys^{(2)}$ holds, the alphabets of the corresponding labeled transition systems are equal, i.e., $\Sigma_1 = \Sigma_2 =: \Sigma$ holds and the "either/or" distinction of the supersets of the sets I^+ and I^- is no restriction. Furthermore, there exists a symmetric relation $\mathcal{R} \subseteq S_1 \times S_2 \cup S_2 \times S_1$ with the properties of Definition 2.15. We now have to show that also a relation $\mathcal{R}^{\otimes} \subseteq S_{1,3} \times S_{2,3} \cup S_{2,3} \times S_{1,3}$ exists such that $Sys^{(1)} \otimes_{(I^+,I^-)} Sys^{(3)} \approx^{\Delta}_b Sys^{(2)} \otimes_{(I^+,I^-)} Sys^{(3)}$ holds where each $S_{i,3}$ with $i = 1, 2$ denotes the global state space of the corresponding composed system, which we define next. Let $\Gamma := (I^+ \cup \Sigma \cup \Sigma_3) \setminus I^-$ be the alphabet of the global behaviors of the composed systems, i.e., $[\![Sys^{(1)} \otimes_{(I^+,I^-)} Sys^{(3)}]\!] = (S_{1,3}, \Gamma, \{\xrightarrow{\alpha}_{1,3}\}_{\alpha\in\Gamma}, S^0_{1,3})$ and similarly, but with $Sys^{(2)}$ instead of $Sys^{(1)}$, $[\![Sys^{(2)} \otimes_{(I^+,I^-)} Sys^{(3)}]\!] = (S_{2,3}, \Gamma, \{\xrightarrow{\alpha}_{2,3}\}_{\alpha\in\Gamma}, S^0_{2,3})$ (cf. Definitions 2.6 and 3.4). Note that $\tau \in \Gamma$ holds because $\tau \in \Sigma \cup \Sigma_3$ and $\tau \notin I^-$

holds. We construct $\mathcal{R}^{\otimes}$ as follows.

$$\mathcal{R}^{\otimes} = \big\{ \big((s_i, s_3), (s_j, s_3)\big) \in S_{i,3} \times S_{j,3} \mid (s_i, s_j) \in \mathcal{R} \text{ for } i = 1,2 \text{ and } j = (i \bmod 2) + 1 \big\}.$$

We need to show that $\mathcal{R}^{\otimes}$ is indeed a relation that establishes branching bisimilarity with explicit divergence of the systems $Sys^{(1)} \otimes_{(I^+,I^-)} Sys^{(3)}$ and $Sys^{(2)} \otimes_{(I^+,I^-)} Sys^{(3)}$. First note that $\mathcal{R}^{\otimes}$ is symmetric since it is constructed out of a symmetric relation in a way that does not affect the symmetry. In the following, we reason about both $i = 1,2$ and $j = (i \bmod 2) + 1$ at the same time. Because of the premise, we know that for every initial state $s_i^0 \in S_i^0$ there is an initial state $s_j^0 \in S_j^0$ such that $(s_i^0, s_j^0) \in \mathcal{R}$. From the construction of $\mathcal{R}^{\otimes}$, it directly follows that for every initial state $s_3^0 \in S_3^0$, the pair $\big((s_i^0, s_3^0), (s_j^0, s_3^0)\big)$ is included in $\mathcal{R}^{\otimes}$, i.e., the first part of Definition 2.15 holds for $\mathcal{R}^{\otimes}$. We now consider the first part (2.1) of the second part and distinguish three cases. For all $\big((s_i, s_3), (s_j, s_3)\big) \in \mathcal{R}^{\otimes}$ with $s_i \in S_i$, $s_j \in S_j$, and $s_3 \in S_3$, and all $t_i \in S_i$, $t_3 \in S_3$, and $\alpha \in \Gamma$ holds: If $(s_i, s_3) \xrightarrow{\alpha}_{i,3} (t_i, t_3)$ then

1. $\alpha \in \Sigma$ and no component of $Sys^{(3)}$ participates in the transition, i.e., $s_3 = t_3$ holds. Since then also $s_i \xrightarrow{\alpha}_i t_i$ and since $(s_i, s_j) \in \mathcal{R}$ we know that either $\alpha = \tau$ and $(t_i, s_j) \in \mathcal{R}$, or that $r_j, t_j \in S_j$ exist with $s_j \xrightarrow{\tau}_j^* r_j$, $r_j \xrightarrow{\alpha}_j t_j$, $(s_i, r_j) \in \mathcal{R}$, and $(t_i, t_j) \in \mathcal{R}$. Thus, we either have $\alpha = \tau$ and $\big((t_i, s_3), (s_j, s_3)\big) \in \mathcal{R}^{\otimes}$, or $(s_j, s_3) \xrightarrow{\tau}_{j,3}^* (r_j, s_3)$, $(r_j, s_3) \xrightarrow{\alpha}_{j,3} (t_j, s_3)$, $\big((s_i, s_3), (r_j, s_3)\big) \in \mathcal{R}^{\otimes}$, and $\big((t_i, s_3), (t_j, s_3)\big) \in \mathcal{R}^{\otimes}$ by the definition of relation $\mathcal{R}^{\otimes}$.

2. $\alpha \in \Sigma_3$ and no component of $Sys^{(i)}$ participates in the transition, i.e., $s_i = t_i$ holds. Since also no component of $Sys^{(j)}$ participates, $s_3 \xrightarrow{\alpha}_3 t_3$, and $(s_i, s_j) \in \mathcal{R}$, also $(s_j, s_3) \xrightarrow{\alpha}_{j,3} (s_j, t_3)$ and $\big((s_i, t_3), (s_j, t_3)\big) \in \mathcal{R}^{\otimes}$ holds by the definition of relation $\mathcal{R}^{\otimes}$.

3. $\alpha \in I^+$: Components of both systems participate in the transition. Then there exist $\beta \in Int_{\text{open}}^{(i)}$ and $\gamma \in Int_{\text{open}}^{(3)}$ such that $\alpha = \beta \cup \gamma$ and it holds that $s_i \xrightarrow{\beta}_i t_i$ and $s_3 \xrightarrow{\gamma}_3 t_3$. Since $(s_i, s_j) \in \mathcal{R}$ and $\beta \neq \tau$, we know that $r_j, t_j \in S_j$ exist with $s_j \xrightarrow{\tau}_j^* r_j$, $r_j \xrightarrow{\beta}_j t_j$, $(s_i, r_j) \in \mathcal{R}$, and $(t_i, t_j) \in \mathcal{R}$. Thus, we have $(s_j, s_3) \xrightarrow{\tau}_{j,3}^* (r_j, s_3)$, $(r_j, s_3) \xrightarrow{\alpha}_{j,3} (t_j, t_3)$, $\big((s_i, s_3), (r_j, s_3)\big) \in \mathcal{R}^{\otimes}$, and $\big((t_i, t_3), (t_j, t_3)\big) \in \mathcal{R}^{\otimes}$ by the definition of relation $\mathcal{R}^{\otimes}$.

Note that if $\alpha = \tau$, which means that an interaction exists that is closed in one of the systems, both $\alpha \in \Sigma$ and $\alpha \in \Sigma_3$ holds. But, only one of the systems makes a step in this case, i.e., we can distinguish the Cases 1 and 2 as above.

For the infinite path case (2.2) of Definition 2.15, we can use the same argument

as in the Cases 1 and 2 above, since the infinite path is either in a system $Sys^{(i)}$ for $i = 1, 2$ and we find the corresponding state t_j, or in $Sys^{(3)}$ in which case it can be found in both composed systems. Thus, the relation $\mathcal{R}^{\otimes}$ establishes branching bisimilarity with explicit divergence of the systems in question, i.e., $Sys^{(1)} \otimes_{(I^+,I^-)} Sys^{(3)} \approx_b^{\Delta} Sys^{(2)} \otimes_{(I^+,I^-)} Sys^{(3)}$ holds. ∎

Proof of Proposition 3.14: We have to show that the interaction model constructed by the closing operator, $IM \setminus\setminus \hat{I}$, is indeed a valid interaction model. Since only the set of closed interactions is modified by the closing operator and *Sys* is a valid interaction system, we have to show that the set $Int_{\text{closed}} \cup (\hat{I} \cap Int)$ is contained in the set of interactions. By assumption, we know that $Int_{\text{closed}} \subseteq Int$ holds, and $(\hat{I} \cap Int) \subseteq Int$ follows from set theory. Thus, the closing operator yields a valid interaction system. ∎

Proof of Proposition 3.15: First, observe that all interactions that are closed before the composition on the right-hand side are also closed after the composition since we required that $\hat{I}^{(1)} \subseteq \hat{I}$ and $\hat{I}^{(2)} \subseteq \hat{I}$ holds. Thus, the equality of the systems depends on the validity of the composition, i.e., we have to show that no interaction is closed that is needed for the composition information. Let $X^{(i)} = \{\alpha \in 2^{Act^{(i)}} \mid \alpha \in I^- \vee (\exists\, \beta \in I^+ \colon \alpha = \beta \cap Act^{(i)} \wedge \alpha \neq \emptyset)\}$ denote the set defined in the theorem. We start with the set I^-. Observe that we demanded that for each $\alpha \in \hat{I}^{(i)}$ with $i = 1, 2$ holds $\alpha \notin X^{(i)}$. Thus no such α is contained in I^-. Next, consider the set I^+. Here, we required for each $\alpha \in \hat{I}^{(i)}$ with $i = 1, 2$ that no $\beta \in I^+$ exists with $\alpha = \beta \cap Act^{(i)}$—otherwise $\alpha \in X^{(i)}$ and $\hat{I}^{(i)} \cap X^{(i)} \neq \emptyset$. Thus, no closed interaction affects the composition information which proves the claim. ∎

Proof of Proposition 3.17: Let $(S_1, \Sigma_1, \{\xrightarrow{\alpha}_1\}_{\alpha \in \Sigma_1}, S_1^0) = [\![Sys^{(1)}]\!]$ with $\Sigma_1 = Int_{\text{open}}^{(1)} \cup \{\tau\}$ and $(S_2, \Sigma_2, \{\xrightarrow{\alpha}_2\}_{\alpha \in \Sigma_2}, S_2^0) = [\![Sys^{(2)}]\!]$ with $\Sigma_2 = Int_{\text{open}}^{(2)} \cup \{\tau\}$ be the corresponding global behaviors of the given interaction systems. Since $Sys^{(1)} \approx_b^{\Delta} Sys^{(2)}$ holds (we assume that the premise of the proposition holds for the proof), the alphabets of the labeled transition systems are equal, i.e., $\Sigma_1 = \Sigma_2 =: \Sigma$, and there exists a symmetric relation $\mathcal{R} \subseteq S_1 \times S_2 \cup S_2 \times S_1$ with the properties of Definition 2.15. We now have to show that also a relation $\hat{\mathcal{R}} \subseteq S_1 \times S_2 \cup S_2 \times S_1$ exists such that $Sys^{(1)} \setminus\setminus \hat{I} \approx_b^{\Delta} Sys^{(2)} \setminus\setminus \hat{I}$ holds. Let $\hat{\Sigma} := \{\tau\} \cup \Sigma \setminus \hat{I}$ be the alphabet of the global behaviors of the systems after the closing operation, i.e., $[\![Sys^{(1)} \setminus\setminus \hat{I}]\!] = (S_1, \hat{\Sigma}, \{\xrightarrow{\alpha}_{\hat{1}}\}_{\alpha \in \hat{\Sigma}}, S_1^0)$ and $[\![Sys^{(2)} \setminus\setminus \hat{I}]\!] = (S_2, \hat{\Sigma}, \{\xrightarrow{\alpha}_{\hat{2}}\}_{\alpha \in \hat{\Sigma}}, S_2^0)$—cf. Definition 3.13 and observe that only the alphabet and transition relation of a global behavior are modified by closing. We

explicitly included τ in the set $\hat{\Sigma}$ since $\tau \in \hat{I}$ is allowed but has no affect. We define $\hat{\mathcal{R}} := \mathcal{R}$ and show that $\hat{\mathcal{R}}$ is indeed a relation that establishes branching bisimilarity with explicit divergence of $Sys^{(1)} \setminus\!\setminus \hat{I}$ and $Sys^{(2)} \setminus\!\setminus \hat{I}$. First note that $\hat{\mathcal{R}}$ is symmetric since $\mathcal{R}$ is symmetric by definition. Since $\mathcal{R}$ equals $\hat{\mathcal{R}}$, the requirement for the initial states in Definition 2.15 is satisfied, i.e., for all states $s_i^0 \in S_i^0$ exists a state $s_j^0 \in S_j^0$ such that $(s_i^0, s_j^0) \in \hat{\mathcal{R}}$ for $i = 1, 2$ and $j = (i \bmod 2) + 1$. We now consider the second part of Definition 2.15. For $i = 1, 2$, $j = (i \bmod 2) + 1$, and all $(s_i, s_j) \in \hat{\mathcal{R}}$ with $s_i \in S_i$ and $s_j \in S_j$ holds

2.1. for all $t_i \in S_i$ and all $\alpha \in \hat{\Sigma}$, if $s_i \xrightarrow{\alpha}_{\hat{i}} t_i$ then

   (a) $\alpha = \tau$ and there is a $\beta \in \Sigma \cap \hat{I}$ with $s_i \xrightarrow{\beta}_i t_i$. Since $(s_i, s_j) \in \mathcal{R}$ and $\beta \neq \tau$ we know that states $r_j, t_j \in S_j$ exist with $s_j \xrightarrow{\tau}_j^* r_j$, $r_j \xrightarrow{\beta}_j t_j$, $(s_i, r_j) \in \mathcal{R}$, and $(t_i, t_j) \in \mathcal{R}$. Thus, also $s_j \xrightarrow{\tau}_{\hat{j}}^* r_j$ and $r_j \xrightarrow{\tau}_{\hat{j}} t_j$ with $(s_i, r_j) \in \hat{\mathcal{R}}$ and $(t_i, t_j) \in \hat{\mathcal{R}}$ because $\hat{\mathcal{R}} = \mathcal{R}$.

   (b) $s_i \xrightarrow{\alpha}_i t_i$ holds. Since $(s_i, s_j) \in \mathcal{R}$ we know that either $\alpha = \tau$ and $(t_i, s_j) \in \mathcal{R}$, or that $r_j, t_j \in S_j$ exist with $s_j \xrightarrow{\tau}_j^* r_j$, $r_j \xrightarrow{\alpha}_j t_j$, $(s_i, r_j) \in \mathcal{R}$, and $(t_i, t_j) \in \mathcal{R}$. Thus, we either have $\alpha = \tau$ and $(t_i, s_j) \in \hat{\mathcal{R}}$, or $s_j \xrightarrow{\tau}_{\hat{j}}^* r_j$ and $r_j \xrightarrow{\alpha}_{\hat{j}} t_j$ with $(s_i, r_j) \in \hat{\mathcal{R}}$ and $(t_i, t_j) \in \hat{\mathcal{R}}$ because $\hat{\mathcal{R}} = \mathcal{R}$.

2.2. if there is an infinite path π over $(S_i, \xrightarrow{\tau}_{\hat{i}})$ such that $\pi[0] = s_i$ then we find this path also over $(S_i, \xrightarrow{\tau}_i)$ or over $(S_i, \bigcup_{\alpha \in \Sigma \cap \hat{I}} \xrightarrow{\alpha}_i \cup \xrightarrow{\tau}_i)$. In the first case, we know that a state $t_j \in S_j$ exists such that $s_j \xrightarrow{\tau}_j t_j$ and $(t_i, t_j) \in \mathcal{R}$ where $t_i = \pi[k]$ for some $k \in \mathbb{N}$ because $(s_i, s_j) \in \mathcal{R}$. In the second case, the path cannot be found over $(S_i, \xrightarrow{\tau}_i)$, thus we find an index $l \in \mathbb{N}$ such that $\pi[l] \xrightarrow{\beta}_i \pi[l+1]$ with $\beta \in \Sigma \cap \hat{I}$ and $\beta \neq \tau$. Let l be the smallest among such indices, i.e., we have $\pi[0] \xrightarrow{\tau}_i^* \pi[l]$. Since $\mathcal{R}$ establishes branching bisimilarity with explicit divergence, we know that $(\pi[l], s_j) \in \mathcal{R}$ holds and that a state $t_j \in S_j$ exists with $(\pi[l+1], t_j) \in \mathcal{R}$. Thus, in either case we find such a state $t_j \in S_j$ with $s_j \xrightarrow{\tau}_{\hat{j}} t_j$ (in the second case because $\beta \in \Sigma \cap \hat{I}$) and $(r_i, t_j) \in \hat{\mathcal{R}}$ where $r_i = \pi[k']$ for some $k' \in \mathbb{N}$ because $\hat{\mathcal{R}} = \mathcal{R}$.

Thus, the relation $\hat{\mathcal{R}}$ establishes branching bisimilarity with explicit divergence of the systems in question, i.e., $Sys^{(1)} \setminus\!\setminus \hat{I} \approx_b^{\Delta} Sys^{(2)} \setminus\!\setminus \hat{I}$ holds. ∎

Proof of Proposition 3.19: Observe that $CS[C]$ is a valid component system with respect to Definition 2.1 because otherwise, two overlapping action sets are also overlapping in CS which contradicts the validity of this component system. The same argument applies for the behavioral model $\{[\![i]\!]\}_{i \in C}$. Thus, we have to show that the interaction model $IM[C]$ is valid. We have to show

three things: First, that every interaction contains at most one action of every component in C, second, that any action in $Act[C]$ is contained in at least one interaction, and third, that the set of closed interactions is a subset of the set of all interactions (cf. Definitions 2.2 and 2.3). Clearly, no interaction contains an action of a component that is not contained in C since all interactions are intersected with $Act[C]$. Furthermore, for all interactions $\alpha \in Int[C]$ at least one "superset" interaction β exists in Int because of "$\exists\, \beta \in Int\colon \alpha = \beta \cap Act[C]$" and the removal of the empty set in the definition of $Int[C]$. Now, the first claim holds, since otherwise, a component that participates with two actions in one interaction can also be found in Int which contradicts the validity of IM. For the second claim, the contrary would again violate the validity of the original interaction model. For the set of closed interactions, observe that it merely consists of interactions that were also closed in the original system and whose participating components are contained in C. This proves the third claim. Thus, the subsystem construction operator yields a valid interaction system. ∎

Proof of Proposition 3.20: By setting $C := Comp$ in Definition 3.18, we get $CS = CS[C]$ and $Act = Act[C]$. This implies $Int = Int[C]$, and thus $IM = IM[C]$ because $Int \subseteq 2^{Act}$ and all components participating in a closed interaction are contained in $Comp$. Since also $\{[\![i]\!]\}_{i \in Comp} = \{[\![i]\!]\}_{i \in C}$ holds, the claim follows. ∎

Proof of Proposition 3.21: First observe that all interactions that are closed on the left-hand side of the equation are also closed on the right-hand side because we explicitly declare them as closed. Second, we show that the composition is valid (cf. Definition 3.4), i.e., (1) $I^+_{C_1,C_2} \subseteq Int_{\text{open}}[C_1] \bowtie Int_{\text{open}}[C_2]$, (2) $I^-_{C_1,C_2} \subseteq Int_{\text{open}}[C_1] \cup Int_{\text{open}}[C_2]$, and (3) $I^-_{C_1,C_2} \sqsubseteq I^+_{C_1,C_2}$ holds. For Claim (1), observe that each $\alpha \in I^+_{C_1,C_2}$ can be partitioned into nonempty sets $\alpha_1 \subseteq Act[C_1]$ and $\alpha_2 \subseteq Act[C_2]$—otherwise we find no $i \in C_1$ and $j \in C_2$ that both participate in α. Since for all such α also $\alpha \in Int[C_1 \cup C_2]$ holds, we can conclude that $\alpha_1 \in Int_{\text{open}}[C_1]$ and $\alpha_2 \in Int_{\text{open}}[C_2]$ holds because for $k = 1, 2$ the interaction α_k cannot be closed in the subsystem $Sys[C_k]$ since $\alpha_k \subseteq \alpha$ but $\text{compset}(\alpha) \nsubseteq C_k$ (cf. Definition 3.18). Thus, also $\alpha \in Int_{\text{open}}[C_1] \bowtie Int_{\text{open}}[C_2]$ holds (cf. Definition 3.2). Claim (2) follows from set theory—since $\{\alpha \in Int[C_1] \cup Int[C_2] \mid \alpha \notin Int[C_1 \cup C_2]\} \subseteq Int[C_1] \cup Int[C_2]$—and the fact that an interaction that does not occur in $Int[C_1 \cup C_2]$ but in $Int[C_1]$ or $Int[C_2]$ cannot be contained in Int_{closed}—otherwise the set of closed interactions is not a subset of the set of interactions in Sys. For Claim (3), choose an $\alpha \in I^-_{C_1,C_2}$. Since $\alpha \in Int[C_1] \cup Int[C_2]$, we find a $\beta \in Int$ with $\alpha = \beta \cap Act[C_k]$ for either

$k = 1$ or $k = 2$ because $\alpha \notin Int[C_1 \cup C_2]$. Consider $\gamma = \beta \cap Act[C_1 \cup C_2]$. Now, $\alpha \subset \gamma$ holds—otherwise $\alpha \in Int[C_1 \cup C_2]$—and thus $\gamma \in I^+_{C_1,C_2}$. We conclude $I^-_{C_1,C_2} \sqsubseteq I^+_{C_1,C_2}$ (cf. Definition 3.3).

