Interaction in Concurrent Systems

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

Christoph Friedrich Minnameier
aus Heidelberg

Mannheim, April 2010

Dekan:	Professor Dr. Felix Freiling, Universität Mannheim
Referentin:	Professor Dr. Mila Majster-Cederbaum, Universität Mannheim
Korreferent:	Professor Dr. Ulrich Hertrampf, Universität Stuttgart

Tag der mündlichen Prüfung: 29. April 2010

Abstract

This dissertation is concerned with the theoretical analysis of component-based models for concurrent systems. We focus on interaction systems, which were introduced by Sifakis et al. in 2003. Centered around interaction systems, we also cover Minsky machines, Petri nets and the Linda calculus and establish relations between the models by giving translations from one to the other. Thus, we gain an insight concerning the expressiveness of the models and learn, given a system described in one syntax, how to simulate it in another. Additionally, these translations allow us to deduce complexity and undecidability results. Namely, we show that the questions whether a LinCa process terminates or diverges under a maximum progress semantics are undecidable. We also prove that the problems of reachability, progress, local and global deadlock and availability are PSPACE-complete in interaction systems.

This complexity-theoretic classification serves as a motivation for the sufficient condition approach that is presented in the second half of this work: We present a generic approach to prove properties for component-based systems that allow for decomposition into subsystems. To avoid the problem of state space explosion, we consider overlapping projections and thus compute over-approximations of the reachable global state space. We enhance the quality of these over-approximations by a technique we call Cross-Checking. Based on the enhanced over-approximations, we may then prove properties of the global system in polynomial time. We demonstrate our ideas by means of interaction systems and for the property of local deadlock.

Zusammenfassung

Diese Dissertation befasst sich mit der theoretischen Analyse komponenten-basierter Modelle für nebenläufige Systeme. Im Mittelpunkt steht dabei das Modell der Interaktionssysteme, welches im Jahr 2003 von Sifakis et al. eingeführt wurde. Im Kontext von Interaktionssystemen betrachten wir Minsky-Maschinen, Petri-Netze und den Linda Kalkül und setzen die verschiedenen Modelle durch Übersetzungen zueinander in Beziehung. Somit erhalten wir einen Einblick in die Ausdrucksstärke der Modelle und erfahren, wie man ein Modell, welches in einer Syntax gegeben ist, mittels einer anderen simulieren kann. Zusätzlich erlauben die genannten Übersetzungen die Folgerung von Komplexitäts- und Entscheidbarkeitsaussagen. Genauer gesagt wird gezeigt, dass die Fragen, ob ein LinCa Prozess terminiert bzw. divergiert unter einer Semantik, die maximalen Fortschritt fordert, unentscheidbar sind. Wir zeigen außerdem, dass die Probleme Erreichbarkeit, Fortschritt, Lokaler und Globaler Deadlock, sowie Verfügbarkeit in Interaktionssystemen PSPACE-vollständig sind.

Diese komplexitätstheoretische Klassifizierung dient als Motivation für den Ansatz einer hinreichenden Bedingung, der in der zweiten Hälfte der Arbeit vorgestellt wird: Wir demonstrieren eine allgemeingültige Methode, Eigenschaften von komponenten-basierten Systemen zu beweisen, die eine Zerlegung in Teilsysteme erlauben. Um das Problem der Zustandsraumexplosion zu vermeiden, betrachten wir überlappende Projektionen und berechnen damit Überapproximationen des global erreichbaren Zustandsraums. Wir verbessern die Qualität dieser Überapproximationen dann mit einer Technik, die wir Cross-Checking nennen. Basierend auf den verbesserten Überapproximationen können wir schließlich Eigenschaften des globalen Systems in polynomieller Zeit beweisen. Wir veranschaulichen unsere Ideen anhand von Interaktionssystemen und für die Eigenschaft Lokaler Deadlock.

"Ein Mensch, der um anderer willen, ohne dass es seine eigene Leidenschaft, sein eigenes Bedürfnis ist, sich um Geld oder Ehre oder sonst etwas abarbeitet, ist immer ein Tor."

- Johann Wolfgang von Goethe -
"Die Leiden des jungen Werther, Brief vom 20. Julius"

Contents

Acknowledgments

First, I am grateful to Professor Mila Majster-Cederbaum for introducing me to this new area of research and for being a great supervisor. Apart from her helpful technical guidance, I thank her for her kind and supportive nature. She was always approachable and still allowed for unsupervised phases whenever I felt I needed to work things out on my own.

I also thank the second referee Professor Ulrich Hertrampf. He aroused my interest in theoretic computer science and inspired me with his excellent way of teaching when I studied at the University of Stuttgart.

Thanks to my fellow workers Moritz, Nils and Christian for making my time very endurable and also for proof-reading and fruitful discussions.

I thank my parents for teaching me tolerance, humor and love as axioms – and for teaching me rationalism to deduce all other virtues in life myself.

I thank Anna for bringing meaning to my life and for pointing out to me from time to time that the above mentioned axioms are much more important than rationalism or anything you can deduce from them.

Chapter 1

Introduction

1.1 Motivation

In 1965, Intel co-founder Gordon E. Moore [Moo65] prognosticated that the maximum available computation speed of a processor would be doubled by developers every 18 months. This claim has been found astonishingly accurate ever since the first personal computers were manufactured.

> *The complexity for minimum component costs has increased at a rate of roughly a factor of two per year ... Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000. I believe that such a large circuit can be built on a single wafer.*

In the past decades, hardware developers have been predicting all the way that the time draws nearer when *Moore's Law* will cease to be valid. Although Moore's Law still seems to be correct, such doubts are not easily dismissed because it seems quite clear that making processing units ever smaller and thus ever faster is not a process that can go on forever.

Independently of the validity of either perspective, recent hardware design tendencies show that information technology is, in practically all areas, ever more relying on multiple cooperating processing units rather than single over-powered cores. Where some ten or twenty years ago, only supercomputers used to consist of multiple processing units, nowadays even commodities like video consoles possess multiple processing units, the best possible example for this is Sony's PS3, whose CELL processor contains nine processing units in one chip [KBLD08].

In general, reasons for this trend can easily be named as the two sides of the same coin: the need for ever stronger processing power on the one hand and the economic want for ever cheaper processing power on the other hand.

While multiple processor systems, communicating software entities or communicating processes, which we all abstract under the name of *concurrent systems*, provide the benefits of reusability and cheap computation power, they also pose specific, hitherto unencountered problems to their developers.

One of the most important and probably the most prominent among these problems is the question whether a system can reach a deadlock, which was illustrated in 1971 by the dutch computer scientist Edsger Dijkstra: Dijkstra set an examination question about a synchronization problem where five computers competed for access to five shared tape drive peripherals. Soon afterwards the problem was retold by Tony Hoare as the *Dining Philosophers Problem* (see Figure 1.1).

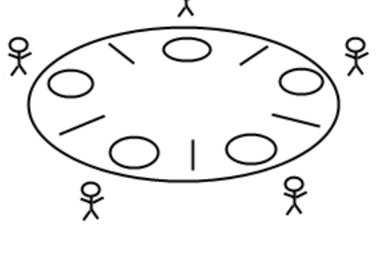


Figure 1.1: The Dining Philosophers Problem

Five philosophers sitting around a table with one fork between every pair of them want to have dinner. Initially, each philosopher is thinking. At every point of time a thinking philosopher can grab one of her adjacent forks. Once she has both forks, she can eat and then put down the forks to resume thinking. As this is a model for a concurrent system, the philosophers act independently of each other, i.e., philosopher one might grab the fork to her right, then philosopher three might grab the fork to her left and then the one to her right, then philosopher one might grab the fork to her left and eat, then philosopher three might eat and so on.

Dijkstra's well-known Dining Philosophers example illustrates the problem of *deadlock*, which is, roughly spoken, the question whether a system (in this case the dinner-party) can reach a situation where neither of the participants will be able to take any further action. Indeed, at second glance, we notice that the philosophers' independence from each other is subject to a constraint: Only one of them at a time can access a single common resource (i.e., a fork): E.g., when philosopher one has taken her left fork, then philosopher two will not be able to access this fork (i.e., her right one).

Hence, if every philosopher stops thinking and grabs the fork to her right, neither of them will be able to get hold of both forks in order to be able to eat. Such a behavior is usually considered unwanted in concurrent systems and a major part of concurrency theory consists of finding ways to decide problems like the existence of a deadlock.

When computer science first encountered these problems (that are often hard to detect in both hardware and software architectures) the need for formalizations arose and was served by the emergence of a new scientific area called formal methods. Formal methods are concerned with *specification* and *verification*, i.e., based on a *model* that can be seen as a mathematic abstraction of the respective architecture the system instance in question is described (specification). Then certain proof techniques – in some cases partly human-directed – are applied in order to prove the desired properties (verification).

Since the first approaches that not only check systems for failures, but also prove certain properties, one can identify five major approaches that try to reach this goal in different ways[1]:

i) Deductive Program Verification,

ii) Abstract Interpretation,

iii) Sufficient Conditions,

iv) Model-Checking,

v) Equivalences.

Deductive program verification goes back to Floyd and Hoare [Flo67, Hoa69] and combines explicit code with logic to formally verify programs. Abstract interpretation comprises approaches that abstract from certain aspects of a system (or program) to make the verification of properties easier. Sufficient conditions cover a large bandwidth of techniques. We use this term for all approaches that try to verify properties of a system under certain circumstances (resp. preconditions). Model-checking proves for a system, which is usually represented by an automaton M, a property which is expressed in a formula of temporal logic, e.g., an LTL-formula ϕ. A more direct approach that is based on the equivalences which are used to describe similarities between (transition) systems is, given a pair of systems, the automatic computation of such equivalences.

Formal methods' major difficulty is to handle large scale systems, due to the problem of *state space explosion*: As a state of the entire system is an element of the cross-product of the state spaces of the n communicating entities, the so called global state space may become exponentially large in n. On the other hand in order to verify, e.g., that a system does not contain a deadlock, it seems – at first glance – unavoidable to explore the global state space.