Third, we prove the claim of the proposition. It is clear that $CS[C_1 \cup C_2] = CS[C_1] \otimes CS[C_2]$ holds and that the behavioral models are equal (cf. Definition 3.4). We have to show that also the interaction models of the subsystems are equal, where it suffices to show that the set of interactions are equal since the equality of the set of closed interactions is then implied by the application of the closing operator. Thus, we have to show that $Int[C_1 \cup C_2] \stackrel{?}{=} \left(I^+_{C_1,C_2} \cup Int[C_1] \cup Int[C_2]\right) \setminus I^-_{C_1,C_2}$ holds (cf. Definition 3.4). We partition the set $Int[C_1 \cup C_2]$ into three sets: Set $X_{1,2}$ that contains interactions where components contained in both C_1 and C_2 participate in, set X_1 where only components from C_1 participate in, and set X_2 for the components in C_2, i.e., $Int[C_1 \cup C_2] = X_{1,2} \cup X_1 \cup X_2$. By definition, we have $X_{1,2} = I^+_{C_1,C_2}$. For the other two sets, we only know that $X_1 \subseteq Int[C_1]$ and $X_2 \subseteq Int[C_2]$ holds since the interactions contained in a subsystem need not to be present in $Sys[C_1 \cup C_2]$. But, for X_k with $k = 1, 2$ we can repair this situation by removing those interactions, i.e., we have $X_k = Int[C_k] \setminus \{\alpha \in Int[C_k] \mid \alpha \notin Int[C_1 \cup C_2]\}$. We substitute the auxiliary sets of the partitioning and get:

$$
\begin{aligned}
Int[C_1 \cup C_2] &= I^+_{C_1,C_2} \\
&\quad \cup \left(Int[C_1] \setminus \{\alpha \in Int[C_1] \mid \alpha \notin Int[C_1 \cup C_2]\}\right) \\
&\quad \cup \left(Int[C_2] \setminus \{\alpha \in Int[C_2] \mid \alpha \notin Int[C_1 \cup C_2]\}\right) \\
&= \left(I^+_{C_1,C_2} \cup Int[C_1] \cup Int[C_2]\right) \\
&\quad \setminus \{\alpha \in Int[C_1] \mid \alpha \notin Int[C_1 \cup C_2]\} \\
&\quad \setminus \{\alpha \in Int[C_2] \mid \alpha \notin Int[C_1 \cup C_2]\} \\
&= \left(I^+_{C_1,C_2} \cup Int[C_1] \cup Int[C_2]\right) \\
&\quad \setminus \{\alpha \in Int[C_1] \cup Int[C_2] \mid \alpha \notin Int[C_1 \cup C_2]\} \\
&= \left(I^+_{C_1,C_2} \cup Int[C_1] \cup Int[C_2]\right) \setminus I^-_{C_1,C_2}.
\end{aligned}
$$

This proves the claim. ∎

Proof of Proposition 3.23: We prove the claim by showing that the composition information of Proposition 3.21 equals the composition information of the new proposition, i.e., $\mathcal{I}_{C_1,C_2} = \mathcal{I}_{C_1 \setminus C_1', C_2 \setminus C_2'}$ which means $I^+_{C_1,C_2} = I^+_{C_1 \setminus C_1', C_2 \setminus C_2'}$ and $I^-_{C_1,C_2} = I^-_{C_1 \setminus C_1', C_2 \setminus C_2'}$. Observe that this proves the claim.

Pick an $\alpha \in I^+_{C_1,C_2}$. By definition, we know that component $i \in C_1$ and component $j \in C_2$ exist that both participate in α. Now, we have $i \notin C_1'$ since otherwise we can find an interaction $\beta \in Int$ with $\alpha \subseteq \beta$ that violates the

requirement about the set C_1'—we would have $i \in C_1' \wedge j \in C_2$ but not $\{i,j\} \not\subseteq \text{compset}(\beta)$. Analogously, we have $j \notin C_2'$ and also $\text{compset}(\alpha) \cap (C_1' \cup C_2') = \emptyset$. Thus, we conclude $\alpha \in I^+_{C_1 \setminus C_1', C_2 \setminus C_2'}$. Since the case $I^+_{C_1 \setminus C_1', C_2 \setminus C_2'} \subseteq I^+_{C_1,C_2}$ can be treated analogously, we conclude $I^+_{C_1,C_2} = I^+_{C_1 \setminus C_1', C_2 \setminus C_2'}$.

Next, we treat $I^-_{C_1,C_2}$. Fix an $\alpha \in I^-_{C_1,C_2}$. Assume $\alpha \in \mathit{Int}[C_1]$. Since $\alpha \notin \mathit{Int}[C_1 \cup C_2]$, we know that a $\beta \in \mathit{Int}[C_1 \cup C_2]$ exists with $\alpha \subset \beta$ and $\beta \notin \mathit{Int}[C_1]$ (similar to the proof of Proposition 3.21). As above, we can conclude that $\text{compset}(\alpha) \cap C_1' = \emptyset$—otherwise $i \in C_1'$ and $j \in C_2$ exist with $\{i,j\} \subseteq \text{compset}(\beta)$ which violates the requirement on C_1'—and thus $\alpha \in \mathit{Int}[C_1 \setminus C_1']$. Because $\alpha \notin \mathit{Int}[C_1 \cup C_2]$, also $\alpha \notin \mathit{Int}[C_1 \setminus C_1' \cup C_2 \setminus C_2']$ holds and thus $\alpha \in I^-_{C_1 \setminus C_1', C_2 \setminus C_2'}$. The case $\alpha \in \mathit{Int}[C_2]$ is treated similarly. Again, since the case $I^-_{C_1 \setminus C_1', C_2 \setminus C_2'} \subseteq I^-_{C_1,C_2}$ can be treated analogously, we conclude $I^-_{C_1,C_2} = I^-_{C_1 \setminus C_1', C_2 \setminus C_2'}$. This proves the claim. ∎

Proof of Theorem 3.26: First observe that (I^+, I^-) is a valid composition since $\emptyset \sqsubseteq I^+$ (cf. Definition 3.3). We prove the claim by contradiction. Assume that both $\mathit{Sys}^{(1)}$ and $\mathit{Sys}^{(2)}$ are deadlock-free but their composition $\mathit{Sys} = \mathit{Sys}^{(1)} \otimes_{(I^+,I^-)} \mathit{Sys}^{(2)}$ is not. Thus, we find a reachable global state s in $[\![\mathit{Sys}]\!]$ with $\text{Suc}(s) = \emptyset$, i.e., there is a global initial state s^0 and a finite path π starting in s^0 and ending in s. For this global state, we consider the two states $s^{(1)}$ and $s^{(2)}$ that are derived from s by ignoring all local states of components not contained in the set $\mathit{Comp}^{(1)}$ and $\mathit{Comp}^{(2)}$ respectively.

Now, we only consider $s^{(1)}$ but the same holds for $s^{(2)}$ in an analogous way. Since s is a deadlock, also $s^{(1)}$ must be a deadlock because all interactions where components of $\mathit{Sys}^{(1)}$ participate in are blocked and we have $I^- = \emptyset$. We show that $s^{(1)}$ is reachable in $[\![\mathit{Sys}^{(1)}]\!]$ by considering the path π from above: If we restrict every state on the path to states of the components in $\mathit{Comp}^{(1)}$ and also every label to the actions of $\mathit{Act}^{(1)}$, we can conclude that consecutive (partial) states on the new path are either identical or can be related by a transition in $[\![\mathit{Sys}^{(1)}]\!]$ that is labeled with an interaction of $\mathit{Int}^{(1)}$. Note that otherwise, we find an interaction $\gamma \in \mathit{Int}$ that occurs as a label on the path π with $\gamma \cap \mathit{Act}^{(1)} \neq \emptyset$ and $(\gamma \cap \mathit{Act}^{(1)}) \notin \mathit{Int}^{(1)}$, which violates our assumption about the sets I^+ and I^- that every new interaction consists of the union of two existing interactions and that no old interaction is allowed, i.e., removed in the composite system.

However, this implies that the deadlocked state $s^{(1)}$ is reachable in $[\![\mathit{Sys}^{(1)}]\!]$ which contradicts the assumed deadlock-freedom of this system. Thus, the interaction system Sys is deadlock-free. ∎

F.3 Proofs from Chapter 4

Proof of Lemma 4.8: Assume that a simple cycle exists in G_{coop} which contains the vertices s and t. Thus, there are two simple paths (cf. Definition A.3)

$$\pi_1 = \langle s, v^1, v^2, \ldots, v^k, t \rangle \text{ and } \pi_2 = \langle s, w^1, w^2, \ldots, w^l, t \rangle$$

with $k, l \in \mathbb{N}$ that have only the vertices s and t in common, i.e., $v^{k'} \neq w^{l'}$ for all $k', l' \in \mathbb{N}$ with $1 \leq k' \leq k$ and $1 \leq l' \leq l$.

We now construct a flow function $f\colon E' \to \mathbb{N}$ in $N_{s,t}$ and show that its flow value $|f| = \sum_{(s_{\text{in}}, v) \in E'} f((s_{\text{in}}, v)) - \sum_{(w, s_{\text{in}}) \in E'} f((w, s_{\text{in}}))$ (for all suitable $v, w \in V'$) equals 2 and is maximal among all such functions, i.e., f is a maximum flow. Let

$$\begin{aligned}
f((s_{\text{in}}, s_{\text{out}})) = 2, \quad & f((s_{\text{out}}, v^1_{\text{in}})) = 1, \quad & f((s_{\text{out}}, w^1_{\text{in}})) = 1, \\
& f((v^1_{\text{in}}, v^1_{\text{out}})) = 1, \quad & f((w^1_{\text{in}}, w^1_{\text{out}})) = 1, \\
& f((v^1_{\text{out}}, v^2_{\text{in}})) = 1, \quad & f((w^1_{\text{out}}, w^2_{\text{in}})) = 1, \\
& \ldots & \ldots \\
& f((v^k_{\text{out}}, t_{\text{in}})) = 1, \quad & f((w^l_{\text{out}}, t_{\text{in}})) = 1,
\end{aligned}$$

and let all other edges $e \in E'$, i.e., those edges that have no corresponding part in the paths π_1 or π_2, transport no flow unit, i.e., $f(e) = 0$. Obviously, f is a maximum flow with $|f| = 2$ because the only edge leaving the source is $(s_{\text{in}}, s_{\text{out}})$, which is fully saturated, and all incoming edges of the source s_{in} have flow value 0. The construction of the flow function f by following the corresponding edges of the paths π_1 and π_2 ensures that a legal flow is created, i.e., all capacity constraints are satisfied and the flow is conserved at every vertex. This proves the implication of the lemma.

Next, assume that a maximum flow f with $|f| = 2$ exists in $N_{s,t}$. Starting with the source s_{in}, we now follow the paths in the directed graph G' that are induced by the flow function. By construction, vertex s_{in} is incident to exactly one outgoing edge, viz. $(s_{\text{in}}, s_{\text{out}})$, which has the flow value 2, i.e., is fully saturated. Further following the route of the flow from vertex s_{out} yields exactly two distinct vertices, denoted by v^1_{in} and w^1_{in} in the following, because any vertex adjacent to s_{out} has exactly one outgoing edge by construction—an edge to the "out" version of the vertex—and these edges have a capacity of only one which yields the distinctness of the vertices v^1_{in} and w^1_{in}. Thus, the flow is split at vertex s_{out}, routed to vertices v^1_{in} and w^1_{in}, and from there to vertices v^1_{out} and w^1_{out} respectively because of the single edge between the "in" and "out" versions of a vertex.

Repeating the argument as above lets us construct two simple paths

$$\pi_1' = \langle s_{\text{in}}, s_{\text{out}}, v_{\text{in}}^1, v_{\text{out}}^1, v_{\text{in}}^2, \ldots, v_{\text{out}}^k, t_{\text{in}} \rangle \text{ and } \pi_2' = \langle s_{\text{in}}, s_{\text{out}}, w_{\text{in}}^1, \ldots, w_{\text{out}}^l, t_{\text{in}} \rangle$$

with $k, l \in \mathbb{N}$ that have only the vertices s_{in}, s_{out}, and t_{in} in common. Here, the distinctness of the inner vertices of the paths is ensured by the flow conservation property of the maximum flow; since otherwise a twice occurring "in" vertex would have a larger incoming flow than it could transport over its single outgoing edge that has the capacity of one. Observe that we split every vertex of the cooperation graph G in the construction of G' to achieve exactly this property of the paths.

We now consider the corresponding paths in the undirected version of the network, i.e., we can find the corresponding vertices in G and construct the following simple paths:

$$\pi_1 = \langle s, v^1, v^2, \ldots, v^k, t \rangle \text{ and } \pi_2 = \langle s, w^1, w^2, \ldots, w^l, t \rangle$$

with the same indices. Because these paths have only the vertices s and t in common, we can concatenate them to a simple cycle as required by the statement of the lemma. ∎

Proof of Theorem 4.9: Assume that *Sys* has a disjoint circular wait free architecture. Then, no simple cycle exists in G_{coop} where two (or more) vertices that represent components lie on (cf. Definition 4.6). Thus, by Lemma 4.8 we can conclude that no maximum flow of value 2 exists in any flow network in question.

Next, assume that for all unordered pairs of components $i, j \in \mathit{Comp}$, whose vertex representations in G_{coop} have at least two neighbors, it holds that no maximum flow of value 2 exists in the associated flow network $N_{\{i\},\{j\}}$. Assume that *Sys* does not have a disjoint circular wait free architecture. Thus, we can find a simple cycle in G_{coop} where at least two vertices that represent components lie on. By the premise and Lemma 4.8, we know that at least one of these vertices has only one neighbor in G_{coop}, since otherwise, the premise is false. But, this contradicts the assumption that the cycle is simple because the neighbor of this vertex occurs twice. ∎

Proof of Lemma 4.10: We show the claim by proving that every vertex that represents a component in the cooperation graph $G'_{\text{coop}} = (V', E')$ of *Sys*′ has only one neighbor except the vertex that represents the new component x. If this statement holds, then the disjoint circular wait freedom of the architecture of *Sys*′ is implied by Theorem 4.9. Note that we do not show here that the

transformation yields as valid interaction system, since this can be shown directly by the validity of the original system.

Assume that a vertex $u \in V'$ exists in G'_{coop} with $|u| = 1$ and $u \neq \{x\}$ that has two distinct neighbors v and w, i.e., u has a degree greater than one. Since $\{u, v\} \in E'$ and $\{u, w\} \in E'$, we know that $|v| > 1$ and $|w| > 1$ holds, i.e., the vertices do not represent components, because otherwise, the edges are not included in E' (cf. Definition 4.4). Let $i \in \mathit{Comp}'$ be the component that is represented by u, i.e., $u = \{i\}$. Since component x participates in every interaction, we know that $x \in v$ and $x \in w$ holds. But, since the vertices are adjacent to u, also $i \in v$ and $i \in w$ holds. The transformation also introduced the interaction $\{\mathit{fresh}_x, \mathit{fresh}_i\} \in \mathit{Int}'$, and thus a vertex $\{x, i\} \in V'$. But, this means $u \subset \{x, i\} \subseteq v$ and $u \subset \{x, i\} \subseteq w$ holds, i.e., u has a degree of one and the vertices v, w cannot be distinct. Thus, Sys' has a disjoint circular wait free architecture. ∎

Proof of Lemma 4.11: In order to be able to establish the isomorphism of the global behaviors of the two interaction systems (cf. Definition 2.18 with $\mathit{LTS}_1 = [\![\mathit{Sys}]\!]$ and $\mathit{LTS}_2 = [\![\mathit{Sys}']\!]$), we have to restrict the codomain of the function mapping the interactions since the fresh actions, that are introduced for the existing components, are not included in Sys and thus, we cannot find a corresponding inverse image for interactions of the type $\{\mathit{fresh}_x, \mathit{fresh}_i\}$ for $i \in \mathit{Comp}$. This restriction does not affect the isomorphism of the systems, since any interaction that merely consists of fresh actions is not enabled in $[\![\mathit{Sys}']\!]$ because no transition labeled with a fresh action of a component (except for component x) exists in the corresponding local behavior.

Thus, in order to show that the two global behaviors are isomorphic up to transition relabeling, we define a bijective function that maps the states, viz. $f\colon S \to S'$, and one that maps the labels, viz. $g\colon \mathit{Int}^{\tau}_{\text{open}} \to (\mathit{Int}'_{\text{open}} \setminus \{\{\mathit{fresh}_x, \mathit{fresh}_i\} \mid i \in \mathit{Comp}\}) \cup \{\tau\}$ (cf. Definition 2.6 for the labeled transition system that represents the global behavior), and show that for every initial state and every transition in one system, we find an equivalent one in the other system. For a global state $s \in S$ and an interaction $\alpha \in \mathit{Int}^{\tau}_{\text{open}}$, we set (where $n = |\mathit{Comp}|$ and $g(\tau) = \tau$):

$$f(s) = f((s_1, \ldots, s_n)) = (s_1, \ldots, s_n, s_x^0) \text{ and } g(\alpha) = \alpha \cup \{\mathit{fresh}_x\}.$$

In other words, the function f extends the tuple representing a global state of $[\![\mathit{Sys}]\!]$ with the (unique) local state s_x^0 of the new component x, and the function g adds the (unique) action fresh_x of x to every interaction. The functions are bijective since for f, every state is uniquely mapped to its extended version and the complete mapping of S' follows from the fact that S_x (the state space

of component x) contains only one state, and for g, every open interaction is uniquely mapped to a version combined with the single action fresh_x of x and all appropriate interactions are covered since x participates in all interactions of Sys'. Furthermore, every closed interaction in Sys has a unique counterpart in Sys', thus we could set $g(\tau) = \tau$.

Now, for every initial state $s^0 \in S^0$ holds $f(s^0) \in S^{0\prime}$ because s^0_x is a local initial state of component x, and analogously, the same argument holds for any state $s^{0\prime} \in S^{0\prime}$, i.e., we find a state $f^{-1}(s^{0\prime}) \in S^0$. Otherwise, the states s^0 and $s^{0\prime}$ respectively are not initial states or f is not bijective.

For a transition $s \xrightarrow{\alpha} t$ with $s, t \in S$ and $\alpha \in \mathit{Int}_{\text{open}}$, we have to show that $f(s) \xrightarrow{\beta}{}' f(t)$ holds with $\beta = g(\alpha)$. Assume that such a transition exists in $[\![\mathit{Sys}]\!]$ but the mapped version does not exist in $[\![\mathit{Sys}']\!]$. We take a look at the global state s and extend it with the local state s^0_x of component x. Since fresh_x is enabled in s^0_x and α is enabled in s, we can execute the interaction $(\alpha \cup \{\mathit{fresh}_x\})$ in s which leads to state t extended with s^0_x since the transition $s^0_x \xrightarrow{a}_x s^0_x$ with $a = \mathit{fresh}_x$ is the only transition of $[\![x]\!]$. But, this means we have a transition $f(s) \xrightarrow{\beta}{}' f(t)$ with $\beta = g(\alpha)$—otherwise the functions f or g are not bijective. Analogously, we can argue for the "only if" case of Definition 2.18 and also for the case of a global τ-transition where we have to use the existence of a $\gamma \in \mathit{Int}_{\text{closed}}$ that corresponds to the transition label. Note that as already mentioned, no transition labeled with an interaction that only consists of fresh actions exists in $[\![\mathit{Sys}']\!]$ because no corresponding local transition exists in the behavior of the components (except for x). Thus, the two systems are isomorphic up to transition relabeling. ∎

Proof of Theorem 4.12: Let Sys be an arbitrary interaction system. If we apply our transformation on Sys that yields Sys', we get a system with a disjoint circular wait free architecture (cf. Lemma 4.10) that exhibits isomorphic behavior up to transition relabeling (cf. Lemma 4.11). We need to prove that the transformation can be performed in linear time in the size of the input Sys. Since we add a fresh action to every component, add a new component with one single action, add that action to every interaction, and add one new interaction of size two for every component, we need to process each component and interaction once and add new interactions of constant length whose number is bounded by the number of components. Thus, these operations can be performed in linear time in the size of the input. ∎

Proof of Corollary 4.13: Let Sys be an arbitrary interaction system. If we apply our transformation on Sys that yields Sys', we get a system with a disjoint

circular wait free architecture that exhibits isomorphic behavior (cf. Theorem 4.12). Thus, any decision problem for *Sys* can also be answered using Sys', if the problem is closed under isomorphism up to transition relabeling (which we assumed), by one call to a decision procedure for interaction systems with a disjoint circular wait free architecture, i.e., we have a many-to-one reduction. Since Theorem 4.12 also showed that the transformation can be performed in linear time, the reduction is a linear-time many-to-one reduction. ∎

F.4 Proofs from Chapter 5

Proof of Proposition 5.4: If *Sys* has strongly exclusive communication, then we have $\alpha \cap \beta = \emptyset$ for all $\alpha, \beta \in Int$ with $\alpha \neq \beta$. Thus, we find no two distinct interactions, i.e., interactions that potentially have different sets of participating components, that satisfy the premise ($\alpha \cap \beta \neq \emptyset$) of Definition 5.1, i.e., *Sys* also has exclusive communication. ∎

Proof of Lemma 5.5: Let $Sys' = \textsc{Exclusive}(Sys)$ denote the result of Algorithm B.8. We first show that the algorithm indeed yields a valid interaction system. Observe that the algorithm neither modifies the set of components nor a component's local state space. Thus, each component's behavior is valid with respect to Definition 2.5 as long as no transition label occurs that is not contained in the corresponding alphabet. This is ensured in lines 14–15. Since each action is only superscripted by a set in line 10, all sets of actions are disjoint (because they are disjoint in the original system). Thus, the component and behavioral models are valid. Now, all interactions of Int' are a subset of the set of actions Act' because any action that is added while constructing a new interaction in line 12 is also added to the corresponding action set in line 11. For the set of interactions Int', the coverage of Act', i.e., $\bigcup_{\alpha \in Int'} \alpha = Act'$, is guaranteed because the for loop in line 7 treats each action at least once and because the original system *Sys* is a valid interaction system. The set of closed interactions is a subset of the set of interactions, i.e., $Int'_{\text{closed}} \subseteq Int'$, since an interaction is only added to $Int'_{\text{closed}} = Int^{\text{new}}_{\text{closed}}$ in line 18 if it was previously added to $Int' = Int^{\text{new}}$ in line 16. Thus, we also have a valid interaction model, and finally, we showed that Sys' is a valid interaction system.

Next, we have to show that Sys' has exclusive communication. If the intersection of two interactions $\alpha, \beta \in Int'$ is nonempty, i.e., $\alpha \cap \beta \neq \emptyset$, we know that the actions in the intersection are equal and thus have the same superscript. Since every action in α is superscripted with $\text{compset}(\alpha)$ and every action in β with $\text{compset}(\beta)$, we conclude that $\text{compset}(\alpha) = \text{compset}(\beta)$ holds (cf.

Definition 5.1). Thus, Sys' has exclusive communication. ∎

Proof of Lemma 5.6: Let $Sys' = \text{EXCLUSIVE}(Sys)$ denote the resulting system of Algorithm B.8. The case that Sys' is a valid interaction system has been shown in the proof of Lemma 5.5. In order to establish the isomorphism of the two systems (cf. Definition 2.18 with $LTS_1 = [\![Sys]\!]$ and $LTS_2 = [\![Sys']\!]$), we define a bijective function that maps the states, viz. $f\colon S \to S'$, and one that maps the labels, viz. $g\colon Int^{\tau}_{\text{open}} \to Int'_{\text{open}} \cup \{\tau\}$ (cf. Definition 2.6 for the labeled transition system that represents the global behavior), and show that for every initial state and every transition in one system, we find an equivalent one in the other system. Note that $Int'_{\text{open}} = Int^{\text{new}} \setminus Int^{\text{new}}_{\text{closed}}$ as computed by Algorithm B.8. For function f we can use the identity function since the local state spaces of the components are not modified by Algorithm B.8, and thus the Cartesian product of the local state spaces is the same for Sys and $\text{EXCLUSIVE}(Sys)$, i.e., we have $S = S'$. Since we also have $S^0 = S^{0\prime}$, we conclude that for all $s \in S$ it holds that $s \in S^0$ holds if and only if $f(s) \in S^{0\prime}$ holds.

The function g is defined as follows: For $\alpha \in Int^{\tau}_{\text{open}}$ we set $g(\alpha) = \tau$ if $\alpha = \tau$ and $g(\alpha) = \{a^{\text{compset}(\alpha)} \mid a \in \alpha\}$ otherwise. Function g is injective since every interaction is uniquely mapped to a version where the actions contained in the interaction are superscripted by the set of participating components. The surjectivity of g follows since all interactions in Int'_{open} are covered by construction: An interaction can only be contained in Int'_{open} if an interaction of the original system was modified by the for loop in line 7 of Algorithm B.8, and an interaction is only closed in Sys' if it is closed in Sys (cf. line 17).

Now, for all $s, t \in S$ and all $\alpha \in Int_{\text{open}}$ we have:

$$
\begin{aligned}
&s \xrightarrow{\alpha} t \Leftrightarrow \forall\, i \in Comp\colon \big(i(\alpha) \neq \emptyset \implies s_i \xrightarrow{i(\alpha)}_i t_i\big) \wedge \big(i(\alpha) = \emptyset \implies s_i = t_i\big)\\
&\Leftrightarrow \big(\forall\, i \in \text{compset}(\alpha)\colon s_i \xrightarrow{i(\alpha)}_i t_i\big) \wedge \big(\forall\, i \notin \text{compset}(\alpha)\colon s_i = t_i\big)\\
&\Leftrightarrow \big(\forall\, i \in \text{compset}(\alpha)\ \exists\, a \in \alpha\colon s_i \xrightarrow{a}_i t_i\big) \wedge \big(\forall\, i \notin \text{compset}(\alpha)\colon s_i = t_i\big)\\
&\Leftrightarrow \big(\forall\, i \in \text{compset}(g(\alpha))\ \exists\, a \in g(\alpha)\colon s_i \xrightarrow{a}'_i t_i\big) \wedge\\
&\qquad\qquad \big(\forall\, i \notin \text{compset}(g(\alpha))\colon s_i = t_i\big)\\
&\Leftrightarrow s \xrightarrow{g(\alpha)}' t \Leftrightarrow f(s) \xrightarrow{\beta}' f(t) \text{ with } \beta = g(\alpha).
\end{aligned}
$$

Analogously, for $s \xrightarrow{\tau} t$—above we only quantified over all $\alpha \in Int_{\text{open}}$, i.e., the case $\alpha = \tau$ needs to be treated separately—we can use the existence of a $\gamma \in Int_{\text{closed}}$ that corresponds to the transition label to conclude that $s \xrightarrow{\tau} t \Leftrightarrow f(s) \xrightarrow{\tau}' f(t)$ holds. Now, we can conclude that for all $s, t \in S$ and all $\alpha \in Int^{\tau}_{\text{open}}$ it holds that $s \xrightarrow{\alpha} t$ if and only if $f(s) \xrightarrow{\beta}' f(t)$ with $\beta = g(\alpha)$. Thus, the two systems are isomorphic up to transition relabeling. ∎

Proof of Corollary 5.9: If $|Comp| < 3$ the premise of the corollary is false, i.e., the corollary holds. Thus, assume that $|Comp| \geq 3$ and that the premise of the corollary holds. Number the components via a bijective function $f : Comp \to \{1, \ldots, n\}$ with $n = |Comp|$ in an arbitrary way with number 1 being the middle component m and number n being the chosen border component k, i.e., $f(m) = 1$ and $f(k) = n$ holds. Then apply Theorem 5.7. ∎

Proof of Theorem 5.10: Assume that the premise of the theorem holds. We show the claim by induction over the "eccentricity levels" of the component graph, where such a level denotes all vertices with the same eccentricity. Note that the innermost level corresponds to the center of the graph, where the eccentricity of the vertices equals the radius, and that the outermost level corresponds to the periphery, where the eccentricities equal the diameter (cf. Definition A.7).

From the premise, we know that $|Comp| \geq 3$ holds and that the architecture is tree-like, thus we have $\mathrm{diam}(G) > \mathrm{rad}(G)$, i.e., there are at least two eccentricity levels. For the base case, we assume that $\mathrm{diam}(G) = \mathrm{rad}(G) + 1$ holds. We have to distinguish two cases:

1. $\mathrm{diam}(G) = 2$ and $\mathrm{rad}(G) = 1$ and
2. $\mathrm{diam}(G) = 3$ and $\mathrm{rad}(G) = 2$.

Note that other cases are not possible (if $\mathrm{diam}(G) = \mathrm{rad}(G) + 1$), since we have at least three vertices and G is a tree in the graph-theoretical sense.

Case 1: In this case, the interaction system has a star-like architecture since the maximal distance of a pair of vertices is two. We have $\mathcal{C}_1^\star = \{Comp\}$, and $m_{1.1}$ denotes the middle component of Sys and $b_{1.1}$ an arbitrary border component. We are given a bijective function $f_{1.1}$ that numbers the components such that number $1_{1.1}$ denotes $m_{1.1}$ and number $n_{1.1}$ denotes $b_{1.1}$. From the premise, we know that for all $2_{1.1} \leq i_{1.1} < n_{1.1}$ holds

$$Sys[\{1_{1.1}, i_{1.1}, i_{1.1} + 1\}] \setminus\setminus \hat{I}_{1_{1.1}, i_{1.1}} \approx_b^\Delta Sys[\{1_{1.1}, i_{1.1} + 1\}] \setminus\setminus \hat{I}_{1_{1.1}, i_{1.1}}.$$

Thus, if we set in Theorem 5.7 the function f to $f_{1.1}$, we can conclude that

$$Sys \setminus\setminus \hat{I} \approx_b^\Delta Sys[\{1_{1.1}, n_{1.1}\}] \setminus\setminus \hat{I}$$

holds since each set $\hat{I}_{1_{1.1}, i_{1.1}}$ equals the respective set $\hat{I}_{1,i}$ of Theorem 5.7 if $f = f_{1.1}$. Since $\mathrm{center}(G) = \{1_{1.1}\}$, we can set $K = \{n_{1.1}\}$, and with $M = \{1_{1.1}, n_{1.1}\}$ the claim follows for this case.

Case 2: In this case, we have $|\mathrm{center}(G)| = 2$. Let k_1 and k_2 denote the two components whose vertex representations lie in the center of the component graph,

i.e., $\text{center}(G) = \{k_1, k_2\}$. From the statement, we have $\mathcal{C}_2^\star = \{C_{2.1}^\star, C_{2.2}^\star\}$ where $C_{2.1}^\star = \{k_1\} \cup \text{nb}_G(k_1)$ and $C_{2.2}^\star = \{k_2\} \cup \text{nb}_G(k_2)$. We are given two bijective functions $f_{2.1} \colon C_{2.1}^\star \to \{1_{2.1}, \ldots, n_{2.1}\}$ with $n_{2.1} = |C_{2.1}^\star|$ and $f_{2.2} \colon C_{2.2}^\star \to \{1_{2.2}, \ldots, n_{2.2}\}$ with $n_{2.2} = |C_{2.2}^\star|$ where $f_{2.1}^{-1}(1_{2.1}) = k_1$, $f_{2.1}^{-1}(n_{2.1}) = k_2$ and $f_{2.2}^{-1}(1_{2.2}) = k_2$, $f_{2.2}^{-1}(n_{2.2}) = k_1$ because the eccentricity of k_1 and k_2 equals 2 and all other components have an eccentricity of 3.

From the premise, we know that for all $2_{2.1} \leq i_{2.1} < n_{2.1}$ holds

$$Sys[\{1_{2.1}, i_{2.1}, i_{2.1}+1\}] \setminus\setminus \hat{I}_{1_{2.1}, i_{2.1}} \approx_b^\Delta Sys[\{1_{2.1}, i_{2.1}+1\}] \setminus\setminus \hat{I}_{1_{2.1}, i_{2.1}}.$$

Thus, if we set in Theorem 5.7 the function f to $f_{2.1}$, we can conclude that

$$Sys[C_{2.1}^\star] \setminus\setminus \hat{I}_\dagger \approx_b^\Delta Sys[\{1_{2.1}, n_{2.1}\}] \setminus\setminus \hat{I}_\dagger \tag{$\dagger$}$$

holds with $\hat{I}_\dagger = \bigcup_{2_{2.1} \leq i_{2.1} < n_{2.1}} \hat{I}_{1_{2.1}, i_{2.1}}$ since each set $\hat{I}_{1_{2.1}, i_{2.1}}$ equals the respective set $\hat{I}_{1,i}$ of Theorem 5.7 if $f = f_{2.1}$.

Analogously, we can conclude (with $f := f_{2.2}$) that

$$Sys[C_{2.2}^\star] \setminus\setminus \hat{I}_\ddagger \approx_b^\Delta Sys[\{1_{2.2}, n_{2.2}\}] \setminus\setminus \hat{I}_\ddagger \tag{$\ddagger$}$$

holds with $\hat{I}_\ddagger = \bigcup_{2_{2.2} \leq i_{2.2} < n_{2.2}} \hat{I}_{1_{2.2}, i_{2.2}}$ since each set $\hat{I}_{1_{2.2}, i_{2.2}}$ equals the respective set $\hat{I}_{1,i}$ of Theorem 5.7 if $f = f_{2.2}$.

Observe that we have $\hat{I} = \hat{I}_\dagger \cup \hat{I}_\ddagger$ and $C_{2.1}^\star \cup C_{2.2}^\star = Comp$. Now, we have (the used proposition and equation is listed in parentheses at the end of each line):

$$\begin{aligned}
&Sys \setminus\setminus \hat{I} \\
&= Sys[C_{2.1}^\star \cup C_{2.2}^\star] \setminus\setminus \hat{I} && (3.20) \\
&= Sys[\{1_{2.1}, \ldots, n_{2.1}\} \cup \{1_{2.2}, \ldots, n_{2.2}\}] \setminus\setminus \hat{I} \\
&= Sys[\{1_{2.1}, \ldots, n_{2.1}\} \cup \{2_{2.2}, \ldots, n_{2.2}-1\}] \setminus\setminus \hat{I} \quad (1_{2.2} = n_{2.1} \text{ and } n_{2.2} = 1_{2.1}) \\
&= Sys[C_{2.1}^\star \cup C_{2.2}^\star \setminus \{1_{2.2}, n_{2.2}\}] \setminus\setminus \hat{I} \\
&= \Big(Sys[C_{2.1}^\star] \underset{\mathcal{I}_{C_{2.1}^\star, C_{2.2}^\star \setminus \{1_{2.2}, n_{2.2}\}}}{\otimes} Sys[C_{2.2}^\star \setminus \{1_{2.2}, n_{2.2}\}]\Big) \setminus\setminus \underbrace{\hat{I} \cup Int_{\text{closed}}}_{\hat{I}'} && (3.21) \\
&= \Big(Sys[C_{2.1}^\star] \underset{\mathcal{I}_{\{1_{2.1}, n_{2.1}\}, C_{2.2}^\star \setminus \{1_{2.2}, n_{2.2}\}}}{\otimes} Sys[C_{2.2}^\star \setminus \{1_{2.2}, n_{2.2}\}]\Big) \setminus\setminus \hat{I}' && (3.23) \\
&= \Big(Sys[C_{2.1}^\star] \setminus\setminus \hat{I}_\dagger \underset{\mathcal{I}_{\{1_{2.1}, n_{2.1}\}, C_{2.2}^\star \setminus \{1_{2.2}, n_{2.2}\}}}{\otimes} Sys[C_{2.2}^\star \setminus \{1_{2.2}, n_{2.2}\}]\Big) \setminus\setminus \hat{I}' && (3.15) \\
&\approx_b^\Delta \Big(Sys[\{1_{2.1}, n_{2.1}\}] \setminus\setminus \hat{I}_\dagger \underset{\mathcal{I}_{\{1_{2.1}, n_{2.1}\}, C_{2.2}^\star \setminus \{1_{2.2}, n_{2.2}\}}}{\otimes} Sys[C_{2.2}^\star \setminus \{1_{2.2}, n_{2.2}\}]\Big) \setminus\setminus \hat{I}' && ((\dagger)) \\
&= \Big(Sys[\{1_{2.1}, n_{2.1}\}] \underset{\mathcal{I}_{\{1_{2.1}, n_{2.1}\}, C_{2.2}^\star \setminus \{1_{2.2}, n_{2.2}\}}}{\otimes} Sys[C_{2.2}^\star \setminus \{1_{2.2}, n_{2.2}\}]\Big) \setminus\setminus \hat{I}' && (3.15) \\
&= Sys[\{1_{2.1}, n_{2.1}\} \cup C_{2.2}^\star \setminus \{1_{2.2}, n_{2.2}\}] \setminus\setminus \hat{I} \cup Int_{\text{closed}} && (3.21)
\end{aligned}$$

$$
\begin{aligned}
&= Sys[C^\star_{2.2}\}] \setminus\setminus \hat{I} && (1_{2.1} = n_{2.2} \text{ and } n_{2.1} = 1_{2.2})\\
&= \left(Sys[C^\star_{2.2}\}] \setminus\setminus \hat{I}_\ddagger\right) \setminus\setminus \hat{I} && (3.15)\\
&\approx^\Delta_b \left(Sys[\{1_{2.2}, n_{2.2}\}] \setminus\setminus \hat{I}_\ddagger\right) \setminus\setminus \hat{I} && (\text{Equation } (\ddagger))\\
&= Sys[\text{center}(G)] \setminus\setminus \hat{I} && (3.15)
\end{aligned}
$$

The last equation follows because we have $\{1_{2.2}, n_{2.2}\} = \{k_2, k_1\} = \text{center}(G)$. Note that when we applied Equations (†) and (‡) in the reasoning above, we also used Propositions 3.12 and 3.17. In the line that follows the line where we use Proposition 3.21 for the second time, we can drop Int_{closed} from the closing operator analogously to the reasoning in the proof of Theorem 5.7, i.e., all interactions of $Int[\{1_{2.1}, n_{2.1}\}]$ that are a subset of a closed interaction of the whole system Sys are also contained in $\hat{I}$ or $Int_{\text{closed}}[C^\star_{2.2}]$ respectively. This is ensured by the exclusive communication of the system.

The reasoning above shows the two base cases. We continue with the induction step and show that for all eccentricity levels $l \in \mathbb{N}$ with $\text{rad}(G) < l < \text{diam}(G)$ holds

$$
Sys[\bigcup_{\text{rad}(G)\le x\le l,\, 1\le y\le|\mathcal{C}^\star_x|} C^\star_{x.y}] \setminus\setminus \hat{I} \approx^\Delta_b Sys[\bigcup_{\text{rad}(G)\le x\le l-1,\, 1\le y\le|\mathcal{C}^\star_x|} C^\star_{x.y}] \setminus\setminus \hat{I}.
$$

To put it differently, we show that we can get rid of the current outermost eccentricity level with respect to index l. We have:

$$
\begin{aligned}
&Sys[\bigcup_{\text{rad}(G)\le x\le l,\, 1\le y\le|\mathcal{C}^\star_x|} C^\star_{x.y}] \setminus\setminus \hat{I}\\
&= Sys[\bigcup_{1\le y\le|\mathcal{C}^\star_l|} C^\star_{l.y} \cup \bigcup_{1\le y\le|\mathcal{C}^\star_{l-1}|} C^\star_{l-1.y} \cup \underbrace{\bigcup_{\text{rad}(G)\le x\le l-2,\, 1\le y\le|\mathcal{C}^\star_x|} C^\star_{x.y}}_{K}] \setminus\setminus \hat{I}\\
&= Sys[\bigcup_{1\le y\le|\mathcal{C}^\star_l|} C^\star_{l.y} \cup \bigcup_{1\le y'\le|\mathcal{C}^\star_{l-1}|} C^\star_{l-1.y'} \cup K] \setminus\setminus \hat{I}\\
&= Sys[\bigcup_{1\le y\le|\mathcal{C}^\star_l|} C^\star_{l.y} \cup \underbrace{\left(\bigcup_{1\le y'\le|\mathcal{C}^\star_{l-1}|} C^\star_{l-1.y'} \setminus \bigcup_{1\le y''\le|\mathcal{C}^\star_l|} \{1_{l.y''}, n_{l.y''}\}\right)}_{L} \cup K] \setminus\setminus \hat{I}
\end{aligned}
$$

Observe that for all indices y' we find a y'' such that $\{1_{l.y''}, n_{l.y''}\} \subseteq C^\star_{l-1.y'}$

$$
= Sys[\underbrace{\bigcup_{1\le y\le|\mathcal{C}^\star_l|} C^\star_{l.y}}_{C} \cup L \cup K] \setminus\setminus \hat{I}
$$

Since $C \cap (L \cup K) = \emptyset$ because of the observation, we can apply P. 3.21

$$
= \left(Sys[C] \underset{\mathcal{I}_{C,L}}{\otimes} Sys[L \cup K]\right) \setminus\setminus \underbrace{\hat{I} \cup Int_{\text{closed}}}_{\hat{I}'}
$$

$$= \Big(Sys[C] \underset{\mathcal{I}_{C',L}}{\otimes} Sys[L \cup K]\Big) \setminus\!\setminus \hat{I}' \quad (\text{with } C' = \bigcup_{1 \le y \le |\mathcal{C}_l^\star|} \{1_{l.y}, n_{l.y}\} \text{ and P. 3.23})$$

$$= \Big(Sys[C] \setminus\!\setminus \hat{I}_l \underset{\mathcal{I}_{C',L}}{\otimes} Sys[L \cup K]\Big) \setminus\!\setminus \hat{I}' \quad (\text{with } \hat{I}_l = \bigcup_{1 \le y \le |\mathcal{C}_l^\star|, 2_{l.y} \le i_{l.y} < n_{l.y}} \hat{I}_{1_{l.y}, i_{l.y}})$$