This thesis deals with issues of decidability and computational complexity in concurrent systems. Its focus lies on models where computation and com-

[1]Here, we confine with sketching these approaches. For a more detailed discussion, especially w.r.t. the techniques presented in this work, see Chapter 7.

munication are separated and on *interaction systems* in particular. Interaction systems are a model for component-based concurrent systems that was introduced by Joseph Sifakis et al. in 2003 [GS03]. As a representative of component-based systems, interaction systems build on components, i.e. reusable (software or hardware) entities that are to a certain degree independent of their environment. A component can be represented by an interface and a behavioral model. Additionally, component-based systems provide the so-called *glue code*, which is used to specify the communication among components and thus build systems from components. This idea – of separating sequential computation from the communication necessary to make the sequential entities work together – also manifests in coordination languages and calculi.

Our aim is to provide a better understanding of the computational capabilities of interaction systems by classifying them among other well-known and well-understood models for concurrency. The translations given between the models not only enable us to reason about a notion of *expressiveness* but also yield complexity results that motivate the sufficient conditions for deadlock-freedom that are presented in this thesis.

Given the (theoretically proven) difficulty of the discussed problems we intend to contribute new ideas to the general area of verification that has not yet found the silver bullet against state space explosion. We introduce an approach to prove properties of component-based systems in which we build subsystems of component-based systems in order to over-approximate the reachable global state space and prove global properties based on predicates on the over-approximation and underline the effectiveness of our approaches by case studies.

1.2 Contribution

1.2.1 Classifying Interaction Systems among related Models of Concurrency

Focusing on interaction systems, we establish relations between the models of our interest by giving translations between them that preserve certain properties, depending on the respective purpose of such a mapping.

Figure 1.2 displays and relates the various models, where we use the following abbreviations: $\widetilde{IS}$ denotes Sifakis' original notion of interaction systems that we extend to a more general class. *1SN* stands for *1-Safe nets* and CFN for *communication-free nets*. While both are subclasses of Petri nets, the former can be considered a finite model while the latter can not, a fact that is also sketched in the picture. Finally, $LinCa_{MTS-mp}$ is used to denote the Linda calculus under a maximum progress semantics. All of these classes are formally defined in Chapter 2. The notion of finite, resp. infinite model refers to the behavior that can be modeled. Cyclic behavior that allows for infinite traces by looping over a finite set of global states is considered finite, whereas models that allow for an infinite state space and thus the existence of non-cyclic infinite traces are – in our terminology – considered infinite.

The arrows in Figure 1.2 represent the translations and are marked by the number of the corresponding section where we present the translation. The picture also includes Yoram Hirshfeld's [Hir94] translation from Minsky machines to communication-free nets (CFN) that inspired our translation from Minsky machines to $LinCa_{MTS-mp}$.

1.2.2 Complexity & Undecidability

When the work on this thesis started, there existed no complexity-theoretic knowledge about problems in interaction systems. As a first contribution in this area we present an NP-hardness result for the problems of local and

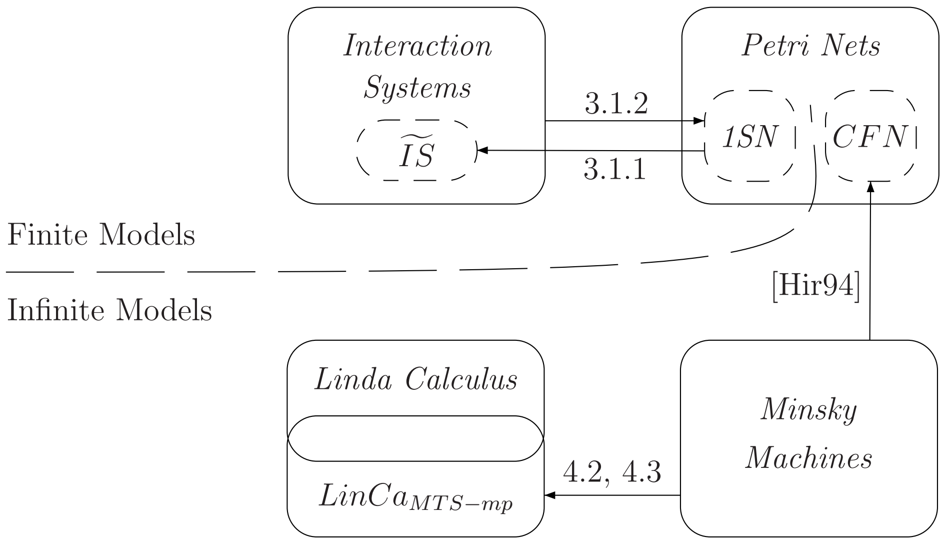


Figure 1.2: Classification Overview

global deadlock in interaction systems (cf. [Min07][2]). An NP-hardness result for progress that is established following the idea of [Min07] can be found in [MMM06] resp. [MMM07b].

A more thorough examination yields a mapping from 1-Safe-nets to interaction systems that implies PSPACE-hardness for reachability and liveness in interaction systems [MM08b]. Together with a chain of reductions and the proof that availability is in PSPACE, we may deduce PSPACE-completeness for most relevant problems in interaction systems [MM08c].

For the *Linda* calculus under a maximum progress semantics ($LinCa_{MTS\text{-}mp}$), we give reductions from *Minsky machines* to *LinCa* which preserve termination respectively divergence and thus imply undecidability of these questions in $LinCa_{MTS\text{-}mp}$ [MM06].

Figure 1.3 displays the undecidability and complexity results that are presented in this thesis. Note that the figure does not contain implications but

[2]This first work on complexity issues in interaction systems served as a motivation for [GGM$^+$07b], where the authors try to ensure properties of interaction systems by construction.

is restricted to the strongest results[3]. Again, edge labels denote the corresponding sections that present the results.

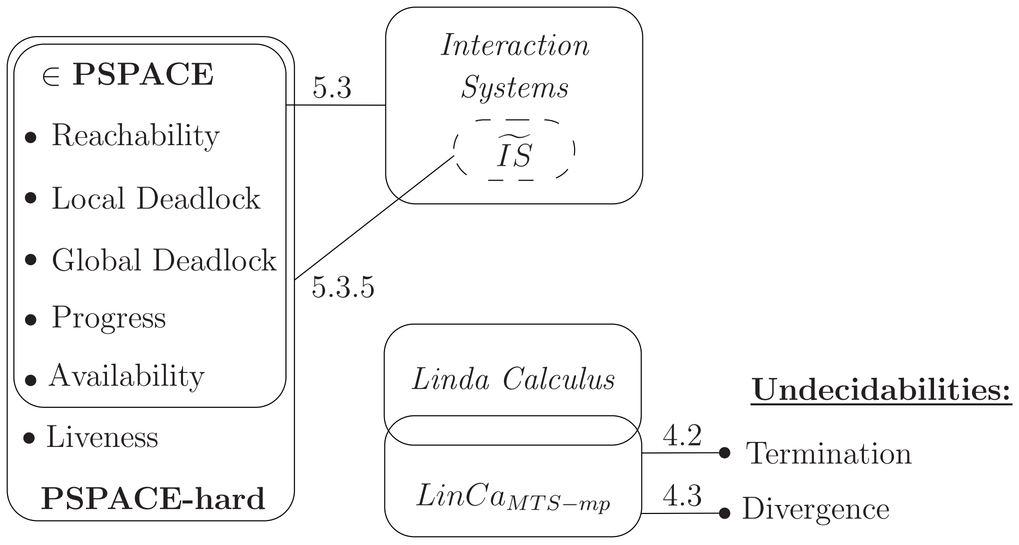


Figure 1.3: Complexity & Undecidability Overview

1.2.3 An efficient Approach

The greatest obstacle to overcome for formal verification has always been state space explosion, i.e., the exponential growth that occurs in the size of the Cartesian product of the local state spaces when increasing the number of components. There exist various approaches to tackle state space explosion and we give a specific discussion of closely related work as well as a more general discussion in Chapter 7.

After establishing the result that (assuming $P \neq NP$) no universal polynomial-time algorithm exists that solves the problem of deadlock for arbitrary interaction systems, we present a polynomial-time algorithm (cf. Algorithm 2, p. 123) that investigates so-called *subsystems* of an interaction system and

[3]I.e., the hardness and undecidability results carry over to supersets and the computability results carry over to subsets.

tries to verify deadlock-freedom based on the information gained in the subsystems [MMM07a].

Our first approach consists of three aspects: We build on reachability analyses for the subsystems. In order to avoid state space explosion we restrict our observations to subsystems of a certain parametrized size d (which becomes manifest as the degree of our polynomial time bound). Then, we prove a property for the global system by means of locally checkable predicates (i.e., predicates on states of the subsystems) that imply deadlock-freedom of corresponding global states. This first approach is able to prove non-trivial systems (cf. Example 6.2, p. 125) deadlock-free even if we restrict the size of observed subsystems to 3. The system is non-trivial and it has an exponentially large reachable global state space and interactions of arbitrary size.

We sketch the basic idea in the center of Figure 1.4 and arrange the aspects in which we improve it in a circular manner around it. The labels of the edges denote in which section we introduce the respective improvement. The ideas that we apply to improve our basic approach can be described as follows.

Firstly, we improve the results of [MMM07a] by enhancing the information that is obtained by the subsystem reachability analyses but restrict ourselves to methods that have at most the same asymptotic complexity as the basic subsystem analyses. We introduce the Cross-Checking technique for reachability, which compares the information given by the subsystem reachability analyses and checks them against each other to distill more information than initially available. Thus, it refutes the reachability of certain global states without exceeding our hitherto established time bounds.

As a next step, we enhance our sufficient condition in order to take into account the information that is given by subsystems of size $d > 3$. In other words, we formulate a dynamic condition (that scales with d). Namely, we distinguish between local deadlocks of size smaller than (or equal to) d and larger than d and detect the smaller ones directly while introducing a condition that excludes the existence of larger ones.

In order to further minimize the number of potential deadlocks, we modify our Cross-Checking technique to prove that certain subsystem states can not be part of large deadlocks. We call this technique Cross-Checking for uncriticalness.

Finally, we argue that restricting all considerations to subsystems that are connected (in the sense of the components' communication structure) does not affect any results established so far and may reduce the number of investigated subsystems to a linear bound even for non-trivial systems.