We could apply Proposition 3.15 here since $\hat{I}_l \subseteq \hat{I}'$

$$= \Big(Sys[\bigcup_{1 \le y \le |\mathcal{C}_l^\star|} C_{l.y}^\star] \setminus\!\setminus \hat{I}_l \underset{\mathcal{I}_{C',L}}{\otimes} Sys[L \cup K]\Big) \setminus\!\setminus \hat{I}'$$

$$\approx_b^\Delta \Big(Sys[\bigcup_{1 \le y \le |\mathcal{C}_l^\star|} \{1_{l.y}, n_{l.y}\}] \setminus\!\setminus \hat{I}_l \underset{\mathcal{I}_{C',L}}{\otimes} Sys[L \cup K]\Big) \setminus\!\setminus \hat{I}' \quad (\text{see below})$$

$$= \Big(Sys[\bigcup_{1 \le y \le |\mathcal{C}_l^\star|} \{1_{l.y}, n_{l.y}\}] \underset{\mathcal{I}_{C',L}}{\otimes} Sys[L \cup K]\Big) \setminus\!\setminus \hat{I}' \quad (\text{Proposition 3.15})$$

$$= \Big(Sys[\bigcup_{1 \le y \le |\mathcal{C}_l^\star|} \{1_{l.y}, n_{l.y}\} \cup L \cup K]\Big) \setminus\!\setminus \hat{I} \cup Int_{\text{closed}} \quad (\text{Proposition 3.21})$$

$$= Sys[\bigcup_{1 \le y \le |\mathcal{C}_l^\star|} \{1_{l.y}, n_{l.y}\} \cup \big(\bigcup_{1 \le y' \le |\mathcal{C}_{l-1}^\star|} C_{l-1.y'}^\star \setminus \bigcup_{1 \le y'' \le |\mathcal{C}_l^\star|} \{1_{l.y''}, n_{l.y''}\}\big) \cup K] \setminus\!\setminus \hat{I}$$

$$= Sys[\bigcup_{1 \le y \le |\mathcal{C}_l^\star|} \{1_{l.y}, n_{l.y}\} \cup \bigcup_{1 \le y' \le |\mathcal{C}_{l-1}^\star|} C_{l-1.y'}^\star \cup K] \setminus\!\setminus \hat{I}$$

$$= Sys[\bigcup_{1 \le y \le |\mathcal{C}_{l-1}^\star|} C_{l-1.y}^\star \cup \bigcup_{\text{rad}(G) \le x \le l-2, 1 \le y \le |\mathcal{C}_x^\star|} C_{x.y}^\star] \setminus\!\setminus \hat{I} \quad (\text{observation above})$$

$$= Sys[\bigcup_{\text{rad}(G) \le x \le l-1, 1 \le y \le |\mathcal{C}_x^\star|} C_{x.y}^\star] \setminus\!\setminus \hat{I}$$

Observe that this reasoning proves our inductive step if we show that

$$Sys[\bigcup_{1 \le y \le |\mathcal{C}_l^\star|} C_{l.y}^\star] \setminus\!\setminus \hat{I}_l \overset{?}{\approx}_b^\Delta Sys[\bigcup_{1 \le y \le |\mathcal{C}_l^\star|} \{1_{l.y}, n_{l.y}\}] \setminus\!\setminus \hat{I}_l \qquad (\star)$$

holds where $\hat{I}_l = \bigcup_{1 \le y \le |\mathcal{C}_l^\star|, 2_{l.y} \le i_{l.y} < n_{l.y}} \hat{I}_{1_{l.y}, i_{l.y}}$. Then, the reasoning above is valid because of Propositions 3.12 and 3.17. Again, exclusive communication ensures that we can drop the set Int_{closed} from the closing operator in the line that follows the line where we use Proposition 3.21 for the second time.

We show Equation ($\star$) by an inductive argument over index y (or the size of the set $\mathcal{C}_l^\star$), i.e., we show for all indices $k \in \mathbb{N}$ with $1 \le k \le |\mathcal{C}_l^\star|$ that

$$Sys[\bigcup_{1 \le y \le k} C_{l.y}^\star \cup \bigcup_{k < y \le |\mathcal{C}_l^\star|} \{1_{l.y}, n_{l.y}\}] \setminus\!\setminus \hat{I}_l \approx_b^\Delta Sys[\bigcup_{1 \le y \le k-1} C_{l.y}^\star \cup \bigcup_{k-1 < y \le |\mathcal{C}_l^\star|} \{1_{l.y}, n_{l.y}\}] \setminus\!\setminus \hat{I}_l$$

holds. In order to be able to prove this claim, we first show that for each fixed k as above (i.e., with $1 \le k \le |\mathcal{C}_l^\star|$) it holds that

$$Sys[C_{l.k}^\star] \setminus\!\setminus \hat{I}_{l.k} \approx_b^\Delta Sys[\{1_{l.k}, n_{l.k}\}] \setminus\!\setminus \hat{I}_{l.k}$$

where $\hat{I}_{l.k} = \bigcup_{2_{l.k} \leq i_{l.k} < n_{l.k}} \hat{I}_{1_{l.k}, i_{l.k}}$. From the premise, we know that for all $2_{l.k} \leq i_{l.k} < n_{l.k}$ holds

$$Sys[\{1_{l.k}, i_{l.k}, i_{l.k}+1\}] \setminus\setminus \hat{I}_{1_{l.k}, i_{l.1}} \approx_b^{\Delta} Sys[\{1_{l.k}, i_{l.k}+1\}] \setminus\setminus \hat{I}_{1_{l.k}, i_{l.k}}.$$

Thus, if we set in Theorem 5.7 the function f to $f_{l.k}$, we can conclude that the claim holds since each set $\hat{I}_{1_{l.k}, i_{l.k}}$ in $\hat{I}_{l.k}$ equals the respective set $\hat{I}_{1,i}$ of Theorem 5.7 if $f = f_{l.k}$.

We proceed with the inductive argument to show Equation ($\star$):

$$\begin{aligned}
&Sys[\bigcup_{1 \leq y \leq k} C^{\star}_{l.y} \cup \bigcup_{k < y \leq |\mathcal{C}^{\star}_l|} \{1_{l.y}, n_{l.y}\}] \setminus\setminus \hat{I}_l \\
&= Sys[C^{\star}_{l.k} \cup \bigcup_{1 \leq y \leq k-1} C^{\star}_{l.y} \cup \bigcup_{k < y \leq |\mathcal{C}^{\star}_l|} \{1_{l.y}, n_{l.y}\}] \setminus\setminus \hat{I}_l \\
&= Sys[C^{\star}_{l.k} \cup \underbrace{(\bigcup_{1 \leq y \leq k-1} C^{\star}_{l.y} \cup \bigcup_{k < y \leq |\mathcal{C}^{\star}_l|} \{1_{l.y}, n_{l.y}\}) \setminus \{1_{l.k}, n_{l.k}\}}_{K}] \setminus\setminus \hat{I}_l \\
&= \Big(Sys[C^{\star}_{l.k}] \underset{\mathcal{I}_{C^{\star}_{l.k}, K}}{\otimes} Sys[K]\Big) \setminus\setminus \underbrace{\hat{I}_l \cup Int_{\text{closed}}[C^{\star}_{l.k} \cup K]}_{\hat{I}'_l} && \text{(Proposition 3.21)} \\
&= \Big(Sys[C^{\star}_{l.k}] \underset{\mathcal{I}_{\{1_{l.k}, n_{l.k}\}, K}}{\otimes} Sys[K]\Big) \setminus\setminus \hat{I}'_l && \text{(Proposition 3.23)} \\
&= \Big(Sys[C^{\star}_{l.k}] \setminus\setminus \hat{I}_{l.k} \underset{\mathcal{I}_{\{1_{l.k}, n_{l.k}\}, K}}{\otimes} Sys[K]\Big) \setminus\setminus \hat{I}'_l && \text{(Proposition 3.15 and } \hat{I}_{l.k} \subseteq \hat{I}'_l) \\
&\approx_b^{\Delta} \Big(Sys[\{1_{l.k}, n_{l.k}\}] \setminus\setminus \hat{I}_{l.k} \underset{\mathcal{I}_{\{1_{l.k}, n_{l.k}\}, K}}{\otimes} Sys[K]\Big) \setminus\setminus \hat{I}'_l && \text{(see above \& P. 3.12, 3.17)} \\
&= \Big(Sys[\{1_{l.k}, n_{l.k}\}] \underset{\mathcal{I}_{\{1_{l.k}, n_{l.k}\}, K}}{\otimes} Sys[K]\Big) \setminus\setminus \hat{I}'_l && \text{(Proposition 3.15)} \\
&= \Big(Sys[\{1_{l.k}, n_{l.k}\} \cup K]\Big) \setminus\setminus \hat{I}_l \cup Int_{\text{closed}}[C^{\star}_{l.k} \cup K] && \text{(Proposition 3.21)} \\
&= Sys[\{1_{l.k}, n_{l.k}\} \cup (\bigcup_{1 \leq y \leq k-1} C^{\star}_{l.y} \cup \bigcup_{k < y \leq |\mathcal{C}^{\star}_l|} \{1_{l.y}, n_{l.y}\}) \setminus \{1_{l.k}, n_{l.k}\}] \setminus\setminus \hat{I}_l \\
&= Sys[\{1_{l.k}, n_{l.k}\} \cup \bigcup_{1 \leq y \leq k-1} C^{\star}_{l.y} \cup \bigcup_{k < y \leq |\mathcal{C}^{\star}_l|} \{1_{l.y}, n_{l.y}\}] \setminus\setminus \hat{I}_l \\
&= Sys[\bigcup_{1 \leq y \leq k-1} C^{\star}_{l.y} \cup \bigcup_{k-1 < y \leq |\mathcal{C}^{\star}_l|} \{1_{l.y}, n_{l.y}\}] \setminus\setminus \hat{I}_l
\end{aligned}$$

In the line that follows the line where we use Proposition 3.21 for the second time, exclusive communication, as discussed above, ensures that we can drop the set $Int_{\text{closed}}[C^{\star}_{l.k} \cup K]$ from the closing operator.

Thus, the induction lets us conclude that the theorem holds. ∎

Proof of Corollary 5.14: If $|Comp| < 3$ the premise of the corollary is false, i.e., the corollary holds. Thus, assume that $|Comp| \geq 3$ and that the premise of the corollary holds. Number the components via a bijective function $f : Comp \to \{1, \ldots, n\}$ with $n = |Comp|$ in an arbitrary way with number 1 being the middle component m and number n being the chosen border component k, i.e., $f(m) = 1$ and $f(k) = n$ holds. Then apply Theorem 5.13. ∎

F.5 Proofs from Chapter 6

Proof of Lemma 6.2: Assume that *Sys* contains a deadlock $s \in S$. Consider the local states s_i of the components in s. Since no interaction is enabled in s, for all such s_i it holds that all interactions $\alpha \in Int(s_i)$ are blocked, i.e., for each α, we find a component $j \in Comp$ with $j(\alpha) \neq \emptyset$ and $\alpha \notin Int(s_j)$—otherwise α is enabled in s. Thus, each component i waits for a component j because of such an α. Since there is only a finite number of components, at least one component k must wait for an already considered component, i.e., we get: i waits for j which waits for ... which waits for k, and k waits for a component between i and k. Considering the components of this cycle and ordering them according to the waiting relation results in the statement of the lemma. ∎

Proof of Lemma 6.4: Assume that the cooperation graph $G_{\text{coop}} = (V, E)$, components $i, j \in Comp$, and interaction $\alpha \in Int$ with $i \neq j$ and $\{i, j\} \subseteq \text{compset}(\alpha)$ are given. Since the cooperation graph is connected by assumption, we can find by depth-first search a path $\pi^{\alpha}_{i,j}$ with $|\pi^{\alpha}_{i,j}| = k$ for $k \in \mathbb{N}$, $\pi^{\alpha}_{i,j}[0] = \{i\}$, $\pi^{\alpha}_{i,j}[k-1] = \{j\}$, and $\pi^{\alpha}_{i,j}[k'] = \text{compset}(\alpha)$ for $k' \in \mathbb{N}$ with $0 < k' < k-1$, i.e., $\pi^{\alpha}_{i,j}$ is a cooperation path (cf. Definition 6.3). Thus, we have $\pi^{\alpha}_{i,j} = \langle v^0, \ldots, v^{k-1} \rangle$ according to Definition A.3 with $v^0 = \{i\}$, $v^{k-1} = \{j\}$, and $v^{k'} = \text{compset}(\alpha)$. We have $v^0 \subseteq v^{k'}$ and $v^{k-1} \subseteq v^{k'}$ (Property 1 for v^0 and v^{k-1}), and since all these vertices are different, we conclude $|\pi^{\alpha}_{i,j}| \geq 3$ (Property 3). From the definition of the cooperation graph (cf. Definition 4.4) and the reachability of $v^{k'}$ from v^0, we know that there is a path from v^0 to $v^{k'}$ such that for all intermediate vertices $v^{l'}$ with $0 < l' < k'$ holds $v^0 \subset v^{l'} \subset v^{k'}$ since v^0 and v^1 are only connected if $v^0 \subset v^1$, $v^{k'}$ and $v^{k'-1}$ are only connected if $v^{k'-1} \subset v^{k'}$, and a similar reasoning also shows that this holds for all pairs of successive intermediate vertices (cf. the definition of the edges in the cooperation graph in Definition 4.4). Observe that $v^0 \subset v^{l'}$ and $v^{l'} \subset v^{k'}$ for all $0 < l' < k'$ implies $|v^{l'}| \geq 2$ and $v^{l'} \subseteq \text{compset}(\alpha)$ (Properties 1 and 2 for all $v^{l'}$). Analogously, we know from the reachability of v^{k-1} from $v^{k'}$ that there is a path from $v^{k'}$ to v^{k-1} such that for all intermediate vertices $v^{l''}$ with $k' < l'' < k-1$ holds $v^{k'} \supset v^{l''} \supset v^{k-1}$, i.e., also $|v^{l''}| \geq 2$ and $v^{l''} \subseteq \text{compset}(\alpha)$

holds (Properties 1 and 2 for all $v^{l''}$). Reconstructing $\pi^{\alpha}_{i,j}$ with v^0, $v^{k'}$, v^{k-1}, and all intermediate vertices $v^{l'}$, $0 < l' < k'$, and $v^{l''}$, $k' < l'' < k-1$, i.e., $\pi^{\alpha}_{i,j} = (v^0, \ldots, v^{l'}, \ldots, v^{k'}, \ldots, v^{l''}, \ldots, v^{k-1})$, results in a cooperation path for which Properties 1, 2, and 3 hold. ∎

Proof of Lemma 6.5: First observe that if *Sys* has a disjoint circular wait free architecture, then its cooperation graph is connected. We prove the claim by contradiction. Assume that no such paths exist, i.e., for all components $i, j, k \in \mathit{Comp}$ and interactions $\alpha, \beta \in \mathit{Int}$ it holds that all cooperation paths $\pi^{\alpha}_{i,j}$ and $\pi^{\beta}_{j,k}$ only have vertices in common that represent components, i.e., $\forall\, v \in V\colon v \in \pi^{\alpha}_{i,j} \;\wedge\; v \in \pi^{\beta}_{j,k} \implies |v| < 2$. By Lemma 6.2, we know—because of the deadlock—that a set $D \subseteq \mathit{Comp}$ exists such that we can order the components in D in the following way: For $0 \leq l < |D|$ it holds (with $i_{|D|} = i_0$) that component $i_l \in D$ wants to perform an interaction $\alpha_l \in \mathit{Int}$, component $i_{l+1} \in D$ is needed by i_l to perform α_l, and i_{l+1} is unable to perform α_l. We now consider the corresponding cooperation paths $\pi^{\alpha_l}_{i_l,i_{l+1}}$ for all $0 \leq l < |D|$. By concatenating these paths, we get a path π that contains all vertices representing components in D. Since we assumed that no two cooperation paths share a common vertex that does not represent a component, the path π also contains a simple cycle in the cooperation graph G_{coop}. But, this contradicts the disjoint circular wait freedom of *Sys*'s architecture because now a simple cycle exists in G_{coop} that contains more than one vertex that represents a component. Thus, there must be a component $i_{l'} \in D$ with $0 \leq l' < |D|$ such that the cooperation paths $\pi^{\alpha_{l'}}_{i_{l'},i_{l'+1}}$ and $\pi^{\alpha_{l'+1}}_{i_{l'+1},i_{l'+2}}$ (with $i_{|D|} = i_0$, $i_{|D|+1} = i_1$, and $\alpha_{|D|} = \alpha_0$) have a vertex v in common that does not represent a component, i.e., $|v| \geq 2$ holds. By setting $i := i_{l'}$, $j := i_{l'+1}$, $k := i_{l'+2}$, $\alpha := \alpha_{l'}$, and $\beta := \alpha_{l'+1}$, the claim follows. ∎

Proof of Theorem 6.6: According to Lemma 6.5, we can find components $i, j, k \in \mathit{Comp}$ and interactions $\alpha, \beta \in \mathit{Int}$ with $i \neq j$, $j \neq k$, $\{i, j\} \subseteq \text{compset}(\alpha)$, and $\{j, k\} \subseteq \text{compset}(\beta)$ such that a vertex v with $|v| \geq 2$ of the cooperation graph G_{coop} exists that is contained in both cooperation paths of the lemma, i.e., $v \in \pi^{\alpha}_{i,j}$ and $v \in \pi^{\beta}_{j,k}$. From the proof of Lemmata 6.2 and 6.5, we know that a set $D \subseteq \mathit{Comp}$ and a component $i_l \in D$ exist such that these components and interactions can be chosen such that in the deadlocked global state s component $i = i_l$ is able to perform interaction $\alpha = \alpha_l$, i.e., $\alpha \in \mathit{Int}(s_i)$, component $j = i_{l+1}$ is able to perform interaction $\beta = \alpha_{l+1}$, i.e., $\beta \in \mathit{Int}(s_j)$, and j is unable to perform α, i.e., $\alpha \notin \mathit{Int}(s_j)$. Thus, all we have to show is $|\text{compset}(\alpha) \cap \text{compset}(\beta)| \geq 2$. By Lemma 6.4, we know that for all vertices on the cooperation paths $\pi^{\alpha}_{i,j}$ and $\pi^{\beta}_{j,k}$ it holds that they are subsets of the

components participating in the interactions α and β respectively. Thus, for vertex v from above, that occurs on both paths, holds $v \subseteq \text{compset}(\alpha)$ and $v \subseteq \text{compset}(\beta)$, i.e., $v \subseteq \text{compset}(\alpha) \cap \text{compset}(\beta)$. Since $|v| \geq 2$, also $|\text{compset}(\alpha) \cap \text{compset}(\beta)| \geq 2$ holds. ■

Proof of Corollary 6.7: Observe that the statement of the corollary corresponds to the negation of the statement of Theorem 6.6 with the following adjustment: Instead of requiring that there is an interaction $\beta \in Int(s_j)$ with $|\text{compset}(\alpha) \cap \text{compset}(\beta)| \geq 2$, we can check whether $\text{compset}(\alpha) \cap \text{coopset}(s_j) \neq \emptyset$ holds, since this implies that there is such an interaction β. ■

Proof of Theorem 6.9: Directly follows from Corollary 6.7, since no problematic states exist, i.e, no two components $i, j \in Comp$, interaction $\alpha \in Int$, and local states $s_i \in S_i$ and $s_j \in S_j$ with $i \neq j$, $\{i, j\} \subseteq \text{compset}(\alpha)$, $\alpha \in Int(s_i)$, $\alpha \notin Int(s_j)$, and $\text{compset}(\alpha) \cap \text{coopset}(s_j) \neq \emptyset$ exist because all $\text{PS}_j(s_i, \alpha) \subseteq \{s_j \in S_j \mid \alpha \notin Int(s_j) \wedge \text{compset}(\alpha) \cap \text{coopset}(s_j) \neq \emptyset\}$ are empty. If $\text{PS}_j(s_i, \alpha)$ is a proper subset, then all other states in the superset are either independent or any combination (s_i, s_j) of corresponding states is either not reachable or able to perform an interaction in which only i and j participate globally (cf. Definition 6.8), i.e., this state combination cannot be part of a reachable deadlock. ■

Proof of Theorem 6.12: Let *Sys* be an interaction system which has a disjoint circular wait free architecture. We assume $Int_{\text{closed}} = \emptyset$ for the proof, i.e., $Int = Int_{\text{open}}$, and address the presence of closed interactions at the end. Let $G_{\text{coop}} = (V, E)$ be *Sys*'s cooperation graph. We prove the claim by contradiction. Assume that the two conditions of Theorem 6.12 hold, although *Sys* is not deadlock-free.

Before we continue, we want to give short sketch of the main idea for the proof: We now consider an arbitrary reachable deadlock and take a look at the entry interactions of a certain set D of components where for all corresponding local states there is a problematic state of another component that is also contained in D. We show how such a set D can be found (which has to exist because of the disjoint circular wait free architecture). Afterwards, we show that either condition one is violated if there is no interaction on the path to the deadlock where only components in D or at least one cycle component, i.e., a component whose vertex representation lies on a simple cycle in G_{coop} (cf. Definition 4.6), in D participate in, or there is such an interaction and we have a contradiction to the second condition.

Thus, there is a deadlocked global state $s \in S$ and a sequence σ starting in a global initial state $s^0 \in S^0$ with $s^0 \xrightarrow{\alpha_0} s^1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_m} s$ for a fixed $m \in \mathbb{N}$. First, we show that we can choose a set $D \subseteq \mathit{Comp}$ with $|D| \geq 2$ similarly to Lemma 6.2 but with additional properties that are helpful in the following: For all components $i \in D$ and all $\alpha \in \mathit{Int}(s_i)$ it holds that

(P1) a component $j \in \mathrm{compset}(\alpha) \setminus \{i\}$ exists with $j \in D$ and $s_j \in \mathrm{PS}_j(s_i, \alpha)$,

(P2) for all $\beta \in \mathit{Int}$ with $|\mathrm{compset}(\alpha) \cap \mathrm{compset}(\beta)| \geq 2$ holds $\mathrm{cycleset}(\beta) \cap D \neq \emptyset$ or $\mathrm{compset}(\beta) \subseteq D$, and

(P3) for all $\beta \in \mathit{Int}$ with $|\mathrm{compset}(\alpha) \cap \mathrm{compset}(\beta)| \geq 2$ it holds that a $\beta' \in \mathit{Int}$ exists with $i(\beta) \cap \beta' \neq \emptyset$ and $\mathrm{compset}(\beta') \cap \mathrm{coopset}(s_i) \neq \emptyset$.

We show how such a set D can be found—afterwards, we argue why this is always possible. We modify the cooperation graph G_{coop} in the following way: Set $G' = (V', E')$ with $V' = V$ and $E' = E$. We repeat the following steps until a fixed point is reached: Consider every component $i \in \mathit{Comp}$ whose vertex representation $\{i\}$ is contained in V': If an interaction $\alpha \in \mathit{Int}(s_i)$ and a component $j \in \mathrm{compset}(\alpha) \setminus \{i\}$ exist such that $\mathrm{compset}(\alpha) \cap \mathrm{coopset}(s_j) = \emptyset$, then: (i) If $\mathrm{cycleset}(\alpha) = \emptyset$, consider the second last vertex on the cooperation path $\pi^{\alpha}_{i,j}$, i.e., the vertex v that is adjacent to the vertex w representing j. Remove the edge between vertex v and w, this divides G' into two connected components since $\mathrm{cycleset}(\alpha) = \emptyset$. Keep only the connected component that contains w as the new G'. (ii) Otherwise, i.e., if $\mathrm{cycleset}(\alpha) \neq \emptyset$, remove i from all vertices, i.e., any vertex $v \in V'$ becomes $v \setminus \{i\}$. If this removal creates a vertex that corresponds to the empty set, we remove this vertex and all incident edges. Again, if this removal divides G' into connected components, keep only the one that contains the vertex representing j as the new G'.

After this modification, a set D of components remains in V' with $|D| \geq 2$. Otherwise, *Sys*'s architecture is not disjoint circular wait free, because if there is no such set D, pick a component $i \in \mathit{Comp}$. We know that $\alpha \in \mathit{Int}(s_i)$ and $j \in \mathrm{compset}(\alpha) \setminus \{i\}$ exist such that $\mathrm{compset}(\alpha) \cap \mathrm{coopset}(s_i) = \emptyset$. Fix the cooperation path $\pi^{\alpha}_{i,j}$. Consider j. Again, there is a $\beta \in \mathit{Int}(s_j)$ and a $k \in \mathrm{compset}(\beta) \setminus \{j\}$ with $\mathrm{compset}(\beta) \cap \mathrm{coopset}(s_j) = \emptyset$. Fix the cooperation path $\pi^{\beta}_{j,k}$. Now, we know that these paths do not share a common vertex v with $|v| \geq 2$ because otherwise, there is a component $l \in \mathrm{compset}(\alpha) \cap \mathrm{compset}(\beta)$ with $l \neq j$. Then, $l \in \mathrm{coopset}(s_j)$ since $\beta \in \mathit{Int}(s_j)$. But, since $l \in \mathrm{compset}(\alpha)$ this implies $\mathrm{compset}(\alpha) \cap \mathrm{coopset}(s_j) \neq \emptyset$ which contradicts our assumption about G'. Thus, no vertex v exists that is contained in both paths. Now, we repeat this argument for all components. Since s is a deadlock and *Comp* is finite, at least one component waits for an already considered one (similar

to the proof of Lemma 6.2), i.e., we can concatenate all above constructed cooperation paths to a cycle in the original version of G_{coop}. But, this contradicts the assumption that *Sys* has a disjoint circular wait free architecture since this cycle contains a simple cycle where more than one vertex representing a component lies on. Thus, we find a set $D \subseteq \mathit{Comp}$ with $|D| \geq 2$.

Next, we show that also the properties P1, P2, and P3 hold for D. For all $i \in D$ and all $\alpha \in \mathit{Int}(s_i)$, we find a $j \in \text{compset}(\alpha) \setminus \{i\}$ such that $j \in D$ and $\text{compset}(\alpha) \cap \text{coopset}(s_j) \neq \emptyset$ holds, because otherwise, the vertex representing i is not contained in the graph G' where the set D is extracted from. Analogously as in the proof of Theorem 6.9, we can then conclude that also $s_j \in \text{PS}_j(s_i, \alpha)$ holds, i.e., we have established Property P1 of D. Next, we show that also Property P2 holds. By way of contradiction, assume that there is an $i \in D$, $\alpha \in \mathit{Int}(s_i)$, and $\beta \in \mathit{Int}$ such that $|\text{compset}(\alpha) \cap \text{compset}(\beta)| \geq 2$ holds but $\text{cycleset}(\beta) \cap D = \emptyset$ and there is a $k \in \mathit{Comp} \setminus D$ with $k \in \text{compset}(\beta)$. Assume $\text{cycleset}(\beta) = \emptyset$. Pick a component $j \in \text{compset}(\alpha) \cap \text{compset}(\beta)$. Since $i \in D$, also $j \in D$ holds. Consider the cooperation path $\pi^{\beta}_{k,j}$. Since $k \notin D$, the edge connecting the second last vertex on the path and vertex $\{j\}$ was removed in the construction of G'. But, since the connected component containing $\{j\}$ is kept, the vertex $\text{compset}(\alpha) \cap \text{compset}(\beta)$ and thus also the vertex $\{i\}$ are not contained in V'. This contradicts $i \in D$. Thus, assume $\text{cycleset}(\beta) \neq \emptyset$. All components $j \in \text{compset}(\alpha) \cap \text{compset}(\beta)$ are contained in D since $i \in D$. Since $\text{cycleset}(\beta) \cap D = \emptyset$, for all components $k \in \text{cycleset}(\beta)$ holds $k \notin D$ and thus $k \notin \text{compset}(\alpha) \cap \text{compset}(\beta)$. Because each k was removed in the construction of G', we find at least one $k \in \text{cycleset}(\beta)$ with $\beta \in \mathit{Int}(s_k)$ and a $j \in \text{compset}(\alpha) \cap \text{compset}(\beta)$ as above with $\text{compset}(\beta) \cap \text{coopset}(s_j) = \emptyset$—since the connected component that contains $\{j\}$ is kept in the construction of G'. Consider the cooperation path $\pi^{\beta}_{k,j}$. We also have $(\text{compset}(\alpha) \cap \text{compset}(\beta)) \cap \text{coopset}(s_j) = \emptyset$. But, this means that for component i and interaction α, we found a component j with $\text{compset}(\alpha) \cap \text{coopset}(s_j) = \emptyset$. Again, this contradicts $i \in D$. Thus, Property P2 holds for D.

In order to see why Property P3 holds too, again assume that there is an $i \in D$, $\alpha \in \mathit{Int}(s_i)$, and $\beta \in \mathit{Int}$ such that $|\text{compset}(\alpha) \cap \text{compset}(\beta)| \geq 2$ holds but for all $\beta' \in \mathit{Int}$ with $i(\beta) \cap \beta' \neq \emptyset$ holds $\text{compset}(\beta') \cap \text{coopset}(s_i) = \emptyset$. Consider a component $j \in \text{compset}(\beta) \cap \text{compset}(\alpha)$. As above, we know that $j \in D$. Now, we know by assumption that for all interactions $\gamma \in \mathit{Int}(s_j)$ with $i \in \text{compset}(\gamma)$ holds $\text{compset}(\gamma) \cap \text{coopset}(s_i) = \emptyset$. There is at least one such γ for a j as above since $i, j \in D$. But, this means $j \notin D$ since j is removed in the construction of G' because of i. Thus, Property P3 holds for D.

Next, we consider sequence σ given at the beginning of the proof that leads

to deadlock s where we use (as already mentioned after the statement of the theorem) the abbreviations $\text{PEI}(s_i, \beta) := \bigcup_{j \in \text{compset}(\beta) \setminus \{i\}} \bigcup_{s_j \in \text{PS}_j(s_i, \beta)} \text{EI}(s_i, s_j)$ and $\text{IPEI}(s_i) := \bigcap_{\beta \in Int(s_i)} \text{PEI}(s_i, \beta)$ for a state $s_i \in S_i$ and an interaction $\beta \in Int$. We distinguish two cases:

1) For all indices l of sequence σ with $0 \leq l \leq m$ holds $|\text{compset}(\alpha_l)| = 1$ or $\text{cycleset}(\alpha_l) \cap D = \emptyset$ and there is a $k \in Comp \setminus D$ with $k \in \text{compset}(\alpha_l)$. Then, for all $i \in D$ and all such α_l holds: Either $i(\alpha_l) = \emptyset$, $i(\alpha_l) \neq \emptyset$ and $|\alpha_l| = 1$, or $i(\alpha_l) \neq \emptyset$ and $|\alpha_l| \geq 2$. In the last case, we know that $\text{cycleset}(\alpha_l) = \emptyset$ holds, since otherwise $\text{cycleset}(\alpha_l) \cap D \neq \emptyset$ holds because $i \in D$. We show that in either case $\text{compset}(\alpha_l) \cap \text{coopset}(s_i) = \emptyset$ holds. For the already considered cases we are done. But in the last case, we know that a $k \in Comp \setminus D$ exists with $k \in \text{compset}(\alpha_l)$. Assume that a $j \in \text{compset}(\alpha_l) \setminus \{i\}$ exists with $j \in \text{coopset}(s_i)$. Then, there must be a $\beta \in Int(s_i)$ with $j \in \text{compset}(\beta)$. We conclude $\{i, j\} \subseteq \text{compset}(\beta) \cap \text{compset}(\alpha_l)$ and thus $|\text{compset}(\beta) \cap \text{compset}(\alpha_l)| \geq 2$. Since $i \in D$ and $\beta \in Int(s_i)$, Property P2 of D and $\text{cycleset}(\alpha_l) = \emptyset$ implies $\text{compset}(\alpha_l) \subseteq D$. But, this contradicts $k \in \text{compset}(\alpha_l)$ and $k \notin D$. Thus, $\text{compset}(\alpha_l) \cap \text{coopset}(s_i) = \emptyset$ holds. Summarizing, we get for all $i \in D$ that $\text{NBRS}(s_i) \cap S_i^0 \neq \emptyset$ since each action of such an i occurring on sequence σ is not an action that is (only) used for cooperation with components that i wants to cooperate with in s_i. From Property P1 of D, we know that for all $i \in D$ and all $\alpha \in Int(s_i)$ it holds that a $j \in \text{compset}(\alpha) \setminus \{i\}$ exists with $j \in D$ and $s_j \in \text{PS}_j(s_i, \alpha)$. For all such j, the same reasoning as above shows $\text{NBRS}(s_j) \cap S_j^0 \neq \emptyset$. Thus, we can conclude that for all $i \in D$ and all $\alpha \in Int(s_i)$ it holds that a component $j \in \text{compset}(\alpha) \setminus \{i\}$ exists with $\left(\bigcup_{s_j \in \text{PS}_j(s_i, \alpha)} \text{NBRS}(s_j)\right) \cap S_j^0 \neq \emptyset$. But this contradicts Condition 1 of the theorem.

2) From the reasoning in Case 1, we know that we can find a maximal index l with $0 \leq l \leq m$ for sequence σ with $|\text{compset}(\alpha_l)| \geq 2$ and $\text{cycleset}(\alpha_l) \cap D \neq \emptyset$ or $\text{compset}(\alpha_l) \subseteq D$. For all subsequent interactions on σ, i.e., all $\alpha_{l'}$ with $l < l' \leq m$, holds $|\text{compset}(\alpha_{l'})| = 1$ or $\text{cycleset}(\alpha_{l'}) \cap D = \emptyset$ and $\exists\, k \in Comp \setminus D \colon k \in \text{compset}(\alpha_{l'})$. Thus, an analogous argument as in Case 1 shows that for all $i \in D$ we have $s_i^{l+1} \in \text{NBRS}(s_i)$, i.e., the local state after the execution of α_l on σ is in the non-interfering backward reachable set of the local part of the deadlock. Since $|\text{compset}(\alpha_l)| \geq 2$, we have to distinguish two subcases:

2.1) $\text{compset}(\alpha_l) \subseteq D$. We already showed that $s_i^{l+1} \in \text{NBRS}(s_i)$ holds for all $i \in D$. Choose a component $i \in \text{compset}(\alpha_l)$ and a $\beta \in Int(s_i)$. From Property P1 of D, we know that a $j \in \text{compset}(\beta) \setminus \{i\}$ exists with $j \in D$ and $s_j \in \text{PS}_j(s_i, \beta)$. Since $j \in D$, we can conclude that $(s_i^{l+1}, s_j^{l+1}) \in \text{NBRS}(s_i) \times \text{NBRS}(s_j)$ holds. From Property P3 of D, we have $\exists\, \alpha' \in Int \colon i(\alpha_l) \cap \alpha' \neq$

$\emptyset \wedge \text{compset}(\alpha') \cap \text{coopset}(s_i) \neq \emptyset$. This implies $\alpha_l \in \text{EI}(s_i, s_j)$ since (s_i^l, s_j^l) is reachable in $[\![Sys[\{i,j\}]]\!]$ and $(s_i^l, s_j^l) \in \text{Pre}((s_i^{l+1}, s_j^{l+1}), \{\alpha_l \cap (A_i \cup A_j)\})$. Now, we see that for all $\beta \in Int(s_i)$, we have $\alpha_l \in \text{PEI}(s_i, \beta)$ and thus $\alpha_l \in \text{IPEI}(s_i)$. Analogously as for i, we get $\alpha_l \in \text{IPEI}(s_k)$ for all $k \in \text{compset}(\alpha_l)$. But, this contradicts Condition 2 of the theorem.

2.2) $\text{cycleset}(\alpha_l) \cap D \neq \emptyset$. Choose a component $i \in \text{cycleset}(\alpha_l) \cap D$. From Property P1 of D, we know that for all $\beta \in Int(s_i)$ a $j \in \text{compset}(\beta) \setminus \{i\}$ exists with $j \in D$ and $s_j \in \text{PS}_j(s_i, \beta)$. From above we can conclude that $(s_i^{l+1}, s_j^{l+1}) \in \text{NBRS}(s_i) \times \text{NBRS}(s_j)$ holds since $j \in D$. A similar reasoning as in Case 2.1 shows that Property P3 of D implies $\alpha_l \in \text{EI}(s_i, s_j)$. Now, we see that for all $\beta \in Int(s_i)$, we have $\alpha_l \in \text{PEI}(s_i, \beta)$ and thus $\alpha_l \in \text{IPEI}(s_i)$. But again, this contradicts Condition 2 of the theorem.

Thus, *Sys* is deadlock-free since if the conditions of the theorem hold and a reachable deadlock exists, we can find a contradiction as shown above: If there is no interaction on the path to the deadlock where only components in D or at least one cycle component in D participate in, then there is a contradiction to the first condition. Otherwise, if there is such an interaction, we have a contradiction to the second condition. We have to address one more item from the very beginning, viz. the presence of closed interactions. In the beginning of the proof, we stated that in case of the presence of a reachable, deadlocked global state $s \in S$ we can find a sequence σ starting in a global initial state $s^0 \in S^0$ with $s^0 \xrightarrow{\alpha_0} s^1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_m} s$ for a fixed $m \in \mathbb{N}$. In the presence of closed interactions some of the α_l, $0 \leq l \leq m$, of this sequence may correspond to τ. However, we know that in this case an interaction exists that we can use for our reasoning instead, i.e., if we have $s^l \xrightarrow{\tau} s^{l+1}$ on the sequence σ where $0 \leq l < m$, we know from Definition 2.6 that an interaction $\alpha_l \in Int$ exists such that for all $i \in Comp$ either $i(\alpha_l) = \{a_i\}$ for an $a_i \in A_i$ and $s_i \xrightarrow{a_i}_i t_i$ holds or we have $i(\alpha_l) = \emptyset$ and $s_i = t_i$ where s_i correspond to i's local part of the global state s^l and t_i to its local part of s^{l+1}. Thus, closed interactions do not cause a problem with our reasoning above. ■

F.6 Proofs from Chapter 7

Proof of Lemma 7.7: The claim directly follows from the properties of branching bisimilarity with explicit divergence which preserves deadlock-freedom. A reachable deadlock in $Sys[\{i\}]$ implies a reachable state without successors in $[\![i]\!]$ and also in $[\![i{:}p]\!]$ because of the port conformance. Thus, this state also occurs in $[\![\{i{:}p\}]\!]$. The other direction follows with an analogous argument. ■

Proof of Theorem 7.8: Assume that the assumptions of the theorem are satisfied and that Sys is not deadlock-free. We now successively consider subsystems of cooperating components of increasing size in an induction like manner, i.e., assume that there is a set $C \subseteq Comp$ of components such that $Sys[C]$ is deadlock-free. Such a set of size at least one must exists since each $Sys[\{i\}]$ with $i \in Comp$ is deadlock-free because of Lemma 7.7. Now, pick a component $j \notin C$ that cooperates with a component in C and consider $C' = C \cup \{j\}$. Assume that $Sys[C']$ is not deadlock-free, although a component $i \in C$ and a port $i{:}p \in \text{ports}(i)$ exist such that $i{:}p$ is connected to a port $j{:}q \in \text{ports}(j)$ and the port behavior $[\![\{i{:}p, j{:}q\}]\!]$ is τ- and deadlock-free—which follows from the assumptions. Note that the port behavior $[\![\{i{:}p, j{:}q\}]\!]$ cannot contain τ-transitions since the port protocols are τ-free and a τ-transition in a port behavior always corresponds to a τ-transition in one of the involved port protocols (cf. Definition 7.4). Since interaction system $Sys[C']$ is not deadlock-free, there is a reachable global state $s_{C'} \in S_{C'}$ that has no successor. Since the corresponding state $s_C \in S_C$ in the system without j is deadlock-free, there must be an action $a_i \in A_i$ and states $s_i, t_i \in S_i$ with s_i being i's local part in s_C and $s_i \xrightarrow{a_i}_i t_i$. But, the corresponding interaction α with $i(\alpha) = i{:}p(\alpha) = \{a_i\}$ is not available in $s_{C'}$ anymore—due to the deadlock, i.e., there must be an action $a_j \in A_j$ with $j(\alpha) = j{:}q(\alpha) = \{a_j\}$ that is not enabled in j's local part s_j in $s_{C'}$. Consider the behavior of the subsystem $Sys[\{i, j\}]$ and the state (s_i, s_j) of $[\![Sys[\{i, j\}]]\!]$, which is reachable from an initial state in $[\![Sys[\{i, j\}]]\!]$ since the deadlocked global state $s_{C'}$ is reachable in $[\![Sys[C']]\!]$. Now, a state $(s_{i:p}, s_{j:q}) \in S_{\{i:p,j:q\}}$ with $(s_i, s_{i:p}) \in \mathcal{R}_{\approx_b^\Delta}$ and $(s_j, s_{j:q}) \in \mathcal{R}'_{\approx_b^\Delta}$ is also reachable in the port behavior $[\![\{i{:}p, j{:}q\}]\!]$ because of the port conformance where $\mathcal{R}_{\approx_b^\Delta}$ and $\mathcal{R}'_{\approx_b^\Delta}$ denote the corresponding relations that establish the branching bisimilarity with explicit divergence. But then, $Sys[C']$ cannot be deadlocked since at least one $\beta \in Int_{\{i:p,j:q\}}$—and thus $\beta \in Int[C']$—is enabled in $(s_{i:p}, s_{j:q})$ because of the port behavior's deadlock-freedom, and this β can neither be blocked by i—because i's cooperation with the other components in C is deadlock-free—nor by j—because the only cooperation partner of j is i—in $[\![Sys[C']]\!]$ because otherwise the port behavior $[\![\{i{:}p, j{:}q\}]\!]$ contains a τ-transition or Sys does not have a tree-like protocol architecture. Thus, applying this induction until all components are covered, i.e., $C' = Comp$, yields the deadlock-freedom of interaction system Sys. ∎

Bibliography

[1] Willibrordus Martinus Pancratius van der Aalst, Kees Max van Hee, and Robert Arie van der Toorn. Component-Based Software Architectures: A Framework Based on Inheritance of Behavior. *Science of Computer Programming (SCP)*, 42(2-3):129–171, 2002.