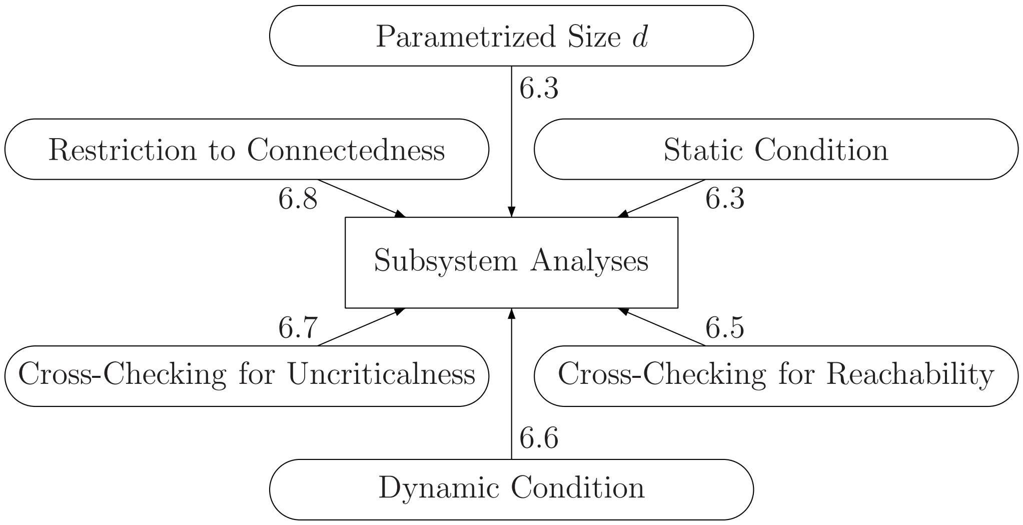

Parametrized Size d
6.3
Restriction to Connectedness
Static Condition
6.8
6.3
Subsystem Analyses
6.7
6.5
Cross-Checking for Uncriticalness
Cross-Checking for Reachability
6.6
Dynamic Condition

Figure 1.4: Efficient Approaches Overview

From a complexity theoretic point of view, our contribution in this topic can be formulated as follows. By our various enhancements of the basic idea of a subsystem reachability analysis we construct an approach that proves (arbitrarily large instances of) Tanenbaum's solution to the Dining Philosophers deadlock-free in polynomial-time by investigating only a linear number of subsystems of size $d = 5$.

To measure the quality of our general approach, respectively the various improvements that we already mentioned, we give – along with our presentation – case studies for each of the respective steps.

1.3 Road Map

We start out in Chapter 2 by presenting the investigated models, i.e., we give an explanation and a formal definition and in some cases provide definitions and motivations for variations of the respective models. The models we discuss in the context of *interaction systems (IS)* are *Petri nets (PN)*, the *Linda calculus (LinCa)* and *Minsky machines (MM)*. We also define problems for the various models and compare them w.r.t. design principles for concurrent systems. We finish Chapter 2 by defining some equivalence notions that are used in this work.

Chapter 3 establishes relations between some of our models (with the focus on *IS*) and between some variations of *LinCa*. Namely, we give translations between *IS* and *1SN* that yield isomorphism (up to a label relation) between the respective global transition systems. Also, we compare three variants of *LinCa* w.r.t the traces that occur in their respective global transition systems.

In Chapter 4 we present undecidability results for *LinCa* under a maximum progress semantics. We provide reductions from Minsky machines to LinCa that prove that the problems of *termination* and *divergence* are undecidable for $LinCa_{MTS-mp}$.

For the model of *IS* we present complexity results in Chapter 5: As an introduction, we give an NP-hardness result for Local and Global Deadlock. Then we present a chain of reductions that – building on the PSPACE-hardness of reachability that is established by the translation function from Petri nets to interaction systems, presented in Section 3.1.1 – proves *Reachability*, *Local* and *Global Deadlock*, *Progress* and *Availability* to be PSPACE-complete in *IS*.

Finally, in Chapter 6 we present a sufficient condition for deadlock-freedom in *IS*. The approach (that carries over to other component-based models that feature multi-party synchronizations) starts with a relatively simple and strong condition that investigates subsystems of a parametrized size d and tries to prove a locally checked predicate. We then generalize and enhance

the approach. Firstly, by applying more sophisticated conditions that refute the existence of deadlocks and secondly by providing a method that significantly enhances the quality of our subsystem reachability approximations without raising the asymptotic time bounds of the overall procedure. The ideas are illustrated by the example of Tanenbaum's version of Dijkstra's Dining Philosophers where we derive empiric data from an implementation of our approach that can be found in [MS08].
We give a conclusion in Chapter 7, summing up our contributions and classifying our techniques for efficient deadlock-detection among other approaches of formal methods.

Chapter 2

Models & Equivalences

In this chapter, we introduce Sifakis' *interaction systems* and along with them the various other models that we investigate and discuss in their context.

We start out by defining two variants of interaction systems: The original version of Sifakis that is subject to certain syntactic constraints and a relaxed, more general version.

Then we introduce Petri nets that, like interaction systems, feature multi-party synchronizations. In contrast to interaction systems, Petri nets do not feature compositionality (at least not in the sense that the identity of composed structures is preserved). As general Petri nets are an infinite model and thus are not appropriate for a comparison with interaction systems, we also define the subclass of 1-safe Petri nets.

We introduce *Linda* as a representative for the class of *coordination languages*. Coordination languages – very much like component-based systems – reflect the *orthogonality paradigm*, i.e., the demand to separate the ideas of computation and communication, a fact that Ciancarini [CJY95] described by the equation "Programming = Computation + Communication". *Linda* and coordination languages in general are thus closely related to component-based models.

Finally, we introduce *Minsky machines*, a Turing complete model for sequential computation. Building upon the fact that termination is undecidable for Minsky machines, we can prove undecidability of, e.g., termination by encoding (in a termination-preserving manner) Minsky machines in other models.

The various models and their respective variations that are discussed, investigated and applied in proofs in this thesis are enlisted here as follows (where the number in parentheses gives the corresponding section of their introduction).

- Interaction Systems (2.3)
    - ▷ *IS* (Generalized interaction systems)
    - ▷ $\widetilde{IS}$ (Original interaction systems)
- Petri Nets (2.4)
    - ▷ *PN* (Petri nets)
    - ▷ *1SN* (1-safe nets)
    - ▷ *CFN* (Communication-free nets)
- Linda (2.5)
    - ▷ LinCa_{ITS} (Interleaving semantics)
    - ▷ LinCa_{MTS} (Multi-step semantics)
    - ▷ $\text{LinCa}_{MTS\text{-}mp}$ (Multi-step semantics with maximum progress)
- Minsky Machines (2.6)
    - ▷ *MM* (Minsky machines)

In each section, we present a model and discuss its particular (dis-)advantages over other models. In order to have the corresponding terminology at hand when we introduce a model and point out its specific characteristics, we introduce some notions in the next section that will be used throughout this chapter and especially when we finally enlist and oppose the properties of the models with respect to each other in Figure 2.1 (p. 20).

2.1 *Characteristic Properties of Models for Concurrent Systems*

Interleaving vs. True-Concurrency

One of the most important and also most controversially discussed issues in concurrency theory is the discussion of *interleaving* semantics vs. *true concurrency*. The distinctions between the two points of view are best displayed by means of a process algebra like *CCS* [Mil89]:
In interleaving semantics we consider the equation $a||b = a.b + b.a$ to be valid, i.e., the parallel execution of two actions a and b is interpreted as the execution of a and then b or b and then a.

The interleaving view argues that two actions never take place at exactly the same time and so they occur one after the other. If we consider this to be true it is convenient to consider the interleaving view as a nice foundation for elegant definitions of models for concurrency. There are however situations when this "abstraction" of reality ceases to be precise enough for our purposes. E.g., if we want to take into account the cases where a starts, then b starts, then b ends and finally a ends (which is a very realistic scenario in concurrent programming) then the interleaving point of view is no longer sufficient.

In this thesis, we investigate both points of view for the Linda calculus. We also define a semantics that does not only allow for true concurrency synchronizations but assumes a common clock for all processes and demands *maximum progress*, i.e., in every clock cycle we have to perform a maximal (w.r.t. set inclusion) amount of actions.

Finite vs. Infinite Models

When we speak of finite, respectively infinite models, we do not refer to their syntactic description (which we clearly want to be finite in all cases) but to their behavior, or more precisely, their state space. For each model that we introduce in this chapter we define a transition system $T = (Q, Lab, \rightarrow, q^0)$

(cf. Definition 2.1, p. 21) that describes its behavior. We call a model infinite iff it allows for a state space Q whose cardinality is infinite. Of course, even models with a finite state space (e.g., interaction systems) feature infinite behavior. However, this is always due to a loop over a sequence of interactions. In contrast to this, general Petri nets or the Linda calculus feature an infinite state space and thus infinite non-repetitive traces due to the unboundedness of their storage (i.e., the places in a Petri net or the tuple space in *LinCa*).

Storage- vs. Channel-based Communication

In concurrent systems we expect multiple processing units to communicate to achieve a common (computation) goal. This communication can be *storage-based*, i.e., unit A writes some piece of information to a shared[1] storage and unit B may read it from there, or *channel-based*, i.e., some piece of information is sent directly over a channel. Communication in interaction systems is channel-based which is reflected in the name of a connector which connects certain ports. For Petri nets one might consider the places as the communicating entities in which case we would view them as channel-based, where the arcs and transitions would be the channels. On the other hand, if we consider the transitions to be the communicating entities, we would say the communication is storage-based through the places[2]. LinCa is clearly storage-based with the tuple space being the storage.

Synchronous vs. Asynchronous Communication

Asynchronous communication describes the fact that the sender of a message does not depend on the receiver in order to send the message. On the other

[1]The extent to which storage is shared may vary: For each processing unit A and each storage unit we can set a value for whether A may read from the storage unit and whether A may write to it.

[2]This view may seem odd at first glance. However, note that a place is lacking anything like a local behavior which a process or a component usually supplies. The fact that a place lacks a "purpose" or a "want" for communication makes it convenient to consider it a mere storage.

hand, she can not rely on the receiver to read the message at once. It may be read later or perhaps never. In contrast, synchronous communication means that the sender has to perform some kind of handshake with the receiver which yields a certain interdependence on the one hand but assures that a message is read at the moment it is sent. Intuitively, the distinction between synchronous and asynchronous communication coincides with the above definition of channel- vs. storage-based communication. There are also special cases where a LiFo- or a FiFo-buffer is used as means of communication. It can be said that synchronous communication can model asynchronous communication (by introducing an auxiliary component that serves as a store) but not the other way round. Interaction systems and Petri nets feature synchronous communication while in LinCa, communication is asynchronous over the tuple space respectively the shared variables.