[2] Parosh Aziz Abdulla, Giorgio Delzanno, and Ahmed Rezine. Automatic Verification of Directory-Based Consistency Protocols. In Olivier Bournez and Igor Potapov, editors, *Proceedings of the 3rd International Workshop on Reachability Problems (RP 2009)*, volume 5797 of *Lecture Notes in Computer Science*, pages 36–50. Springer-Verlag, Berlin, Germany, 2009.

[3] Gregory Dominic Abowd, Robert John Allen, and David Barnard Garlan. Using Style to Understand Descriptions of Software Architecture. In *Proceedings of the 1st ACM SIGSOFT Symposium on Foundations of Software Engineering (SIGSOFT 1993)*, pages 9–20. ACM, New York, NY, USA, 1993.

[4] Jean-Raymond Abrial, Stephen A. Schuman, and Bertrand Meyer. Specification Language. In R. M. McKeag and A. M. Macnaughten, editors, *On the Construction of Programs*, pages 343–410. Cambridge University Press, Cambridge, UK, 1980.

[5] Franz Achermann and Oscar Nierstrasz. A Calculus for Reasoning about Software Composition. *Theoretical Computer Science (TCS)*, 331 (2-3):367–396, 2005.

[6] Alessandro Aldini and Marco Bernardo. On the Usability of Process Algebra: An Architectural View. *Theoretical Computer Science (TCS)*, 335 (2-3):281–329, 2005.

[7] Robert John Allen and David Barnard Garlan. A Formal Approach to Software Architectures. In Jan van Leeuwen, editor, *Proceedings of the 12th IFIP World Computer Congress on Information Processing (IFIP 1992)*, pages 134–141. North-Holland, Amsterdam, The Netherlands, 1992.

[8] Robert John Allen and David Barnard Garlan. A Formal Basis for

Architectural Connection. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 6(3):213–249, 1997.

[9] Bowen Alpern and Fred Barry Schneider. Defining Liveness. *Information Processing Letters (IPL)*, 21(4):181–185, 1985.

[10] Bowen Alpern and Fred Barry Schneider. Recognizing Safety and Liveness. *Distributed Computing*, 2(3):117–126, 1987.

[11] Nasreddine Aoumeur and Gunter Saake. A Component-Based Petri Net Model for Specifying and Validating Cooperative Information Systems. *Data & Knowledge Engineering (DKE)*, 42(2):143–187, 2002.

[12] Farhad Arbab. Abstract Behavior Types: A Foundation Model for Components and Their Composition. In Frank Sipke de Boer, Marcello Maria Bonsangue, Susanne Graf, and Willem-Paul de Roever, editors, *Proceedings of the 1st International Symposium on Formal Methods for Components and Objects (FMCO 2002)*, volume 2852 of *Lecture Notes in Computer Science*, pages 33–70. Springer-Verlag, Berlin, Germany, 2003.

[13] Farhad Arbab. Reo: A Channel-Based Coordination Model for Component Composition. *Mathematical Structures in Computer Science*, 14(3): 329–366, 2004.

[14] André Arnold. *Finite Transition Systems: Semantics of Communicating Systems*. Prentice Hall, Hertfordshire, UK, 1994.

[15] Edward Anthony Ashcroft. Proving Assertions about Parallel Programs. *Journal of Computer and System Sciences (JCSS)*, 10(1):110–135, 1975.

[16] Paul Camille Attie and Hana Chockler. Efficiently Verifiable Conditions for Deadlock-Freedom of Large Concurrent Programs. In Radhia Cousot, editor, *Proceedings of the 6th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2005)*, volume 3385 of *Lecture Notes in Computer Science*, pages 465–481. Springer-Verlag, Berlin, Germany, 2005.

[17] Ralph-Johan Back and Reino Elias Mikael Kurki-Suonio. Distributed Cooperation with Action Systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 10(4):513–554, 1988.

[18] John Warner Backus. The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference. In *Proceedings of the 1st International Conference on Information Processing (IFIP 1959)*, pages 125–131. UNESCO, Paris, France, 1959.

[19] Eric Badouel, Albert Benveniste, Marius Dorel Bozga, Benoît Caillaud,

Olivier Constant, Bernhard Josko, Qin Ma, Roberto Passerone, and Mark Skipper. SPEEDS Metamodel Syntax and Draft Semantics, 2007. Deliverable D2.1c.

[20] Josephus Cornelis Maria Baeten. A Brief History of Process Algebra. *Theoretical Computer Science (TCS)*, 335(2-3):131–146, 2005.

[21] Josephus Cornelis Maria Baeten. Formal Methods. Lecture given at the IPA Basic Course on Formal Methods. Eindhoven University of Technology, 16 January 2006. URL: `http://www.win.tue.nl/ipa/archive/fmbasiccourse2006/FM05b.ppt`.

[22] Josephus Cornelis Maria Baeten and Robert Jan van Glabbeek. Another Look at Abstraction in Process Algebra (Extended Abstract). In Thomas Ottmann, editor, *Proceedings of the 14th Colloquium on Automata, Languages and Programming (ICALP 1987)*, volume 267 of *Lecture Notes in Computer Science*, pages 84–94. Springer-Verlag, Berlin, Germany, 1987.

[23] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, Cambridge, MA, USA, 2008.

[24] Christel Baier, Tobias Blechmann, Joachim Klein, and Sascha Klüppelholz. A Uniform Framework for Modeling and Verifying Components and Connectors. In John Field and Vasco Thudichum Vasconcelos, editors, *Proceedings of the 11th International Conference on Coordination Models and Languages (COORDINATION 2009)*, volume 5521 of *Lecture Notes in Computer Science*, pages 247–267. Springer-Verlag, Berlin, Germany, 2009.

[25] Brenda Sue Baker. Approximation Algorithms for NP-Complete Problems on Planar Graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994.

[26] Eric Barboni and Rémi Bastide. Software Components: A Formal Semantics Based on Coloured Petri Nets. In *Proceedings of the 2nd International Workshop on Formal Aspects of Component Software (FACS 2005)*, volume 160 of *Electronic Notes in Theoretical Computer Science*, pages 57–73. Elsevier B.V., Amsterdam, The Netherlands, 2006.

[27] Leonor Barroca, Jon Hall, and Patrick Hall. An Introduction and History of Software Architectures, Components, and Reuse. In *Software Architectures: Advances and Applications*, pages 1–11. Springer-Verlag, London, UK, 2000.

[28] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*, 2nd edition. Addison-Wesley Professional, Boston, MA, USA, 2003.

[29] Twan Basten. Branching Bisimilarity Is an Equivalence Indeed! *Information Processing Letters (IPL)*, 58(3):141–147, 1996.

[30] Ananda Shankar Basu, Marius Dorel Bozga, and Joseph Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *Proceedings of the 4th International Conference on Software Engineering and Formal Methods (SEFM 2006)*, pages 3–12. IEEE Computer Society, Los Alamitos, CA, USA, 2006.

[31] Hubert Baumeister, Florian Hacklinger, Rolf Hennicker, Alexander Knapp, and Martin Wirsing. A Component Model for Architectural Programming. In *Proceedings of the 2nd International Workshop on Formal Aspects of Component Software (FACS 2005)*, volume 160 of *Electronic Notes in Theoretical Computer Science*, pages 75–96. Elsevier B.V., Amsterdam, The Netherlands, 2006.

[32] Oliver Becker. *Implementierung von Algorithmen zur Deadlock- und Fortschritt-Erkennung in komponentenbasierten Systemen*. Diploma thesis, University of Dortmund, 2012.

[33] Mordechai Ben-Ari, Zohar Manna, and Amir Pnueli. The Temporal Logic of Branching Time. In *Proceedings of the 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1981)*, pages 164–176. ACM, New York, NY, USA, 1981.

[34] Saddek Bensalem, Marius Dorel Bozga, Thanh-Hung Nguyen, and Joseph Sifakis. Compositional Verification for Component-Based Systems and Application. In Sung Deok Cha, Jin-Young Choi, Moonzoo Kim, Insup Lee, and Mahesh Viswanathan, editors, *Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis (ATVA 2008)*, volume 5311 of *Lecture Notes in Computer Science*, pages 64–79. Springer-Verlag, Berlin, Germany, 2008.

[35] Saddek Bensalem, Marius Dorel Bozga, Thanh-Hung Nguyen, and Joseph Sifakis. D-Finder: A Tool for Compositional Deadlock Detection and Verification. In Ahmed Bouajjani and Oded Maler, editors, *Proceedings of the 21st International Conference on Computer Aided Verification (CAV 2009)*, volume 5643 of *Lecture Notes in Computer Science*, pages 614–619. Springer-Verlag, Berlin, Germany, 2009.

[36] Saddek Bensalem, Marius Dorel Bozga, Thanh-Hung Nguyen, and Joseph Sifakis. Compositional Verification for Component-Based Systems and Application. *IET Software*, 4(3):181–193, 2010.

[37] Johannes Aldert Bergstra and Jan Willem Klop. Algebra of Communi-

cating Processes with Abstraction. *Theoretical Computer Science (TCS)*, 37:77–121, 1985.

[38] Johannes Aldert Bergstra, Jan Willem Klop, and Ernst-Rüdiger Olderog. Failures without Chaos: A Process Semantics for Fair Abstraction. In Martin Wirsing, editor, *Proceedings of the IFIP TC2/WG2.2 Working Conference on Formal Description of Programming Concepts–III*, pages 77–101. Elsevier B.V., Amsterdam, The Netherlands, 1987.

[39] Marco Bernardo, Paolo Ciancarini, and Lorenzo Donatiello. Architecting Families of Software Systems with Process Algebras. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(4):386–426, 2002.

[40] Marc Bezem and Jan Friso Groote. Invariants in Process Algebra with Data. In Bengt Jonsson and Joachim Parrow, editors, *Proceedings of the 5th International Conference on Concurrency Theory (CONCUR 1994)*, volume 836 of *Lecture Notes in Computer Science*, pages 401–416. Springer-Verlag, Berlin, Germany, 1994.

[41] Armin Biere, Cyrille Artho, and Viktor Schuppan. Liveness Checking as Safety Checking. In *Proceedings of the 7th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2002)*, volume 66(2) of *Electronic Notes in Theoretical Computer Science*, pages 160–177. Elsevier B.V., Amsterdam, The Netherlands, 2002.

[42] Dines Bjørner and Clifford B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *Lecture Notes in Computer Science*, Berlin, Germany, 1978. Springer-Verlag.

[43] Barry William Boehm. *Software Engineering Economics*, 1st edition. Prentice Hall, Upper Saddle River, NJ, USA, 1981.

[44] Borzoo Bonakdarpour, Marius Dorel Bozga, and Gregor Gössler. A Theory of Fault Recovery for Component-Based Models. In *Proceedings of the 30th IEEE Symposium on Reliable Distributed Systems (SRDS 2011)*, pages 265–270. IEEE Computer Society, Los Alamitos, CA, USA, 2011.

[45] Marius Dorel Bozga, Olivier Constant, Bernhard Josko, Qin Ma, and Mark Skipper. SPEEDS Metamodel Syntax and Static Semantics, 2007. Deliverable D2.1b.

[46] Marius Dorel Bozga, Vassiliki Sfyrla, and Joseph Sifakis. Modeling Synchronous Systems in BIP. In *Proceedings of the 9th International Conference on Embedded Software (EMSOFT 2009)*, pages 77–86. ACM, New York, NY, USA, 2009.

[47] Luboš Brim, Ivana Černá, Pavlína Vařeková, and Barbora Zim-

merova. Component-Interaction Automata as a Verification-Oriented Component-Based System Specification. In *Proceedings of the 4th Workshop on Specification and Verification of Component-Based Systems (SAVCBS 2005)*. ACM, New York, NY, USA, 2005.

[48] Guy H. Broadfoot. ASD Case Notes: Costs and Benefits of Applying Formal Methods to Industrial Control Software. In John Fitzgerald, Ian James Hayes, and Andrzej Tarlecki, editors, *Proceedings of the 13th International Symposium on Formal Methods (FM 2005)*, volume 3582 of *Lecture Notes in Computer Science*, pages 548–551. Springer-Verlag, Berlin, Germany, 2005.

[49] Stephen D. Brookes and Andrew William Roscoe. Deadlock Analysis in Networks of Communicating Processes. *Distributed Computing*, 4(4): 209–230, 1991.

[50] Stephen D. Brookes and William Chesley Rounds. Behavioural Equivalence Relations Induced by Programming Logics. In Josep Díaz, editor, *Proceedings of the 10th Colloquium on Automata, Languages and Programming (ICALP 1983)*, volume 154 of *Lecture Notes in Computer Science*, pages 97–108. Springer-Verlag, Berlin, Germany, 1983.

[51] Stephen D. Brookes, Charles Antony Richard Hoare, and Andrew William Roscoe. A Theory of Communicating Sequential Processes. *Journal of the ACM (JACM)*, 31(3):560–599, 1984.

[52] Frederick Phillips Brooks and Kenneth Eugene Iverson. *Automatic Data Processing, System 360 Edition*. Wiley, New York, NY, USA, 1969.

[53] Michael C. Browne, Edmund Melson Clarke, and Orna Grumberg. Characterizing Finite Kripke Structures in Propositional Temporal Logic. *Theoretical Computer Science (TCS)*, 59(1-2):115–131, 1988.

[54] Hans de Bruin. A Grey-Box Approach to Component Composition. In Krzysztof Czarnecki and Ulrich W. Eisenecker, editors, *Proceedings of the 1st International Symposium on Generative and Component-Based Software Engineering (GCSE 1999)*, volume 1799 of *Lecture Notes in Computer Science*, pages 195–209. Springer-Verlag, Berlin, Germany, 2000.

[55] Randal Everitt Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers (TC)*, 35(8):677–691, 1986.

[56] Julius Richard Büchi. On a Decision Method in Restricted Second Order Arithmetic. In Ernest Nagel, Patrick Suppes, and Alfred Tarski, editors, *Proceedings of the 1st International Congress on Logic, Methodology and*

Philosophy of Science (LMPS 1960), pages 1–11. Stanford University Press, Stanford, CA, USA, 1962.

[57] Jerry Robert Burch, Edmund Melson Clarke, Kenneth Lauchlin McMillan, David L. Dill, and Lucius James Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science (LICS 1990)*, pages 428–439. IEEE Computer Society, Los Alamitos, CA, USA, 1990.

[58] Tomáš Bureš, Petr Hnětynka, and František Plášil. SOFA 2.0: Balancing Advanced Features in a Hierarchical Component Model. In Doo-Kwon Baik, David Primeaux, Naohiro Ishii, and Roger Lee, editors, *Proceedings of the 4th International Conference on Software Engineering Research, Management and Applications (SERA 2006)*, pages 40–48. IEEE Computer Society, Los Alamitos, CA, USA, 2006.

[59] Arthur Thomas Charlesworth. The Multiway Rendezvous. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 9(3):350–366, 1987.

[60] Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity Results for 1-Safe Nets. In Rudrapatna Kallikote Shyamasundar, editor, *Proceedings of the 13th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1993)*, volume 761 of *Lecture Notes in Computer Science*, pages 326–337. Springer-Verlag, Berlin, Germany, 1993.

[61] Shing Chi Cheung and Jeff Kramer. Context Constraints for Compositional Reachability Analysis. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(4):334–377, 1996.

[62] Shing Chi Cheung and Jeff Kramer. Checking Safety Properties Using Compositional Reachability Analysis. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 8(1):49–78, 1999.

[63] Shing Chi Cheung, Dimitra Giannakopoulou, and Jeff Kramer. Verification of Liveness Properties Using Compositional Reachability Analysis. *ACM SIGSOFT Software Engineering Notes (SEN)*, 22(6):227–243, 1997.

[64] Avram Noam Chomsky. Three Models for the Description of Language. *IRE Transactions on Information Theory*, 2:113–124, 1956.

[65] Alonzo Church. An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics*, 58(2):345–363, 1936.

[66] Edmund Melson Clarke and Ernest Allen Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Proceedings of the Workshop on Logic of Programs (LOP 1981)*, volume

131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, Berlin, Germany, 1982.

[67] Edmund Melson Clarke and Jeannette Marie Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4): 626–643, 1996.

[68] Edmund Melson Clarke, Ernest Allen Emerson, and Aravinda Prasad Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[69] Edmund Melson Clarke, David Esley Long, and Kenneth Lauchlin McMillan. Compositional Model Checking. In *Proceedings of the 4th Annual Symposium on Logic in Computer Science (LICS 1989)*, pages 353–362. IEEE, Piscataway, NJ, USA, 1989.

[70] Edmund Melson Clarke, Thomas Filkorn, and Somesh Jha. Exploiting Symmetry in Temporal Logic Model Checking. In Costas Courcoubetis, editor, *Proceedings of the 5th International Workshop on Computer Aided Verification (CAV 1993)*, volume 697 of *Lecture Notes in Computer Science*, pages 450–462. Springer-Verlag, Berlin, Germany, 1993.

[71] Edmund Melson Clarke, Orna Grumberg, and David Esley Long. Model Checking and Abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.

[72] Edmund Melson Clarke, Orna Grumberg, and Doron Angel Peled. *Model Checking*. The MIT Press, Cambridge, MA, USA, 2000.

[73] Walter Rance Cleaveland and Scott Allen Smolka. Strategic Directions in Concurrency Research. *ACM Computing Surveys*, 28(4):607–625, 1996.

[74] Stephen Arthur Cook. The Complexity of Theorem-Proving Procedures. In Michael Alexander Harrison, Ranan B. Banerji, and Jeffrey David Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971)*, pages 151–158. ACM, New York, NY, USA, 1971.

[75] Don Coppersmith and Shmuel Winograd. Matrix Multiplication via Arithmetic Progressions. In Alfred Vaino Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC 1987)*, pages 1–6. ACM, New York, NY, USA, 1987.

[76] Thomas H. Cormen, Charles Eric Leiserson, Ronald Linn Rivest, and Clifford Stein. *Introduction to Algorithms*, 2nd edition. The MIT Press and McGraw-Hill Book Company, Cambridge, MA, USA, 2001.

[77] Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 1977)*, pages 238–252. ACM, New York, NY, USA, 1977.

[78] Brad J. Cox. *Object-Oriented Programming: An Evolutionary Approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[79] Leandro Dias da Silva and Angelo Perkusich. Composition of Software Artifacts Modelled Using Colored Petri Nets. *Science of Computer Programming (SCP)*, 56(1-2):171–189, 2005.

[80] Luca de Alfaro and Thomas Anton Henzinger. Interface Automata. *ACM SIGSOFT Software Engineering Notes (SEN)*, 26(5):109–120, 2001.

[81] Luca de Alfaro and Thomas Anton Henzinger. Interface Theories for Component-Based Design. In Thomas Anton Henzinger and Christoph Meyer Kirsch, editors, *Proceedings of the 1st International Workshop on Embedded Software (EMSOFT 2001)*, volume 2211 of *Lecture Notes in Computer Science*, pages 148–165. Springer-Verlag, Berlin, Germany, 2001.

[82] Luca de Alfaro and Thomas Anton Henzinger. Interface-Based Design. In Manfred Broy, Johannes Grünbauer, David Harel, and Charles Antony Richard Hoare, editors, *Engineering Theories of Software Intensive Systems*, volume 195 of *NATO Science Series II*, pages 83–104. Springer-Verlag, Berlin, Germany, 2005.

[83] Rocco De Nicola and Frits Willem Vaandrager. Action versus State Based Logics for Transition Systems. In Irène Guessarian, editor, *Semantics of Systems of Concurrent Processes*, volume 469 of *Lecture Notes in Computer Science*, pages 407–419. Springer-Verlag, Berlin, Germany, 1990.

[84] Rocco De Nicola and Frits Willem Vaandrager. Three Logics for Branching Bisimulation. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science (LICS 1990)*, pages 118–129. IEEE Computer Society, Los Alamitos, CA, USA, 1990. Also available as CS-R9012, CWI Report, Center for Mathematics and Computer Science, Amsterdam, The Netherlands, 1990.

[85] Rocco De Nicola and Frits Willem Vaandrager. Three Logics for Branching Bisimulation. *Journal of the ACM (JACM)*, 42(2):458–487, 1995.

[86] Rocco De Nicola, Ugo Montanari, and Frits Willem Vaandrager. Back and Forth Bisimulations. In Josephus Cornelis Maria Baeten and

Jan Willem Klop, editors, *Proceedings of the 1st International Conference on Concurrency Theory (CONCUR 1990)*, volume 458 of *Lecture Notes in Computer Science*, pages 152–165. Springer-Verlag, Berlin, Germany, 1990.

[87] Frank Dederichs and Rainer Weber. Safety and Liveness from a Methodological Point of View. *Information Processing Letters (IPL)*, 36(1):25–30, 1990.

[88] Richard Allan DeMillo, Richard Jay Lipton, and Alan Jay Perlis. Social Processes and Proofs of Theorems and Programs. *Communications of the ACM (CACM)*, 22(5):271–280, 1979.

[89] Reinhard Diestel. *Graph Theory*, 4th edition, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, Germany, 2010.

[90] Edsger Wybe Dijkstra. Cooperating Sequential Processes. In F. Genuys, editor, *Programming Languages: NATO Advanced Study Institute*, pages 43–112. Academic Press, New York, NY, USA, 1968. Originally appeared as Technical Report EWD-123, Eindhoven University of Technology, Eindhoven, The Netherlands, 1965.

[91] Edsger Wybe Dijkstra. The Structure of the "THE"-Multiprogramming System. *Communications of the ACM (CACM)*, 11(5):341–346, 1968.

[92] Edsger Wybe Dijkstra. On the Role of Scientific Thought. In *Selected Writings on Computing: A Personal Perspective*, pages 60–66. Springer-Verlag, Berlin, Germany, 1982.

[93] Agostino Dovier, Carla Piazza, and Alberto Policriti. An Efficient Algorithm for Computing Bisimulation Equivalence. *Theoretical Computer Science (TCS)*, 311(1-3):221–256, 2004.

[94] Laurent Doyen, Thomas Anton Henzinger, Barbara Jobstmann, and Tatjana Petrov. Interface Theories with Component Reuse. In *Proceedings of the 8th International Conference on Embedded Software (EMSOFT 2008)*, pages 79–88. ACM, New York, NY, USA, 2008.

[95] Ernest Allen Emerson and Joseph Yehuda Halpern. "Sometimes" and "Not Never" Revisited: On Branching versus Linear Time Temporal Logic. *Journal of the ACM (JACM)*, 33(1):151–178, 1986.

[96] Ernest Allen Emerson and Chin-Laung Lei. Model Checking under Generalized Fairness Constraints. Technical Report TR-84-20, University of Texas at Austin, Austin, TX, USA, 1984.

[97] Ernest Allen Emerson and Chin-Laung Lei. Modalities for Model Check-

ing: Branching Time Logic Strikes Back. *Science of Computer Programming (SCP)*, 8(3):275–306, 1987.

[98] Ernest Allen Emerson and Aravinda Prasad Sistla. Symmetry and Model Checking. In Costas Courcoubetis, editor, *Proceedings of the 5th International Workshop on Computer Aided Verification (CAV 1993)*, volume 697 of *Lecture Notes in Computer Science*, pages 463–478. Springer-Verlag, Berlin, Germany, 1993.

[99] Shimon Even, Alon Itai, and Adi Shamir. On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM Journal on Computing (SICOMP)*, 5(4):691–703, 1976.

[100] Robert W Floyd. Algorithm 97: Shortest Path. *Communications of the ACM (CACM)*, 5(6):345, 1962.

[101] Robert W Floyd. Assigning Meaning to Programs. In Jacob Theodore Schwartz, editor, *Mathematical Aspects of Computer Science, Proceedings of Symposia in Applied Mathematics*, volume 19, pages 19–32. American Mathematical Society, Providence, RI, USA, 1967.

[102] Willem Jan Fokkink, Jun Pang, and Jan Cornelius van de Pol. Cones and Foci: A Mechanical Framework for Protocol Verification. *Formal Methods in System Design (FMSD)*, 29(1):1–31, 2006.

[103] Lester Randolph Ford and Delbert Ray Fulkerson. A Simple Algorithm for Finding Maximal Network Flows and an Application to the Hitchcock Problem. *Canadian Journal of Mathematics*, 9:210–218, 1957.

[104] Lester Randolph Ford and Delbert Ray Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, USA, 1962.

[105] Felix Christoph Freiling, Christian Lambertz, and Mila Emilia Majster-Cederbaum. Easy Consensus Algorithms for the Crash-Recovery Model. In Gadi Taubenfeld, editor, *Proceedings of the 22nd International Symposium on Distributed Computing (DISC 2008)*, volume 5218 of *Lecture Notes in Computer Science*, pages 507–508. Springer-Verlag, Berlin, Germany, 2008.

[106] Felix Christoph Freiling, Christian Lambertz, and Mila Emilia Majster-Cederbaum. Modular Consensus Algorithms for the Crash-Recovery Model. In *Proceedings of the 10th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2009)*, pages 287–292. IEEE Computer Society, Los Alamitos, CA, USA, 2009. Presented at the 2nd International Workshop on Reliability, Availability, and Security (WRAS 2009).

[107] Erich Gamma, Richard Helm, Ralph Edward Johnson, and John Matthew Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, MA, USA, 1994.

[108] Michael Randolph Garey and David Stifler Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, USA, 1979.

[109] David Barnard Garlan. Formal Approaches to Software Architecture. In David Alex Lamb, editor, *Proceedings of the 1993 ICSE Workshop on Studies of Software Design*, volume 1078 of *Lecture Notes in Computer Science*, pages 64–76. Springer-Verlag, Berlin, Germany, 1996.

[110] David Barnard Garlan and Mary Margaret Shaw. An Introduction to Software Architecture. In Vincenzo Ambriola and Genoveffa Tortora, editors, *Advances in Software Engineering and Knowledge Engineering, Volume I*, pages 1–39. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1993.

[111] Dimitra Giannakopoulou and Corina S. Păsăreanu. Interface Generation and Compositional Verification in JavaPathfinder. In Marsha Chechik and Martin Wirsing, editors, *Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering (FASE 2009)*, volume 5503 of *Lecture Notes in Computer Science*, pages 94–108. Springer-Verlag, Berlin, Germany, 2009.

[112] Robert Jan van Glabbeek. The Linear Time – Branching Time Spectrum (Extended Abstract). In Josephus Cornelis Maria Baeten and Jan Willem Klop, editors, *Proceedings of the 1st International Conference on Concurrency Theory (CONCUR 1990)*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer-Verlag, Berlin, Germany, 1990.

[113] Robert Jan van Glabbeek. The Linear Time – Branching Time Spectrum II. In Eike Best, editor, *Proceedings of the 4th International Conference on Concurrency Theory (CONCUR 1993)*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer-Verlag, Berlin, Germany, 1993.

[114] Robert Jan van Glabbeek and Willem Pieter Weijland. Branching Time and Abstraction in Bisimulation Semantics (Extended Abstract). In Gerhard X Ritter, editor, *Proceedings of the 11th IFIP World Computer Congress on Information Processing (IFIP 1989)*, pages 613–618. North-Holland, Amsterdam, The Netherlands, 1989.

[115] Robert Jan van Glabbeek and Willem Pieter Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM (JACM)*, 43(3):555–600, 1996.

[116] Robert Jan van Glabbeek, Sebastiaan Pascal Luttik, and Nikola Trčka. Branching Bisimilarity with Explicit Divergence. *Fundamenta Informaticae*, 93(4):371–392, 2009.

[117] Robert Jan van Glabbeek, Sebastiaan Pascal Luttik, and Nikola Trčka. Computation Tree Logic with Deadlock Detection. *Logical Methods in Computer Science*, 5(4:5):1–24, 2009.

[118] Patrice Godefroid. Using Partial Orders to Improve Automatic Verification Methods. In Edmund Melson Clarke and Robert Paul Kurshan, editors, *Proceedings of the 2nd International Workshop on Computer Aided Verification (CAV 1990)*, volume 531 of *Lecture Notes in Computer Science*, pages 176–185. Springer-Verlag, Berlin, Germany, 1991.

[119] Patrice Godefroid and Pierre Wolper. Using Partial Orders for the Efficient Verification of Deadlock Freedom and Safety Properties. In Kim Guldstrand Larsen and Arne Skou, editors, *Proceedings of the 3rd International Workshop on Computer Aided Verification (CAV 1991)*, volume 575 of *Lecture Notes in Computer Science*, pages 332–342. Springer-Verlag, Berlin, Germany, 1992.

[120] Gregor Gössler. Component-Based Design of Heterogeneous Reactive Systems in Prometheus. Technical Report 6057, INRIA, 2006.

[121] Gregor Gössler and Joseph Sifakis. Composition for Component-Based Modeling. In Frank Sipke de Boer, Marcello Maria Bonsangue, Susanne Graf, and Willem-Paul de Roever, editors, *Proceedings of the 1st International Symposium on Formal Methods for Components and Objects (FMCO 2002)*, volume 2852 of *Lecture Notes in Computer Science*, pages 443–466. Springer-Verlag, Berlin, Germany, 2003.

[122] Gregor Gössler and Joseph Sifakis. Component-Based Construction of Deadlock-Free Systems. In Paritosh Kulin Pandya and Jaikumar Radhakrishnan, editors, *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2003)*, volume 2914 of *Lecture Notes in Computer Science*, pages 420–433. Springer-Verlag, Berlin, Germany, 2003.

[123] Gregor Gössler and Joseph Sifakis. Composition for Component-Based Modeling. *Science of Computer Programming (SCP)*, 55(1-3):161–183, 2005.

[124] Gregor Gössler, Susanne Graf, Mila Emilia Majster-Cederbaum, Moritz Martens, and Joseph Sifakis. Ensuring Properties of Interaction Systems. In Thomas William Reps, Mooly Sagiv, and Jörg Bauer, editors, *Program Analysis and Compilation, Theory and Practice, Essays Dedicated to Reinhard Wilhelm on the Occasion of His 60th Birthday*, volume 4444 of *Lecture Notes*

in Computer Science, pages 201–224. Springer-Verlag, Berlin, Germany, 2006.

[125] Susanne Graf and Bernhard Steffen. Compositional Minimization of Finite State Systems. In Edmund Melson Clarke and Robert Paul Kurshan, editors, *Proceedings of the 2nd International Workshop on Computer Aided Verification (CAV 1990)*, volume 531 of *Lecture Notes in Computer Science*, pages 186–196. Springer-Verlag, Berlin, Germany, 1991.

[126] Jan Friso Groote and Faron Moller. Verification of Parallel Systems via Decomposition. In Walter Rance Cleaveland, editor, *Proceedings of the 3rd International Conference on Concurrency Theory (CONCUR 1992)*, volume 630 of *Lecture Notes in Computer Science*, pages 62–76. Springer-Verlag, Berlin, Germany, 1992.

[127] Jan Friso Groote and Jan Cornelius van de Pol. State Space Reduction Using Partial τ-Confluence. In Mogens Nielsen and Branislav Rovan, editors, *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science (MFCS 2000)*, volume 1893 of *Lecture Notes in Computer Science*, pages 383–393. Springer-Verlag, Berlin, Germany, 2000.

[128] Jan Friso Groote and Martin Paul Alexander Sellink. Confluence for Process Verification. *Theoretical Computer Science (TCS)*, 170(1-2):47–81, 1996.

[129] Jan Friso Groote and Jan Springintveld. Focus Points and Convergent Process Operators: A Proof Strategy for Protocol Verification. *The Journal of Logic and Algebraic Programming (JLAP)*, 49(1-2):31–60, 2001.

[130] Jan Friso Groote and Frits Willem Vaandrager. An Efficient Algorithm for Branching Bisimulation and Stuttering Equivalence. In Michael Stewart Paterson, editor, *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP 1990)*, volume 443 of *Lecture Notes in Computer Science*, pages 626–638. Springer-Verlag, Berlin, Germany, 1990.

[131] Jan Friso Groote, Adrianus Hubertus Johannes Mathijssen, Michel Adriaan Reniers, Yaroslav Sergiyovych Usenko, and Muck Joost van Weerdenburg. The Formal Specification Language mCRL2. In Ed Brinksma, David Harel, Angelika Mader, Perdita Stevens, and Roel Wieringa, editors, *Methods for Modelling Software Systems (MMOSS)*, volume 06351 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Dagstuhl, Germany, 2007.

[132] Jan Friso Groote, Ammar Osaiweran, and Jacob Hendrikus Wesselius.

Analyzing the Effects of Formal Methods on the Development of Industrial Control Software. In *Proceedings of the 27th International Conference on Software Maintenance (ICSM 2011)*, pages 467–472. IEEE Computer Society, Los Alamitos, CA, USA, 2011.

[133] Hans Peter Gumm. Another Glance at the Alpern-Schneider Characterization of Safety and Liveness in Concurrent Executions. *Information Processing Letters (IPL)*, 47(6):291–294, 1993.

[134] George T. Heineman and William T. Councill. *Component-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[135] Matthew C. B. Hennessy and Arthur John Robin Gorell Milner. On Observing Nondeterminism and Concurrency. In Jaco Willem de Bakker and Jan van Leeuwen, editors, *Proceedings of the 7th Colloquium on Automata, Languages and Programming (ICALP 1980)*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer-Verlag, Berlin, Germany, 1980.

[136] Matthew C. B. Hennessy and Arthur John Robin Gorell Milner. Algebraic Laws for Nondeterminism and Concurrency. *Journal of the ACM (JACM)*, 32(1):137–161, 1985.

[137] Rolf Hennicker, Stephan Janisch, and Alexander Knapp. On the Observable Behaviour of Composite Components. In Carlos Canal and Corina S. Păsăreanu, editors, *Proceedings of the 5th International Workshop on Formal Aspects of Component Software (FACS 2008)*, volume 260 of *Electronic Notes in Theoretical Computer Science*, pages 125–153. Elsevier B.V., Amsterdam, The Netherlands, 2010.

[138] Alex Ho, Steven Smith, and Steven Hand. On Deadlock, Livelock, and Forward Progress. Technical Report UCAM-CL-TR-633, University of Cambridge, Computer Laboratory, 2005.

[139] Charles Antony Richard Hoare. An Axiomatic Basis for Computer Programming. *Communications of the ACM (CACM)*, 12(10):576–580, 1969.

[140] Charles Antony Richard Hoare. A Model for Communicating Sequential Processes. In R. M. McKeag and A. M. Macnaughten, editors, *On the Construction of Programs*, pages 229–254. Cambridge University Press, Cambridge, UK, 1980.

[141] Charles Antony Richard Hoare. *Communicating Sequential Processes*. Prentice Hall, Upper Saddle River, NJ, USA, 1985.

[142] Gerard Johan Holzmann. The Model Checker SPIN. *IEEE Transactions on Software Engineering (TSE)*, 23(5):279–295, 1997.

[143] John Edward Hopcroft. An $n \log n$ Algorithm for Minimizing States in a Finite Automaton. In Zvi Kohavi and Azaria Paz, editors, *Proceedings of the International Symposium on the Theory of Machines and Computations*, pages 189–196. Academic Press, New York, NY, USA, 1971.

[144] John Edward Hopcroft and Robert Endre Tarjan. Algorithm 447: Efficient Algorithms for Graph Manipulation. *Communications of the ACM (CACM)*, 16(6):372–378, 1973.

[145] John Edward Hopcroft, Rajeev Motwani, and Jeffrey David Ullman. *Introduction to Automata Theory, Languages, and Computation*, 3rd edition. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[146] Paola Inverardi and Sebastián Uchitel. Proving Deadlock Freedom in Component-Based Programming. In Heinrich Hußmann, editor, *Proceedings of the 4th International Conference on Fundamental Approaches to Software Engineering (FASE 2001)*, volume 2029 of *Lecture Notes in Computer Science*, pages 60–75. Springer-Verlag, Berlin, Germany, 2001.

[147] Chung-Wah Norris Ip and David L. Dill. Better Verification through Symmetry. In *Proceedings of the 11th IFIP WG10.2 International Conference on Computer Hardware Description Languages and their Applications (CHDL 1993)*, pages 97–111. North-Holland, Amsterdam, The Netherlands, 1993.

[148] Pavel Ježek, Jan Kofroň, and František Plášil. Model Checking of Component Behavior Specification: A Real Life Experience. In *Proceedings of the 2nd International Workshop on Formal Aspects of Component Software (FACS 2005)*, volume 160 of *Electronic Notes in Theoretical Computer Science*, pages 197–210. Elsevier B.V., Amsterdam, The Netherlands, 2006.

[149] Yuh-Jzer Joung and Scott Allen Smolka. Coordinating First-Order Multiparty Interactions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):954–985, 1994.

[150] Roope Kaivola. Compositional Model Checking for Linear-Time Temporal Logic. In Gregor von Bochmann and David Karl Probst, editors, *Proceedings of the 4th International Workshop on Computer Aided Verification (CAV 1992)*, volume 663 of *Lecture Notes in Computer Science*, pages 248–259. Springer-Verlag, Berlin, Germany, 1993.

[151] Roope Kaivola and Antti Valmari. The Weakest Compositional Semantic Equivalence Preserving Nexttime-Less Linear Temporal Logic. In Wal-

ter Rance Cleaveland, editor, *Proceedings of the 3rd International Conference on Concurrency Theory (CONCUR 1992)*, volume 630 of *Lecture Notes in Computer Science*, pages 207–221. Springer-Verlag, Berlin, Germany, 1992.

[152] Paris Christos Kanellakis and Scott Allen Smolka. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing (PODC 1983)*, pages 228–240. ACM, New York, NY, USA, 1983.

[153] David Ron Karger and Matthew Steven Levine. Random Sampling in Residual Graphs. In *Proceedings of the 34th ACM Symposium on Theory of Computing (STOC 2002)*, pages 63–66. ACM, New York, NY, USA, 2002.

[154] Richard Manning Karp. Understanding Science through the Computational Lens. *Journal of Computer Science and Technology (JCST)*, 26(4): 569–577, 2011.

[155] Matt Kaufmann, Panagiotis Manolios, and J Strother Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[156] Matt Kaufmann, Panagiotis Manolios, and J Strother Moore. *Computer-Aided Reasoning: ACL2 Case Studies*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[157] Robert Marion Keller. Formal Verification of Parallel Programs. *Communications of the ACM (CACM)*, 19(7):371–384, 1976.

[158] Thomas Kropf. *Introduction to Formal Hardware Verification: Methods and Tools for Designing Correct Circuits and Systems*, 1st edition. Springer-Verlag, Berlin, Germany, 1999.

[159] Yat-Sang Kwong. On the Absence of Livelocks in Parallel Programs. In Gilles Kahn, editor, *Proceedings of the International Symposium on Semantics of Concurrent Computation*, volume 70 of *Lecture Notes in Computer Science*, pages 172–190. Springer-Verlag, Berlin, Germany, 1979.

[160] Christian Lambertz. Exploiting Architectural Constraints and Branching Bisimulation Equivalences in Component-Based Systems. In Seyyed MohammadReza Mousavi and Emil Sekerinski, editors, *Proceedings of the Doctoral Symposium of the 2nd World Congress on Formal Methods (FM 2009-DS)*, number 0915 in Computer Science Reports, pages 1–7. Eindhoven University of Technology, Eindhoven, The Netherlands, 2009.

[161] Christian Lambertz and Mila Emilia Majster-Cederbaum. Port Protocols for Deadlock-Freedom of Component-Based Systems. In Simon Bliudze, Roberto Bruni, Davide Grohmann, and Alexandra Silva, editors, *Pro-*

ceedings of the 3rd Interaction and Concurrency Experience Workshop (ICE 2010), volume 38 of *Electronic Proceedings in Theoretical Computer Science*, pages 7–11. Open Publishing Association, 2010.

[162] Christian Lambertz and Mila Emilia Majster-Cederbaum. Analyzing Component-Based Systems on the Basis of Architectural Constraints. In Farhad Arbab and Marjan Sirjani, editors, *Proceedings of the 4th International Conference on Fundamentals of Software Engineering (FSEN 2011)*, volume 7141 of *Lecture Notes in Computer Science*, pages 64–79. Springer-Verlag, Berlin, Germany, 2012.

[163] Christian Lambertz and Mila Emilia Majster-Cederbaum. Efficient Deadlock Analysis of Component-Based Software Architectures. Manuscript submitted to Elsevier's Science of Computer Programming (SCP) journal, under revision with respect to minor modifications, 2012.

[164] Leslie Lamport. Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering (TSE)*, 3(2):125–143, 1977.

[165] Leslie Lamport. "Sometime" Is Sometimes "Not Never": On the Temporal Logic of Programs. In *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1980)*, pages 174–185. ACM, New York, NY, USA, 1980.

[166] Leslie Lamport. What Good Is Temporal Logic? In R. E. A. Mason, editor, *Proceedings of the 9th IFIP World Computer Congress on Information Processing (IFIP 1983)*, pages 657–668. North-Holland, Amsterdam, The Netherlands, 1983.

[167] Kim Guldstrand Larsen and Liu Xinxin. Compositionality through an Operational Semantics of Contexts. In Michael Stewart Paterson, editor, *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP 1990)*, volume 443 of *Lecture Notes in Computer Science*, pages 526–539. Springer-Verlag, Berlin, Germany, 1990.

[168] Thierry Lecomte, Thierry Servat, and Guilhem Pouzancre. Formal Methods in Safety-Critical Railway Systems. In *Proceedings of the 10th Brazilian Symposium on Formal Methods (SMBF 2007)*, 2007.

[169] C. Y. Lee. Representation of Switching Circuits by Binary-Decision Programs. *Bell Systems Technical Journal*, 38(4):985–999, 1959.

[170] Edward Ashford Lee and Alberto Luigi Sangiovanni-Vincentelli. Comparing Models of Computation. In *Proceedings of the 1996 International Conference on Computer-Aided Design (ICCAD 1996)*, pages 234–241. IEEE Computer Society, Los Alamitos, CA, USA, 1996.

[171] Stefan Leue, Alin Ştefănescu, and Wei Wei. A Livelock Freedom Analysis for Infinite State Asynchronous Reactive Systems. In Christel Baier and Holger Hermanns, editors, *Proceedings of the 17th International Conference on Concurrency Theory (CONCUR 2006)*, volume 4137 of *Lecture Notes in Computer Science*, pages 79–94. Springer-Verlag, Berlin, Germany, 2006.