Degree of Synchronization

It is reasonable to assume that communication can never really be synchronous because there will always be a delay ϵ in communication of two entities, so synchronous communication models something on a higher layer of abstraction that is realized (or approximated) by specific protocols. Similarly, some models provide a means for synchronization between more than two entities for a more intuitive and more compressed description of certain operational sequences. Interaction systems feature multi-party synchronization due to the notion of connectors. Petri nets feature multi-party synchronizations by allowing for more than two ingoing arcs for a transition. LinCa processes on the other hand allow for pairwise synchronization[3] only.

Endogenous vs. Exogenous Communication

In [Arb98], Farhad Arbab classified coordination models and languages as either endogenous or exogenous: In endogenous models and languages there

[3]In this context, we consider a synchronization to be a communication of a process with the tuple space.

exist primitives that must be incorporated within a computation for its coordination. In applications that use such models, primitives that affect the coordination of each module are inside the module itself. In contrast, exogenous models and languages provide primitives that support the coordination of entities from without. In applications that use exogenous models, primitives that affect the coordination of each module are outside the module itself. Endogenous models lead to intermixing of coordination primitives with computation code. This entangles the semantics of computation with coordination, thus making the coordination part inside the application implicit and sometimes nebulous, a fact Petri nets are often accused for. However, endogenous coordination models are quite intuitive for a huge variety of applications: One of the main reasons that the Linda tuple space coordination model has been a reference model in the context of distributed programming for such a long time is its naturalness and flexibility.

Compositionality and Identity Preservation

It is possible for more or less all models for concurrent systems to somehow merge two systems to a single combined system by defining some glue code for their components. When we speak about compositionality we usually refer to the question to which extent one may define "reasonable" general operators that can be used to compose a system out of atomic components or out of already existing subsystems. One of the major aspects of such an operator is the preservation of component identity: As mentioned above, Petri nets somehow mix up communication and computation and once a net has been constructed, one can hardly say for which part of a computation a certain place or a certain transition is responsible. More generally, there is no or hardly any correspondence between the natural, logic decomposition of the overall task into sub tasks on the one hand and the interweavement of "subnets" on the other. In Petri nets, composing a net out of smaller nets lets the parts lose their identity and this makes it impossible to decompose a large net into original parts to reason about global properties on a different layer

of abstraction. When composing an interaction system from components by defining the glue-code, the identity of components is preserved, i.e., the user may still identify single components after composition. This fact is very substantial for the ideas presented in Chapter 6, where build subsystems of a system for a more efficient (approximative) analysis of the reachable global state space.
Figure 2.1 displays the models that are defined in this chapter and connects them to various design properties of formal models. Please note that our models do come from diverse contexts and have different natures that are sometimes hard to compare. We abstain here from justifying every marking "$\times$" in the table but rather aim to give an overview as well as a quick reference for whenever the reader is interested in the commonalities or differences between a pair of models.

	IS	$\widetilde{IS}$	PN	$1SN$	CFN	$LinCa$	$LinCa_{MTS-mp}$	$LinCa_{MTS}$
interleaving	×	×	×	×	×	×		
true concurrency							×	×
maximum progress								×
finite	×	×		×				
infinite			×		×	×	×	×
storage-based						×	×	×
channel-based	×	×	×	×	×			
synchronous	×	×	×	×				
asynchronous					×	×	×	×
pairwise synch.						×	×	×
multi-party synch.	×	×	×	×				
endogenous comm.			×	×	×	×	×	×
exogenous comm.	×	×						
identity pres.	×	×						
identity not pres.			×	×	×	×	×	×

Figure 2.1: Properties of the Models

2.2 Basic Definitions

Definition 2.1
A **labeled transition system** is a quadruple $T = (Q, Lab, \rightarrow, q^0)$, where Q is the (possibly infinite) set of states, Lab is the set of labels and $\rightarrow \subseteq Q \times Lab \times Q$ is a ternary relation (of labeled transitions). $q^0 \in Q$ is the designated starting state. For $q, q' \in Q$ and $a \in Lab$, $(q, a, q') \in \rightarrow$ is also denoted by $q \xrightarrow{a} q'$. This represents the fact that there is a transition from state q to state q' with label a. We write $q \not\rightarrow$ iff $\nexists a \in Lab, q' \in Q$ with $q \xrightarrow{a} q'$.

Remark 2.1
In this Chapter we describe the semantics (respectively the behavior) for different models of computation, always by means of a labeled transition system. Please note that throughout this work we will – for ease of notation – often identify the syntactic description of a system with its behavior.

Definition 2.2
Given a label set Lab, we denote by Lab^* the **Kleene star closure** of Lab. Let Lab^* contain all finite sequences over labels $a \in Lab$, including the empty sequence, respectively the empty word, which we denote by ϵ.

Definition 2.3
Let $\rightarrow \subseteq Q \times Lab \times Q$ be a ternary relation.
Then $\rightarrow^* \subseteq Q \times Lab^* \times Q$ denotes the **reflexive and transitive closure** of $\rightarrow$ by

- $\forall q \in Q \ (q, \epsilon, q) \in \rightarrow^*$.
- $((q, w, q') \in \rightarrow^* \wedge (q', a, q'') \in \rightarrow) \Rightarrow (q, w \circ a, q'') \in \rightarrow^*$,
  where $\circ$ denotes concatenation.

For the following definitions let $T = (Q, Lab, \rightarrow, q^0)$ be a labeled transition system.

Definition 2.4
Let Reach(T) $\subseteq Q$ denote the **reachable state space** of T given by Reach(T) $:= \{q \in Q \mid \exists w \in Lab^* \ (q^0, w, q) \in \rightarrow^*\}$

Definition 2.5
Given a state $q \in Q$ we denote by ***reachability of q*** the question, whether $q \in Reach(T)$.

Definition 2.6
By Traces(T) $\subseteq Lab^*$, we denote the **set of traces** of T, i.e., the words in Lab^* that correspond to a transition sequence starting in q^0.
Traces(T) $:= \{w \in Lab^* \mid \exists q \in Q \ (q^0, w, q) \in \rightarrow^*\}$.

Definition 2.7
We call an infinite transition sequence $q^0 \xrightarrow{\alpha_1} q^1 \xrightarrow{\alpha_2} q^2 \ldots$ which starts in q^0 a **run** of T.

Definition 2.8
Sometimes the set Lab will contain a designated label $\tau \in Lab$ that we call the silent (or internal) action, i.e., an action which is not visible to external observers. For this case, the **visible transition relation** $\rightarrow^+ \subseteq Q \times Lab \setminus \{\tau\} \times Q$ will consist of an arbitrary number of τ transitions followed by a visible transition.
$q \xrightarrow{a}^+ q'$ iff $\exists q^1, ..., q^n \in Q$, s. t. $q \xrightarrow{\tau} q^1 \xrightarrow{\tau} ... \xrightarrow{\tau} q^n \xrightarrow{a} q'$.
If Lab contains a designated silent action τ we replace $\rightarrow$ by $\rightarrow^+$ in the definition of the set of traces. As a consequence, Traces(T) $\subseteq (Lab \setminus \{\tau\})^*$.

Definition 2.9
We say that a labeled transition system T **terminates** if $\exists q \in Reach(T) \ q \not\rightarrow$.

Definition 2.10
We say that a labeled transition system T **diverges** if it contains at least one run.

Remark 2.2
Please note that the Definitions 2.9 and 2.10 obviously allow a labeled transition system to both terminate and diverge.

Definition 2.11
A **multiset** is a set that may include multiple instances of the same element. Given a multiset M, we write $(a, k) \in M$ $(k \geq 0)$ iff M includes exactly k instances of the element a. For ease of notation we will sometimes write $a \in M$ instead of $(a, 1) \in M$ and $a \notin M$, instead of $(a, 0) \in M$. We will use the operators $\uplus$, $\setminus$ and $\subseteq$ on multisets in their intuitive meaning.

Remark 2.3
Please note that in Chapter 6 we use the operator $\uplus$ as a union on normal sets to denote the fact that the corresponding sets are disjoint. However, it will in this case be clear from the context that we are not operating on multisets.

Definition 2.12
Given a multiset M we denote by $set(M)$ the set derived from M by deleting every instance of each element except for one, i.e.,
$set(M) = \{a \mid \exists i > 0 \in \mathbb{N} : (a, i) \in M\}$.

Definition 2.13
Given a set S we denote the **power-multiset**, i.e., the set of all multisets over S by $\wp(S)$.

2.3 Interaction Systems

Interaction systems are a model for component-based systems that was proposed and discussed in detail in [GS03], [Sif05], [GS05], [BBS06], [GGM+07b], [GGM+07a] and [MMM07a]. We start out by introducing our own (slightly generalized) notion of interaction systems in Section 2.3.1 and then introduce the original definition from [GS03] (to which we refer by $\widetilde{IS}$) in Section 2.3.2. A major motivation for the invention of *IS* [GS03] was to construct "an appropriate setting where absence of deadlock means satisfaction of strong coordination properties." The resulting framework consisted of three layers as depicted in Figure 2.2, where the different layers have the following purposes:

- The *static description layer* consists of a set of components (which are denoted by squares). The interface of a component is given by its so-called *port set* that defines the actions that are offered for synchronization. The port sets of the various components are pairwise disjoint.

- While the static description layer may be considered sufficient to integrate a component into a system we provide the *behavioral layer* to allow (respectively control) access to the component's behavior. We may deny access to a component's behavior (and provide a component as a *black box*). On the other hand, in order to modify a component's behavior or to allow for reasoning about the global behavior we may allow others to access a component's behavior (and provide it as a *white box*).

- The *interaction layer* specifies the *allowed synchronizations* between the components. A synchronization is called an *interaction* and denotes the synchronous execution of actions of different components. In Sifakis' original definition of interaction systems the formal definition of the interaction layer (that is called *interaction model*) may not be defined at will but is subject to some constraints that we have removed for the definition of the class *IS*.

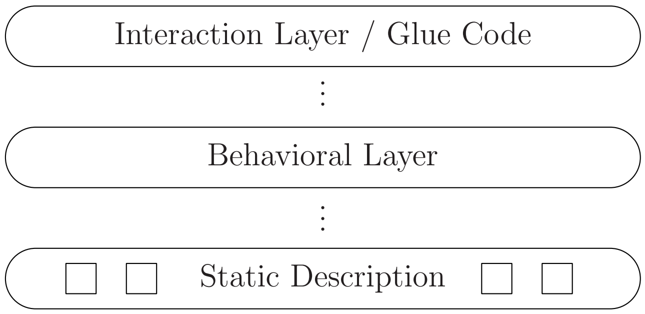


Figure 2.2: Layered system description

This layer concept provides several advantages:
Firstly, it is possible to detain information by providing business partners, e.g., with information about the interface of a (software or hardware) component but not with its local behavior. This idea basically corresponds to marking variables or functions "private" in conventional programming languages. Secondly, you can easily abstract from information, i.e., even if you have information about the local behaviors you can ignore it and thus assume another degree of abstraction. Thirdly and most importantly in our context, we can build subsystems of a system by projecting the glue-code (i.e. the interactions) to a subset of the components. As we will see in Section 6.1, this property is necessary for application of our *Cross-Checking* technique.

Many models for concurrent systems (e.g., most process algebras) abstain from synchronizations of more than two processes. The main reason for this is that allowing such multi-party-synchronizations does not extend the expressiveness, i.e., you can simulate multi-party-synchronizations by a sequence of pairwise synchronizations.

Interaction systems are similar to Lynch's *I/O automata* [CCK$^+$05, KLSV06], Henzinger's *interface automata* [AH01] and Arnold's *synchronous product of labeled transition systems* [Arn94], where Sifakis' *interactions* correspond to Arnold's *synchronization vectors*. As to Arnold's transition systems, there are some minor syntactical differences, e.g., Arnold does not require the local transition systems' label sets to be disjoint but on the other hand his synchronizations are not sets but vectors (compared to sets in interaction systems), which means that the information which action belongs to which local transition system is always available.

In Arnold's synchronization vector syntax every transition system (syntactically) occurs in every synchronization, i.e., if it does not participate, there is an additional null action e as a placeholder. However, as the resulting blow-up is only linear in the number of components. It does not have any effect on the complexity results or the techniques presented in this work.

While Arnold's synchronous products of finite transition systems are very similar to our notion of generalized interaction systems IS there is a larger gap towards Sifakis' original interaction systems $\widetilde{IS}$ which build a subclass of IS by demanding some additional constraints.

2.3.1 Generalized Interaction Systems (IS)

A ***generalized interaction system*** is a tuple $Sys = (K, \{A_i\}_{i \in K}, Int, \{T_i\}_{i \in K})$, where

- K is the set of ***components***.
  If not stated otherwise, we assume $K = \{1, \ldots, n\}$. In general we denote the number $|K|$ of components by n.

- A_i is the set of ***ports*** resp. ***actions*** of component $i \in K$.
  The ***port sets*** A_i are pairwise disjoint.

- $Int = \{\alpha_1, \ldots, \alpha_{|Int|}\}$ is the set of ***interactions***.
  An interaction α is a finite set of actions: $\alpha \subseteq \bigcup_{i \in K} A_i$.
  Each interaction α is subject to the constraint that for each component i at most one action $a_i \in A_i$ is in α. Also every action must occur in at least on interaction, i.e., $\bigcup_{i \in K} A_i = \bigcup_{\alpha \in Int} \alpha$.

- $T_i = (Q_i, A_i, \rightarrow_i, q_i^0)$ is the finite local transition system of component $i \in K$, where every state $q_i \in Q_i$ must have at least one outgoing transition. Let m denote the size of the largest local state space of a component, i.e., $m = \max_{i \in K} |Q_i|$.

An interaction $\alpha = \{a_{i_1}, \ldots, a_{i_k}\}$ with $a_{i_j} \in A_{i_j}$ describes that the components $i_1, \ldots, i_k$ cooperate via these ports.

Remark 2.4
If we want to emphasize that we are talking about the component set K of a certain interaction system we will refer to K by $K[Sys]$. However, if the correspondence is clear from the context we will just write K. (We use the analogous notation for the set Int.)

Definition 2.14
Given an interaction $\alpha \in \mathit{Int}$ and a component $i \in K$ we denote by $i(\alpha) := A_i \cap \alpha$ the ***participation*** of i in α.

Remark 2.5
We will sometimes identify a singleton set $\{a\}$ with the respective element a it contains.

Definition 2.15
For $q_i \in Q_i$ we define the set of ***enabled actions*** $ea(q_i) := \{a_i \in A_i \mid \exists q_i' \in Q_i$, s.t. $q_i \xrightarrow{a_i}_i q_i'\}$. As mentioned above, we assume that the T_i's are non-terminating, i.e., $\forall i \in K \; \forall q_i \in Q_i \; ea(q_i) \neq \emptyset$.

Definition 2.16
The ***global behavior*** $T_{Sys} = (Q, \mathit{Int}, \rightarrow_{Sys}, q^0)$ of *Sys* (henceforth also referred to as global transition system) is obtained from the behaviors of the individual components, given by the transition systems T_i, and the interactions *Int* in a straightforward manner:

- The global state space $Q = \prod_{i \in K} Q_i$ is the Cartesian product (which we consider to be order independent) of the local state spaces Q_i. We denote states by tuples $(q_1, \ldots, q_n)$ and call them global states.

- The relation $\rightarrow_{Sys} \subseteq Q \times \mathit{Int} \times Q$ is defined by
  $\forall \alpha \in \mathit{Int}\, \forall q, q' \in Q \quad q = (q_1, \ldots, q_n) \xrightarrow{\alpha}_{Sys} q' = (q'_1, \ldots, q'_n)$ iff
  $\forall i \in K \quad (q_i \overset{i(\alpha)}{\rightarrow}_i q'_i$ if $i(\alpha) \neq \emptyset$ and $q'_i = q_i$ otherwise).

- $q^0 = (q^0_1, \ldots, q^0_n)$ is the global starting state for *Sys*.

Less formally, a transition labeled by α may take place in the global transition system when each component i participating in α is ready to perform $i(\alpha)$. In this case we say that the interaction α is **enabled**.

Example 2.1
Let $Sys = (\{1,2,3\}, \{A_i\}_{1\leq i\leq 3}, \mathit{Int}, \{T_i\}_{1\leq i\leq 3})$, where $A_1 = \{a_1, b_1\}, A_2 = \{a_2, b_2\}, A_3 = \{a_3, b_3, d_3\}$, $\mathit{Int} = \{\{a_1, a_2, a_3\}, \{b_1, b_2, b_3\}, \{d_3\}, \{b_1, b_2\}\}$ and the local transition systems T_i are given in Figure 2.3.

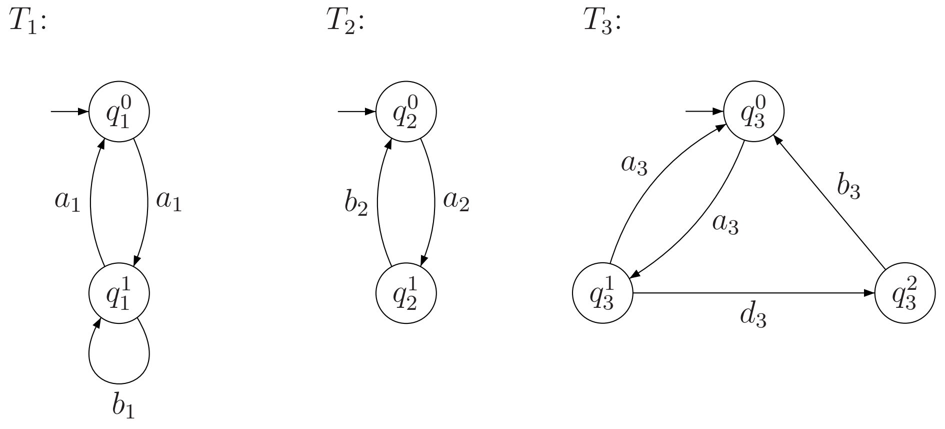


Figure 2.3: The local transition systems T_i of Example 2.1

Definition 2.17
Let the set ***IS*** consist of all possible generalized interaction systems as defined above.

For the following definitions let $Sys \in IS$ be an interaction system.

Definition 2.18
Given a global state q we say that a set of components $D \subseteq K$ is a **local deadlock** in q if every interaction in which any of the components in D could (in its present local state) participate is blocked by some other component in D.
More formally, D is a local deadlock in q if ($D \neq \emptyset$ and)
$\forall i \in D \; \forall \alpha \in Int : (ea(q_i) \cap \alpha \neq \emptyset) \Rightarrow (\exists j \in D \; j(\alpha) \nsubseteq ea(q_j))$.

Remark 2.6
Informally, a local deadlock (D in q) as defined above is a situation where a set D of components can never again participate in an interaction and where the reason for this fact lies within D itself. As the notion of local deadlock plays a central role in this thesis, we want to illustrate why Definition 2.18 coincides with this informal characterization:
Firstly, a situation where a component i can never again participate in an interaction, can only arise due to i's dependency on other components. We ensured this by our definition of interaction systems, where we demanded that every local state of a component must have at least one outgoing transition (i.e., enable at least one action) and that every action must occur in at least one interaction. Therefore, the fact that i is never again able to participate in an interaction again implies that i is restricted by some other component.
Secondly, there may be situations when a set D of components is never again able to participate in an interaction and where we still do not want to speak of a deadlock situation, because the reason does not lie within D itself. Assume a system with three components $K = \{i, j, k\}$ that may initially perform an arbitrary number of ternary synchronizations. At some point the components i and j perform a binary synchronization and proceed to local states where they can still perform binary synchronizations but henceforth exclude k. In this case $D = \{k\}$ can never again participate in an interaction but we do not want to refer to this case by the notion of deadlock.

Definition 2.19
$D \subseteq K$ is a **minimal local deadlock** in q if no proper subset of D is a local deadlock in q.

Definition 2.20
We say that a state $q \in Q$ **contains a deadlock** if a subset $D \subseteq K$ is a local deadlock in q.
We define the **predicate *DL* on global states** by $DL(q) = \mathit{true}$ if q contains a deadlock and $DL(q) = \mathit{false}$, otherwise.