[172] Philip M. Lewis II, Richard Edwin Stearns, and Juris Hartmanis. Memory Bounds for Recognition of Context-Free and Context-Sensitive Languages. In *Proceedings of the 6th Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1965)*, pages 191–202. IEEE Computer Society, Washington, DC, USA, 1965.

[173] Peter Liggesmeyer, Martin Rothfelder, Michael Rettelbach, and Thomas Ackermann. Qualitätssicherung Software-basierter technischer Systeme – Problembereiche und Lösungsansätze. *Informatik-Spektrum*, 21(5):249–258, 1998.

[174] Richard Jay Lipton. *The P=NP Question and Gödel's Lost Letter*, 1st edition. Springer Science+Business Media, New York, NY, USA, 2010.

[175] Jeff Magee and Jeff Kramer. *Concurrency: State Models & Java Programs*. John Wiley & Sons, Hoboken, NJ, USA, 1999.

[176] Jeff Magee, Naranker Dulay, Susan Eisenbach, and Jeff Kramer. Specifying Distributed Software Architectures. In Wilhelm Schäfer and Pere Botella, editors, *Proceedings of the 5th European Software Engineering Conference (ESEC 1995)*, volume 989 of *Lecture Notes in Computer Science*, pages 137–153. Springer-Verlag, Berlin, Germany, 1994.

[177] Jeff Magee, Jeff Kramer, and Dimitra Giannakopoulou. Software Architecture Directed Behaviour Analysis. In *Proceedings of the 9th International Workshop on Software Specification and Design (IWSSD 1998)*, pages 144–146. IEEE Computer Society, Los Alamitos, CA, USA, 1998.

[178] Mila Emilia Majster-Cederbaum and Moritz Martens. Robustness in Interaction Systems. In John Derrick and Jüri Vain, editors, *Proceedings of the 27th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2007)*, volume 4574 of *Lecture Notes in Computer Science*, pages 325–340. Springer-Verlag, Berlin, Germany, 2007.

[179] Mila Emilia Majster-Cederbaum and Moritz Martens. Compositional Analysis of Deadlock-Freedom for Tree-Like Component Architectures. In Luca de Alfaro and Jens Palsberg, editors, *Proceedings of the 8th International Conference on Embedded Software (EMSOFT 2008)*, pages 199–206. ACM, New York, NY, USA, 2008.

[180] Mila Emilia Majster-Cederbaum and Moritz Martens. Using Architectural Constraints for Deadlock-Freedom of Component Systems with Multiway Cooperation. In Wei-Ngan Chin and Shengchao Qin, editors, *Proceedings of the 3rd International Symposium on Theoretical Aspects of Software Engineering (TASE 2009)*, pages 225–232. IEEE Computer Society, Los Alamitos, CA, USA, 2009.

[181] Mila Emilia Majster-Cederbaum and Moritz Martens. Deadlock-Freedom in Component Systems with Architectural Constraints. *Formal Methods in System Design*, 41(2):129–177, 2012.

[182] Mila Emilia Majster-Cederbaum and Christoph Friedrich Minnameier. Deriving Complexity Results for Interaction Systems from 1-Safe Petri Nets. In Viliam Geffert, Juhani Karhumäki, Alberto Bertoni, Bart Preneel, Pavol Návrat, and Mária Bieliková, editors, *Proceedings of the 34th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008)*, volume 4910 of *Lecture Notes in Computer Science*, pages 352–363. Springer-Verlag, Berlin, Germany, 2008.

[183] Mila Emilia Majster-Cederbaum and Christoph Friedrich Minnameier. Everything Is PSPACE-Complete in Interaction Systems. In John Fitzgerald, Anne Haxthausen, and Husnu Yenigun, editors, *Proceedings of the 5th International Colloquium on Theoretical Aspects of Computing (ICTAC 2008)*, volume 5160 of *Lecture Notes in Computer Science*, pages 216–227. Springer-Verlag, Berlin, Germany, 2008.

[184] Mila Emilia Majster-Cederbaum and Christoph Friedrich Minnameier. Cross-Checking – Enhanced Over-Approximation of the Reachable Global State Space of Component-Based Systems. In Olivier Bournez and Igor Potapov, editors, *Proceedings of the 3rd International Workshop on Reachability Problems (RP 2009)*, volume 5797 of *Lecture Notes in Computer Science*, pages 189–202. Springer-Verlag, Berlin, Germany, 2009.

[185] Mila Emilia Majster-Cederbaum and Nils Semmelrock. Reachability in Tree-Like Component Systems Is PSPACE-Complete. In *Proceedings of the 6th International Workshop on Formal Aspects of Component Software (FACS 2009)*, volume 263 of *Electronic Notes in Theoretical Computer Science*, pages 197–210. Elsevier B.V., Amsterdam, The Netherlands, 2010.

[186] Mila Emilia Majster-Cederbaum and Nils Semmelrock. Reachability in Cooperating Systems with Architectural Constraints Is PSPACE-Complete. Manuscript, 2012.

[187] Mila Emilia Majster-Cederbaum, Moritz Martens, and Christoph Friedrich Minnameier. A Polynomial-Time Checkable

Sufficient Condition for Deadlock-Freedom of Component-Based Systems. In Jan van Leeuwen, Giuseppe Francesco Italiano, Wiebe van der Hoek, Christoph Meinel, Harald Sack, and František Plášil, editors, *Proceedings of the 33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2007)*, volume 4362 of *Lecture Notes in Computer Science*, pages 888–899. Springer-Verlag, Berlin, Germany, 2007.

[188] Mila Emilia Majster-Cederbaum, Moritz Martens, and Christoph Friedrich Minnameier. Liveness in Interaction Systems. In *Proceedings of the 4th International Workshop on Formal Aspects of Component Software (FACS 2007)*, volume 215 of *Electronic Notes in Theoretical Computer Science*, pages 57–74. Elsevier B.V., Amsterdam, The Netherlands, 2008.

[189] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems – Specification*. Springer-Verlag, Berlin, Germany, 1992.

[190] Moritz Martens. *Establishing Properties of Interaction Systems*. PhD thesis, University of Mannheim, 2009.

[191] Malcolm Douglas McIlroy. Mass-Produced Software Components. In Peter Naur and Brian Randell, editors, *Proceedings of the NATO Software Engineering Conference*, pages 138–155. NATO Science Committee, Brussels, Belgium, 1969.

[192] Nenad Medvidović and Richard Newton Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering (TSE)*, 26(1):70–93, 2000.

[193] Albert Ronald Meyer and Larry Joseph Stockmeyer. The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space. In *Proceedings of the 13th Annual Symposium on Switching and Automata Theory (SWAT 1972)*, pages 125–129. IEEE Computer Society, Los Alamitos, CA, USA, 1972.

[194] Steven P. Miller, Michael William Whalen, and Darren Duane Cofer. Software Model Checking Takes Off. *Communications of the ACM (CACM)*, 53(2):58–64, 2010.

[195] Arthur John Robin Gorell Milner. Algebras for Communicating Systems. In *Proceedings of the AFCET/SMF Joint Colloquium in Applied Mathematics*, Paris, France, 1978. Also available as Technical Report CSR-25-78, Computer Science Department, University of Edinburgh, 1978.

[196] Arthur John Robin Gorell Milner. An Algebraic Theory for Synchronization. In Klaus Weihrauch, editor, *Proceedings of the 4th GI Conference on Theoretical Computer Science*, volume 67 of *Lecture Notes in Computer Science*, pages 27–35. Springer-Verlag, Berlin, Germany, 1979.

[197] Arthur John Robin Gorell Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 1980.

[198] Arthur John Robin Gorell Milner. Four Combinators for Concurrency. In *Proceedings of the 1st ACM Symposium on Principles of Distributed Computing (PODC 1982)*, pages 104–110. ACM, New York, NY, USA, 1982.

[199] Arthur John Robin Gorell Milner. Calculi for Synchrony and Asynchrony. *Theoretical Computer Science (TCS)*, 25(3):267–310, 1983.

[200] Arthur John Robin Gorell Milner. *Communication and Concurrency*. Prentice Hall, Upper Saddle River, NJ, USA, 1989.

[201] Arthur John Robin Gorell Milner. Operational and Algebraic Semantics of Concurrent Processes. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 1201–1242. Elsevier B.V., Amsterdam, The Netherlands, 1990.

[202] Arthur John Robin Gorell Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, Part I. *Information and Computation*, 100(1): 1–40, 1992.

[203] Arthur John Robin Gorell Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, Part II. *Information and Computation*, 100 (1):41–77, 1992.

[204] Christoph Friedrich Minnameier. Local and Global Deadlock-Detection in Component-Based Systems Are NP-Hard. *Information Processing Letters (IPL)*, 103(3):105–111, 2007.

[205] Christoph Friedrich Minnameier. *Interaction in Concurrent Systems*. PhD thesis, University of Mannheim, 2010.

[206] Fabrizio Montesi and Davide Sangiorgi. A Model of Evolvable Components. In Martin Wirsing, Martin Hofmann, and Axel Rauschmayer, editors, *Proceedings of the 5th International Conference on Trustworthly Global Computing (TGC 2010)*, volume 6084 of *Lecture Notes in Computer Science*, pages 153–171. Springer-Verlag, Berlin, Germany, 2010.

[207] Alexandre Cabral Mota, Rodrigo Ramos, and Augusto Cezar Alves Sampaio. Systematic Development of Trustworthy Component Systems.

In Ana Cavalcanti and Dennis Dams, editors, *Proceedings of the 2nd World Congress on Formal Methods (FM 2009)*, volume 5850 of *Lecture Notes in Computer Science*, pages 140–156. Springer-Verlag, Berlin, Germany, 2009.

[208] Ian Munro. Efficient Determination of the Transitive Closure of a Directed Graph. *Information Processing Letters (IPL)*, 1(2):56–58, 1971.

[209] John R. Myhill. Finite Automata and the Representation of Events. Technical Report WADD TR-57-624, Wright-Patterson Air Force Base, Ohio, USA, 1957.

[210] Gleb Naumovich and Lori A. Clarke. Classifying Properties: An Alternative to the Safety-Liveness Classification. *ACM SIGSOFT Software Engineering Notes (SEN)*, 25(6):159–168, 2000.

[211] Anil Nerode. Linear Automaton Transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

[212] Oscar Nierstrasz and Franz Achermann. A Calculus for Modeling Software Components. In Frank Sipke de Boer, Marcello Maria Bonsangue, Susanne Graf, and Willem-Paul de Roever, editors, *Proceedings of the 1st International Symposium on Formal Methods for Components and Objects (FMCO 2002)*, volume 2852 of *Lecture Notes in Computer Science*, pages 339–360. Springer-Verlag, Berlin, Germany, 2003.

[213] Tobias Nipkow, Lawrence Charles Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2002.

[214] Esko Nuutila. *Efficient Transitive Closure Computation in Large Digraphs*. PhD thesis, Helsinki University of Technology, 1995.

[215] Ammar Osaiweran, Tom Fransen, Jan Friso Groote, and Bart J. van Rijnsoever. Experience Report on Designing and Developing Control Components Using Formal Methods. In Dimitra Giannakopoulou and Dominique Méry, editors, *Proceedings of the 18th International Symposium on Formal Methods (FM 2012)*, volume 7436 of *Lecture Notes in Computer Science*, pages 341–355. Springer-Verlag, Berlin, Germany, 2012.

[216] Susan Owicki and Leslie Lamport. Proving Liveness Properties of Concurrent Programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):455–495, 1982.

[217] Sam Owre, John Rushby, and Natarajan Shankar. PVS: A Prototype Verification System. In Deepak Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction (CADE-11)*, volume 607

of *Lecture Notes in Computer Science*, pages 748–752. Springer-Verlag, Berlin, Germany, 1992.

[218] Gordon J. Pace, Frédéric Lang, and Radu Mateescu. Calculating τ-Confluence Compositionally. In Warren Alva Hunt, Jr. and Fabio Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification (CAV 2003)*, volume 2725 of *Lecture Notes in Computer Science*, pages 446–459. Springer-Verlag, Berlin, Germany, 2003.

[219] Robert Allan Paige and Robert Endre Tarjan. Three Partition Refinement Algorithms. *SIAM Journal on Computing (SICOMP)*, 16(6):973–989, 1987.

[220] Christos Harilaos Papadimitriou. *Computational Complexity*. Addison-Wesley Longman Publishing Co., Inc., Reading, MA, USA, 1994.

[221] Pavel Parízek and František Plášil. Modeling Environment for Component Model Checking from Hierarchical Architecture. In *Proceedings of the 3rd International Workshop on Formal Aspects of Component Software (FACS 2006)*, volume 182 of *Electronic Notes in Theoretical Computer Science*, pages 139–153. Elsevier B.V., Amsterdam, The Netherlands, 2007.

[222] David Michael Ritchie Park. Concurrency and Automata on Infinite Sequences. In Peter Deussen, editor, *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, Berlin, Germany, 1981.

[223] David Lorge Parnas. On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM (CACM)*, 15:1053–1058, 1972.

[224] Doron Angel Peled. All from One, One for All: On Model Checking Using Representatives. In Costas Courcoubetis, editor, *Proceedings of the 5th International Workshop on Computer Aided Verification (CAV 1993)*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer-Verlag, Berlin, Germany, 1993.

[225] Doron Angel Peled. *Software Reliability Methods*. Springer-Verlag, Berlin, Germany, 2001.

[226] Dewayne Elwood Perry and Alexander L. Wolf. Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes (SEN)*, 17(4):40–52, 1992.

[227] Carl Adam Petri. *Kommunikation mit Automaten*. Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, Bonn, Germany, 1962. Also available as *Communication with Automata*, Technical Report RADC-

TR-65-377, volume 1, supplement 1. Griffiss Air Force Base, New York, NY, USA, 1966.

[228] František Plášil and Stanislav Višňovský. Behavior Protocols for Software Components. *IEEE Transactions on Software Engineering (TSE)*, 28 (11):1056–1076, 2002.

[229] František Plášil, Dušan Bálek, and Radovan Janeček. SOFA/DCUP: Architecture for Component Trading and Dynamic Updating. In *Proceedings of the 4th International Conference on Configurable Distributed Systems (ICCDS 1998)*, pages 43–52. IEEE Computer Society, Los Alamitos, CA, USA, 1998.

[230] Amir Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, pages 46–57. IEEE Computer Society, Los Alamitos, CA, USA, 1977.

[231] Amir Pnueli, Natarajan Shankar, and Eli Singerman. Fair Synchronous Transition Systems and Their Liveness Proofs. In Anders Peter Ravn and Hans Rischel, editors, *Proceedings of the 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT 1998)*, volume 1486 of *Lecture Notes in Computer Science*, pages 198–209. Springer-Verlag, Berlin, Germany, 1998.

[232] Emil Leon Post. Formal Reductions of the General Combinatorial Decision Problem. *American Journal of Mathematics*, 65(2):197–215, 1943.

[233] Arthur Norman Prior. *Time and Modality*. Oxford University Press, Oxford, UK, 1957.

[234] Arthur Norman Prior. *Past, Present, and Future*. Oxford University Press, Oxford, UK, 1967.

[235] Jean-Pierre Queille and Joseph Sifakis. Specification and Verification of Concurrent Systems in CESAR. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *Proceedings of the 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, Berlin, Germany, 1982.

[236] Michael Oser Rabin and Dana Stewart Scott. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[237] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, Axel Legay, and Roberto Passerone. Modal Interfaces: Unifying Interface Automata and Modal Specifications. In *Proceedings of the 9th International*

Conference on Embedded Software (EMSOFT 2009), pages 87–96. ACM, New York, NY, USA, 2009.

[238] Michel Adriaan Reniers and Timothy Ariën Carol Willemse. Folk Theorems on the Correspondence between State-Based and Event-Based Systems. In Ivana Černá, Tibor Gyimóthy, Juraj Hromkovič, Keith Jefferey, Rastislav Královič, Marko Vukolić, and Stefan Wolf, editors, *Proceedings of the 37th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2011)*, volume 6543 of *Lecture Notes in Computer Science*, pages 494–505. Springer-Verlag, Berlin, Germany, 2011.

[239] John Edmund Savage. *Models of Computation: Exploring the Power of Computing*, 1st edition. Addison-Wesley Longman Publishing Co., Inc., Reading, MA, USA, 1997.

[240] Walter John Savitch. Relationships Between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences (JCSS)*, 4(2):177–192, 1970.

[241] Nils Semmelrock. Analysis of Cooperating Systems by Refined Over-Approximations (Extended Abstract). Abstract accepted for presentation at the FACS 2011 Doctoral Track, 2011.

[242] Mary Margaret Shaw and David Barnard Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, Upper Saddle River, NJ, USA, 1996.

[243] Joseph Sifakis. A Framework for Component-Based Construction (Extended Abstract). In *Proceedings of the 3rd IEEE International Conference on Software Engineering and Formal Methods (SEFM 2005)*, pages 293–300. IEEE Computer Society, Los Alamitos, CA, USA, 2005.

[244] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*, 8th edition. John Wiley & Sons, Hoboken, NJ, USA, 2008.

[245] Aravinda Prasad Sistla. On Characterization of Safety and Liveness Properties in Temporal Logic. In *Proceedings of the 4th ACM Symposium on Principles of Distributed Computing (PODC 1985)*, pages 39–48. ACM, New York, NY, USA, 1985.

[246] Aravinda Prasad Sistla and Edmund Melson Clarke. The Complexity of Propositional Linear Temporal Logics. *Journal of the ACM (JACM)*, 32(3): 733–749, 1985.

[247] Jiří Srba. On the Power of Labels in Transition Systems. In Kim Guld-

strand Larsen and Mogens Nielsen, editors, *Proceedings of the 12th International Conference on Concurrency Theory (CONCUR 2001)*, volume 2154 of *Lecture Notes in Computer Science*, pages 277–291. Springer-Verlag, Berlin, Germany, 2001.

[248] Andrew James Stothers. *On the Complexity of Matrix Multiplication*. PhD thesis, Graduate School of Mathematics, University of Edinburgh, 2010.

[249] Clemens Alden Szyperski. *Component Software: Beyond Object-Oriented Programming*, 2nd edition. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[250] Andrew Stuart Tanenbaum. *Modern Operating Systems*, 3rd edition. Prentice Hall, Upper Saddle River, NJ, USA, 2007.

[251] Robert Endre Tarjan. A Note on Finding the Bridges of a Graph. *Information Processing Letters*, 2(6):160–161, 1974.

[252] Axel Thue. Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln. In *Videnskabs-Selskabets Skrifter Matematisk-naturvidenskabelige Klasse*, number 10. Kristiania, Oslo, Norway, 1914.

[253] Stavros Tripakis, Ben Lickly, Thomas Anton Henzinger, and Edward Ashford Lee. On Relational Interfaces. In *Proceedings of the 9th International Conference on Embedded Software (EMSOFT 2009)*, pages 67–76. ACM, New York, NY, USA, 2009.

[254] Alan Mathison Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1):230–265, 1937.

[255] Antti Valmari. Stubborn Sets for Reduced State Space Generation. In Grzegorz Rozenberg, editor, *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets (PETRI NETS 1990)*, volume 483 of *Lecture Notes in Computer Science*, pages 491–515. Springer-Verlag, Berlin, Germany, 1991.

[256] Antti Valmari. The State Explosion Problem. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528. Springer-Verlag, Berlin, Germany, 1998.

[257] Antti Valmari. Bisimilarity Minimization in $O(m \log n)$ Time. In Giuliana Franceschinis and Karsten Wolf, editors, *Proceedings of the 30th International Conference on Applications and Theory of Petri Nets (PETRI NETS 2009)*, volume 5606 of *Lecture Notes in Computer Science*, pages 123–142. Springer-Verlag, Berlin, Germany, 2009.

[258] Moshe Ya'akov Vardi. From Church and Prior to PSL. In Orna Grumberg and Helmut Veith, editors, *Proceedings of the Symposium on 25 Years of Model Checking (25MC)*, volume 5000 of *Lecture Notes in Computer Science*, pages 150–171. Springer-Verlag, Berlin, Germany, 2008.

[259] Virginia Vassilevska Williams. Breaking the Coppersmith-Winograd Barrier. Manuscript, 2011.

[260] Richard Veryard. *The Component-Based Business: Plug and Play*. Springer-Verlag, Berlin, Germany, 2001.

[261] Stephen Warshall. A Theorem on Boolean Matrices. *Journal of the ACM (JACM)*, 9(1):11–12, 1962.

[262] James Christopher Westland. The Cost of Errors in Software Development: Evidence from Industry. *Journal of Systems and Software (JSS)*, 62 (1):1–9, 2002.

[263] Jeannette Marie Wing. Formal Methods. In *Encyclopedia of Software Engineering*, pages 504–517. John Wiley & Sons, New York, NY, USA, 1994.

[264] Bang Ye Wu and Kun-Mao Chao. *Spanning Trees and Optimization Problems*. Discrete Mathematics and Its Applications. Chapman & Hall/CRC, Boca Raton, FL, USA, 2004.

Index

Symbols

A

B

C

D

E

G

I

K

L

M

N

O

P

Q

R

S

T

U

V

W