Definition 2.21
We say that a system *Sys* contains a deadlock if some therein reachable state q contains a deadlock. Otherwise we say that *Sys* is **deadlock-free**.

Definition 2.22
A ***global deadlock*** is a special case of a local deadlock, where $D = K$.

Remark 2.7
Obviously in a global state q for which some set $D \subseteq K$ is a local deadlock in q, no component in D can ever again participate in every interaction. Please note that – according to Definitions 2.9, 2.21 and 2.22 – it is equivalent to say that "Sys terminates" or "Sys contains a global deadlock".

Definition 2.23
Given a trace t (or a run r) of the global transition system we say that an interaction α **occurs** in t (respectively r), if α is performed at some point in t (respectively r). We say that a component **occurs** in t (respectively r) if some interaction α with $i(\alpha) \neq \emptyset$ occurs in t (respectively r).

Definition 2.24
A component $i \in K$ **makes progress** in *Sys* if it occurs infinitely often in

every run of *Sys*.

Remark 2.8

According to Definition 2.24 a component i makes progress in Sys iff there is no reachable cycle in the behavior of Sys such that i does not occur in the cycle. More formally:
$i \in K$ makes progress iff

$$\neg(\ \exists q \in Reach(Sys)\ \exists k \in \mathbb{N}\ \exists q^1, \ldots, q^k \in Reach(Sys)\ \exists \alpha_1, \ldots, \alpha_{k+1} \in Int$$
$$q \xrightarrow{\alpha_1} q^1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_k} q^k \xrightarrow{\alpha_{k+1}} q \wedge \forall 1 \le l \le k+1\ i(\alpha_l) = \emptyset).$$

Definition 2.25

We say a component $i \in K$ ***is live*** in *Sys*, if for every reachable state q there is a way to continue performing interactions such that eventually an interaction may be performed in which i participates. I.e., i is live if
$\forall q \in Reach(Sys)\ \exists q', q'' \in Reach(Sys)\ \exists w \in Int^*\ \exists \alpha \in Int$
$q \xrightarrow{w}{}^{*} q' \ \wedge\ q' \xrightarrow{\alpha} q'' \ \wedge\ i(\alpha) \neq \emptyset$.

Definition 2.26

We say a component $i \in K$ is **available** in *Sys*, if infinitely often some interaction in which i participates is enabled in every run.

Remark 2.9

According to Definition 2.26, a component i is available in Sys iff there is no reachable cycle in the global transition system of Sys such that none of the states on the cycle enables an interaction in which i participates. This observation is analogous to Remark 2.8 so we abstain from a formalization here.

2.3.2 Original Interaction Systems ($\widetilde{IS}$)

The notion of interaction system given above is a somewhat relaxed version of the original definition that is presented in [GS03]. Here we provide the original definition and refer to this class by $\widetilde{IS}$.

When we defined our more general class *IS*, we found it convenient to straightforwardly define a set *Int* of allowed synchronizations. In the original definition from [GS03] there is however a distinction between two types of synchronizations, namely *connectors* and *complete interactions* which are subject to the following constraints:

- Let $C = \{c_1, \ldots, c_{|C|}\}$ be the ***connector set***.
  A **connector** c_k $(1 \leq k \leq |C|)$ is a finite set of actions $c_k \subseteq \bigcup_{i \in K} A_i$.
  Each connector c_k is subject to the constraint that for each component i at most one action $a_i \in A_i$ is in c_k.
  C is subject to the constraint that every action of every component occurs in at least one connector of C and no connector contains any other connector.

- Let C be a set of ***connectors***. Let $Comp = \{\alpha_1, \ldots, \alpha_{|Comp|}\}$ be the set of ***complete interactions***.
  Each $\alpha_k \in Comp$ is a subset of some $c_l \in C$.
  Also, *Comp* has to be upwards-closed w.r.t. C, i.e.:
  $\forall \alpha \in Comp\, \forall c \in C : ((\alpha \subset \alpha' \subset c) \Rightarrow \alpha' \in Comp)$.

The idea behind this distinction is that a connector provides the possibility for certain actions to communicate. In a hardware framework this can be imagined as a physical connection between several components' ports. Assuming that a connector $\{a, b, c, d\}$ implements a communication that requires active participation of a, b, c and d, we might not want to allow the synchronization as long as any of the actions a, b, c or d is not available.

However, assuming that a connector $\{a, b, c, d\}$ contains sending a piece of information (a) and receiving it (b, c, d) by three other components, we might want to specify that sending should always be allowed, but receiving is only possible if somebody sends information at the same time. So we want to include, e.g., $\{a, d\}$ (or even $\{a\}$) as a possible interaction. On the other hand, this implies that an arbitrary number of listeners should be admitted as long as the sending action occurs. So we specify all interactions α with $\{a\} \subseteq \alpha \wedge \alpha \subseteq \{a, b, c, d\}$ to be complete.

Definition 2.27
Let the set $\boldsymbol{\widetilde{IS}}$ consist of all possible original interaction systems as defined above.

Obviously, this yields (from a not too formal point of view) $\widetilde{IS} \subsetneq IS$, i.e., the abolition of these restrictions defines a strictly larger class of interaction systems. However, for some systems in *IS* it is possible to split up *Int* and give a valid definition of C and *Comp*.

Example 2.2
For the set *Int* given for Example 2.1, e.g., $C = \{\{a_1, a_2, a_3\}, \{b_1, b_2, b_3\}, \{d_3\}\}$ and $Comp = \{\{b_1, b_2\}\}$ would be sound w.r.t. Sifakis' original definition.

To obtain a more convenient notation for our reductions in Chapter 5, we define the following sets that we may interpret as decidability problems.

Definition 2.28
$\boldsymbol{Reachability} := \{(Sys, q) \mid Sys \in \widetilde{IS} \text{ and } q \in Reach(Sys)\}$.
$\boldsymbol{LDIS} := \{Sys \in \widetilde{IS} \mid \exists q \in Reach(Sys), \text{ s.t. } DL(q)\}$.
$\boldsymbol{GDIS} := \{Sys \in \widetilde{IS} \mid \exists q \in Reach(Sys), \text{ s.t. } q \not\rightarrow\}$.
$\boldsymbol{Progress} := \{(Sys, k) \mid Sys \in (\widetilde{IS} \setminus GDIS) \text{ and } k \in K[Sys] \text{ makes progress in } Sys\}$.
$\boldsymbol{Availability} := \{(Sys, k) \mid Sys \in (\widetilde{IS} \setminus GDIS) \text{ and } k \in K[Sys] \text{ is available in } Sys\}$.

Remark 2.10
The distinction between connectors and complete interactions has been made mainly for the purpose of composing larger systems from subsystems. As this

compositional feature is of little interest to us, we have abolished this distinction together with the requirement of upwards-closedness of $Comp$ *w.r.t.* C. *The main motivation for this relaxation is the decomposition of interaction systems to subsystems in Chapter 6: Keeping to the original definition* $\widetilde{IS}$ *would mean that the subsystems we build do not necessarily belong to the class* $\widetilde{IS}$ *even though the part where they contradict the definition (i.e., the upwards-closedness of* $Comp$ *w.r.t.* C*) is irrelevant for the purpose of extracting reachability information from them.*
We find it important to remark that in all mappings in this work from (resp. to) interaction systems we may always substitute $\widetilde{IS}$ *with IS (resp. IS with* $\widetilde{IS}$*), i.e., it is always possible to gain the stronger result even if this variant is not given explicitly.*

- $trans_1$*: 1SN* $\rightarrow \widetilde{IS}$ *in Section 3.1.1 already uses the smaller image space* $\widetilde{IS}$
- $trans_2$*:* $IS \rightarrow$ *1SN in Section 3.1.2 already applies to the larger range IS.*
- $f_1, \ldots, f_4$ *in Section 5.3 all map* $\widetilde{IS}$ *to* $\widetilde{IS}$. *It is clear that the functions could analogously be defined from IS to IS (but not from IS to* $\widetilde{IS}$*).*

This yields that our PSPACE-hardness results hold for the subclass $\widetilde{IS}$ *while our PSPACE-inclusion result given in Section 5.3.5 holds for the superclass IS. This is graphically represented in Figure 1.3 by the fact that the PSPACE-hardness-box and the PSPACE-inclusion-box are linked to the respective classes.*

2.4 Petri Nets

2.4.1 General Petri Nets (PN)

A ***Petri net*** [CEP93] is a four-tuple $N = (P, T, F, M_0)$ such that:

- P and T are finite disjoint sets. Their elements are called ***places*** and ***transitions***, respectively.
- $F \subseteq (P \times T) \cup (T \times P)$. F is called the ***flow relation***.
- $M_0 : P \to \mathbb{N}$ is called the ***initial marking*** of N.

Definition 2.29
For a Petri net N with a set of places P, we call a mapping $M : P \to \mathbb{N}$ a ***marking*** of N. We will also represent markings as multisets, where we write $(p, k) \in M$ (read: there are k instances of p in M) iff $M(p) = k$.

Definition 2.30
For places as well as transitions we define the notion of **preset and postset**:

For $p \in P$, *preset(p)* $:= \{t \in T \mid (t, p) \in F\}$,
postset(p) $:= \{t \in T \mid (p, t) \in F\}$.
For $t \in T$, *preset(t)* $:= \{p \in P \mid (p, t) \in F\}$,
postset(t) $:= \{p \in P \mid (t, p) \in F\}$.

For technical reasons, we only consider nets in which every node has a nonempty preset or a nonempty postset.

Definition 2.31
Let $N = (P, T, F, M_0)$ be a Petri net. A transition $t \in T$ ***is enabled under a marking*** M if $M(p) > 0$ for every place p in the preset of t. Given a transition t, we define the relation $\xrightarrow{t}$ as follows: $M \xrightarrow{t} M'$ if t is enabled under M and for all $s \in P$ we have $M'(s) = M(s) + F(t, s) - F(s, t)$, where $F(x, y)$ is 1 if $(x, y) \in F$ and 0 otherwise. We say that the transition t **is performed at** M.

Definition 2.32
The ***global behavior*** $T = (\mathcal{M}, T, \rightarrow, M_0)$ of a net (henceforth also referred to as global transition system) is obtained from the interplay of places and transitions as follows:

- $\mathcal{M}$ is the set of all markings, i.e., the mappings from P to $\mathbb{N}$. We also call the markings global states and $\mathcal{M}$ the global state space.
- The relation $\rightarrow \subseteq \mathcal{M} \times T \times \mathcal{M}$ is defined by $(M, t, M') \in\rightarrow$ iff $M \xrightarrow{t} M'$.
- $M^0 \in \mathcal{M}$ is the initial marking.

Remark 2.11
The semantics of a net as described in Definition 2.32 is often referred to as "token game semantics": For an intuitive understanding we can interpret (for a marking $M \in \mathcal{M}$, a place $p \in P$ and $k \in \mathbb{N}$) the terms $M(p) = k$ respectively $(p, k) \in M$ as the fact that in the marking M, the place p contains k so-called "tokens". A transition then shuffles tokens from one place to the other.

Example 2.3
The Petri net N_1, is given in Figure 2.4. Transitions are represented by squares and places by circles. A black dot in a place p represents the fact that p contains one token.
The reachable markings[4] of N_1 are $\{\{p_1, p_2, p_3\}, \{p_3, p_4, p_5\}, \{p_1, p_6\}\}$.

2.4.2 1-safe Nets (1SN)

While general Petri nets are an infinite model (cf. Figure 1.2) it is a natural idea to investigate subclasses that consist of finite instances. For this purpose one may define (for $k \in \mathbb{N}$) the set of *k-safe nets* that contains exactly those

[4]Remember that we identify markings with states of the global transition system for which we have defined the notion of reachability in Definition 2.5.

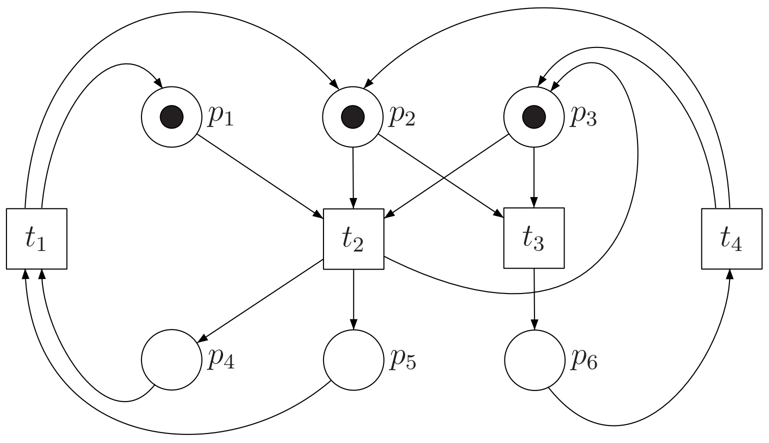


Figure 2.4: A 1-safe net N_1

Petri nets, where in every reachable marking, every place contains at most k tokens. For this thesis we define the class of 1-safe Petri nets, which are a well-studied computation model.

Definition 2.33
A marking M of a net N is called **1-safe**, if for every place p of the net $M(p) \leq 1$. A net N is called 1-safe if all its reachable markings are 1-safe.

Definition 2.34
We define the class $\mathit{1SN} \subseteq PN$ of **1-safe Petri nets** by
$\mathit{1SN} := \{N \in PN \mid \forall M \in \mathit{Reach}(N)\ M \text{ is 1-safe}\}$. For a 1-safe net $N = (\mathcal{M}, T, \rightarrow, M_0)$ we restrict $\mathcal{M}$, i.e., the global state space of N, to the set of mappings from P to $\{0, 1\}$. Thus, we can also refer to a marking M as a set, where we write $p \in M$ if $M(p) = 1$ and $p \notin M$ otherwise.

Let $N = (P, T, F, M_0) \in \mathit{1SN}$ be a 1-safe net. Then the following questions concerning N's behavior $T = (\mathcal{M}, T, \rightarrow_N, M_0)$ are known to be PSPACE-complete [CEP93].

Definition 2.35
The ***reachability*** problem for 1-safe nets consists of deciding, for a marking

M of N, whether $M_0 \rightarrow_N^* M$.

Definition 2.36
The ***liveness*** problem for 1-safe nets consists of deciding, whether every transition can always occur again. More precisely, if for every reachable marking $M \in \mathit{Reach}(N)$ and every transition $t \in T$, there is some $M' \in \mathcal{M}$ with $M \rightarrow_N^* M'$ and M' enables t.

Definition 2.37
The ***deadlock*** problem[5] for 1-safe nets consists of deciding, whether every reachable marking enables some transition. If this is the case we call the net deadlock-free.

Example 2.4
N_1, as defined in Example 2.3, is 1-safe, deadlock-free and live.

2.4.3 Communication-free Nets (CFN)

Although the class *CFN* does not occur in this thesis, we still want to mention it for the sake of related work. From the concurrency point of view the simplest Petri nets are those in which no transition has more than one ingoing arc and thus no cooperation between places is needed to fire a transition. Thus, all tokens flow through the net independently of each other. We call the class consisting of these nets *communication-free nets* (*CFN*).
The class *CFN* is equivalent to Christensen's basic parallel processes *BPP* (cf. [CHM93]) and it is very difficult to program such a net to perform any particular computation. Nevertheless, Hirshfeld (cf. [Hir94]) was able to prove trace equivalence to be undecidable for communication-free nets (cf. Figure 1.2) by applying a technique developed by Jancar (cf. [Jan94]).

[5]In analogoy to our definitions for interaction systems, we would call this deadlock *global*. This is analogous to saying that the behavior of the net *terminates*.

We mention these results, because they inspired our encodings of Minsky machines in interaction system which we present in Chapter 4.

2.5 The Linda Calculus

In this section we give a short introduction to coordination languages in general and then turn to the Linda language and the Linda calculus *LinCa*. A Coordination Language is a language specifically defined to allow two or more parties (components) to communicate for the purpose of coordinating operations to accomplish some shared (computation) goal. *Linda* seems to be the best known Coordination Language.

A *Linda* process may contain several parallel subprocesses that communicate via a so called *Tuple Space*. The *Tuple Space* is some kind of global store, where pieces of information (represented by tuples) are stored. Implementations of tuple spaces have been developed for Smalltalk, Java (JavaSpaces), Python, Ruby, TCL, Lua, Lisp, Prolog and the .NET framework (cf. [YSR09]). Ciancarini, Jensen and Yankelevich [CJY95] defined *LinCa*, the *Linda calculus*, and along with it both single-step and multi-step semantics.

In *Linda*, a tuple is a vector consisting of variables and/or constants, and there is a matching relation that is similar to data type matching in common programming languages. For the purpose of investigating the properties of the coordination through the *Tuple Space*, it is common practice to ignore the matching relation and internal propagation of tuples. Tuples are distinguished from each other by giving them unique names $(t_1, t_2, t_3, ...)$ and *LinCa* is based on a *Tuple Space* that is countably infinite. As far as the semantics for *LinCa* is concerned, the traditional interleaving point of view does not make any assumptions about the way concurrent actions are performed, i.e., for any number of processing units and independently of their speed, all possible interleavings of actions are admitted. On the other hand, the traditional multi-step point of view allows actions to be carried out concurrently or interleaved.

Apart from the standard interleaving and multi-step semantics, we are going to introduce a multi-step semantics, where we demand *maximum progress* in every transition, i.e., additional actions must be performed in the present step if possible. In other words, we consider only maximal (w.r.t. set inclusion) sets of actions for each transition. We motivate this semantics by the following example.

Example 2.5
Let us assume a system[6], in which a number of workers (processes) have to perform different jobs (calculations) on some object (tuple). In a setting with a common clock for all processes and where the workers' calculations (plus taking up the object) can always be finished within one clock cycle, we would (for maximum efficiency) want the systems semantics to represent the actual proceeding as follows: All workers are idle while the foreman supplies an object. The foreman waits while all workers read the object simultaneously and perform their jobs (by processing their respective copy of the data object) in the consecutive clock cycle. All workers put their results into the tuple space simultaneously while the foreman deletes the object, and so on.

2.5.1 LinCa

2.5.1.1 Syntax

LinCa processes:
Note that by *Tuple Space* (or TS in short), we denote the basic set from which *tuples* are chosen and by a *Tuple Space configuration*, we refer to the state of our store in the present computation, i.e., a *Tuple Space configuration* is a multiset over the *Tuple Space*. In order to show some properties of the various semantics that are introduced in the next section, we will add some designated tuples to TS. We will denote these extra tuples by c, d, e and we will write TS_{cde} for $TS \cup \{c, d, e\}$, where $TS \cap \{c, d, e\} = \emptyset$.

[6]We will define this system more formally later in this chapter (cf. Example 2.6).

Definition 2.38

Given a fixed *Tuple Space* TS, we can define the set of **LinCa processes** $LinCa_{TS}$ as the set of processes derived from the grammar given in Figure 2.5, where every time we apply one of the rules $P := in(t).P$, $P := out(t).P$, $P := rd(t).P$ or $P :=!\ in(t).P$, t is substituted by an element of the *Tuple Space*. $in(t), out(t)$ and $rd(t)$ are called actions. If $t \in \{c, d, e\}$ then they are called *internal* actions, else *observable* actions. Trailing zeros (.0) will be dropped in examples.

$$\boxed{\mathrm{P} := 0 \;\Big|\; \mathrm{in(t).P} \;\Big|\; \mathrm{out(t).P} \;\Big|\; \mathrm{rd(t).P} \;\Big|\; \mathrm{P} \mid \mathrm{P} \;\Big|\; !\ \mathrm{in(t).P}}$$

Figure 2.5: LinCa

Remark 2.12

*The rule $P :=\ !\ in(t).P$ will be interpreted as the possibility to run an arbitrary number of parallel processes $in(t).P$. We refer to the !-operator by **replication**. As a counter-part to recursion, replication is used to feature infinite behavior (i.e., in a process calculus that uses neither of the concepts, no process can diverge, in terms of our notions defined in Section 2.2). Please note that in LinCa, replication is **in-guarded**, i.e., in order to invoke another parallel process P by replication, we will have to perform a deleting read operation on a tuple t of the tuple space.*
This fact is essential for our multi-step semantics: With only a finite number of tuples in the tuple space at every point of time, we will never be able to replicate an infinite number of parallel processes in one step.

Definition 2.39

Let P be a *LinCa*-process. Then $ea(P)$ denotes the multiset of enabled actions of P, which are defined in Figure 2.6. We define a decomposition of (the

1) $ea(0) = \emptyset$
2) $ea(in(t).P) = \{in(t)\}$
3) $ea(out(t).P) = \{out(t)\}$
4) $ea(rd(t).P) = \{rd(t)\}$
5) $ea(!\, in(t).P) = \{(in(t), \infty)\}$
6) $ea(P \mid Q) = ea(P) \uplus ea(Q)$

Figure 2.6: The set of enabled actions $ea(P)$ of a process $P \in LinCa$

$ea_{IN}(P) = \{(t,i) \mid (in(t),i) \in ea(P)\}$
$ea_{OUT}(P)$ analogously
$ea_{RD}(P)$ analogously

Figure 2.7: The sets $ea_{IN}(P), ea_{OUT}(P), ea_{RD}(P)$ of a process $P \in LinCa$

tuples occuring in) $ea(P)$ into three subsets $ea_{IN}(P), ea_{OUT}(P), ea_{RD}(P)$ as given in Figure 2.7. The notions $(in(t), \infty) \in ea(P)$ and $(t, \infty) \in ea_{IN}(P)$ describe the fact, that infinitely many actions $in(t)$ are *enabled* in P. These notions will only be used for *enabled actions*, never for *Tuple Space* configurations, because (due to the in-guardedness of replication, cf. Remark 2.12) all computed *Tuple Space* configurations remain finite.

In the following, we define three different semantics for LinCa processes. As for the hitherto introduced models, we will use a labeled transition system for this purpose, where for either semantics we will present a different definition for the transition relation. In these transition systems, states are pairs $< P, M >$ of *LinCa*-processes and *Tuple Space* configurations and transition labels are triples (I, O, R) of (possibly empty) multisets of tuples, where I represents the performed *in*-actions, O the performed *out*-actions and R the performed *rd*-actions. We write τ instead of (I, O, R) iff $I, O, R \in \wp(\{c, d, e\})$ and call τ an *internal* label and a transition $q \xrightarrow{\tau} q'$ an *internal* transition. A label $a = (I, O, R) \neq \tau$ is called an *observable* label and a transition $q \xrightarrow{a} q'$ is called an *observable* transition.

Definition 2.40
Let *SEM* $\in$ {*ITS, MTS, MTS-mp*}, denote the *interleaving*, the *multistep* and the *multistep with maximum progress* semantics for LinCa. The ***global behavior*** $SEM[P] = (Q, L, \rightarrow_{SEM}, q^0)$ of a LinCa process P under *SEM* is obtained from the parallel processes operating on their shared storage, the tuple space, as follows.

- $Q = LinCa_{TS} \times \wp(TS)$. We also call these pairs global states and the set of all such pairs the global state space.
- $L = \wp(TS) \times \wp(TS) \times \wp(TS)$.
- The relations $\rightarrow_{SEM}$ are definied in the following section.
- $q^0 =< P, \emptyset >$

2.5.1.2 Semantics

In this section, we introduce the ITS-semantics for *LinCa* based on the semantics given in [BGLZ05b] and a MTS-semantics that we consider the natural extension of ITS. In the given MTS-semantics, we allow (in contrast to [CJY95]) an arbitrarily large number of *rd*-actions to be performed simultaneously on a single instance of a tuple.
To describe the various semantics, we split the semantic description into two parts: A set of rules for *potential* transitions of *LinCa*-processes (Figures 2.8 and 2.10) and an additional rule to establish the semantics in which we check if some *potential* transition is allowed under the present *Tuple Space* configuration (Figures 2.9, 2.11 and 2.13), respectively. This allows us to reuse the rules in Figure 2.8 (henceforth called *pure syntax* rules) for the succeeding *MTS* and *MTS-mp* semantics. Choosing this representation makes it convenient to point out common features and differences of the discussed semantics.

$$\boxed{\begin{array}{l}
1)\ in(t).P \overset{(\{t\},\emptyset,\emptyset)}{\rightarrow} P \\
2)\ out(t).P \overset{(\emptyset,\{t\},\emptyset)}{\rightarrow} P \\
3)\ rd(t).P \overset{(\emptyset,\emptyset,\{t\})}{\rightarrow} P \\
4)\ !\ in(t).P \overset{(\{t\},\emptyset,\emptyset)}{\rightarrow} P \mid !\ in(t).P \\
5)\ \dfrac{P \overset{(I,O,R)}{\rightarrow} P'}{P \mid Q \overset{(I,O,R)}{\rightarrow} P' \mid Q}
\end{array}}$$

Figure 2.8: ITS: pure syntax (symmetrical rule for 5 omitted)

$$\boxed{\dfrac{P \overset{(I,O,R)}{\rightarrow} P' \in \mathit{ITS\text{-}Rules} \quad I \subseteq M \quad R \subseteq M}{<P,M> \overset{(I,O,R)}{\rightarrow}_{ITS} <P',(M \setminus I) \uplus O>}}$$

Figure 2.9: ITS

In contrast to [BGLZ05b] we label transitions. We have to do so to record which actions a step-transition performs in order to check if this is possible under the present *Tuple Space* configuration. The labels serve as a link between the rules of *pure syntax* and the semantic rule: For a *potential* transition $P \overset{(I,O,R)}{\rightarrow} P'$ the multisets I, O and R contain the tuples on which we want to perform *in*, *out*, respectively *rd* actions. In MTS (see Figure 2.11), such a *potential* transition is only valid for some *Tuple Space* configuration M, if $I \uplus set(R) \subseteq M$, i.e., M includes enough instances of each tuple to satisfy all performed *in*-actions and at least one additional instance for the performed *rd*-actions on that tuple (if any *rd*-actions are performed). For *out*-actions there is no such restriction.

In Figure 2.13 we use the notion of *maximality* of a *potential* transition for some *Tuple Space* configuration M. *Maximality* is given iff conditions 1) and 2) in Figure 2.12 hold, where the first condition means, that all enabled out-actions have to be performed, whereas the second condition means, that as many of the *in* and *rd*-actions as possible have to be performed. More precisely 2.1) represents the case, that the number of instances of some tuple

ITS-Rules 1) - 5) (from Figure 2.8)

$$6) \quad !\, in(t).P \overset{(\{(t,i)\},\emptyset,\emptyset)}{\rightarrow} \prod_i P \mid !\, in(t).P$$

$$7) \quad \frac{P \overset{(I_P,O_P,R_P)}{\rightarrow} P' \quad Q \overset{(I_Q,O_Q,R_Q)}{\rightarrow} Q'}{P \mid Q \overset{(I_P \uplus I_Q, O_P \uplus O_Q, R_P \uplus R_Q)}{\rightarrow} P' \mid Q'}$$

Figure 2.10: MTS: pure syntax

$$\frac{P \overset{(I,O,R)}{\rightarrow} P' \in \textit{MTS-Rules} \quad (I \uplus Set(R)) \subseteq M}{<P,M> \overset{(I,O,R)}{\rightarrow}_{MTS} <P',(M \backslash I) \uplus O>}$$

Figure 2.11: MTS

t in the present *Tuple Space* configuration M exceeds the number of enabled in-actions on that tuple. In this case all in-actions and all rd-actions have to be performed. We define the relations $\rightarrow_{ITS}$, $\rightarrow_{MTS}$ and $\rightarrow_{MTS\text{-}mp}$ as the smallest relations satisfying the corresponding rule in Figure 2.9, 2.11 and 2.13.

Example 2.6
We end this section by formally modeling Example 2.5 (p. 40). A foreman supplies a group of workers with jobs.
Let $P := foreman \mid worker_1 \mid ... \mid worker_n$, where:

foreman $=$ *out(object).wait.in(object).foreman*
$worker_i$ $=$ *rd(object).out*$(result_i)$*.*$worker_i$

(Please note that *wait*-operator is used for ease of notation only, it is not part of the discussed language but can easily be simulated.)
Ciancarini's original MTS semantics would allow P to evolve in a variety of ways. However, given a common clock and given that all workers can per-

$$\boxed{\begin{array}{l}
\quad 1)\ (t,i) \in ea_{OUT}(P) \Rightarrow (t,i) \in O \\
\wedge\ 2)\ (t,i) \in M \wedge (t,j) \in ea_{IN}(P) \wedge (t,k) \in ea_{RD}(P) \Rightarrow \\
\qquad (\quad 2.1)\ j < i \wedge (t,j) \in I \wedge (t,k) \in R \\
\qquad\ \vee\ 2.2)\ j \geq i \wedge (t,i) \in I \wedge (t,0) \in R \\
\qquad\ \vee\ 2.3)\ j \geq i \wedge (t,i-1) \in I \wedge (t,k) \in R \wedge k \geq 1\ )
\end{array}}$$

Figure 2.12: Condition for *Maximality* of a transition $P \overset{(I,O,R)}{\rightarrow} P'$ for some *Tuple Space* configuration M

$$\boxed{\frac{P \overset{(I,O,R)}{\rightarrow} P' \in \mathit{MTS\text{-}Rules} \quad P \overset{(I,O,R)}{\rightarrow} P' \text{ is maximal for } M}{<P,M> \overset{(I,O,R)}{\rightarrow}_{\mathit{MTS\text{-}mp}} <P',(M \backslash I) \uplus O>}}$$

Figure 2.13: MTS-mp

form their rd-operations (as well as their internal calculation which we abstract from in LinCa) within one clock cycle, the expected/desired maximum-progress behavior of P (that has already been described in the introduction) corresponds to the (in this case deterministic) behavior of $MTS\text{-}mp[P]$.

2.6 Minsky Machines

A Minsky machine (or random access machine) $\hat{M}$ [SS63] consists of m registers, that may store arbitrarily large natural numbers and a program (i.e., sequence of n enumerated instructions) of the form:

$$\begin{array}{c} I_1 \\ I_2 \\ \vdots \\ I_n \end{array}$$

Each instruction I_i can either be a successor instruction that increments a register and continues with the next instruction or a decrease/jump instruction that decreases a register or jumps to a designated instruction if the value of the register is already zero. We denote each instruction (for $1 \leq j \leq m$, $s \in \mathbb{N}$) by

a) $i : Succ(r_j)$, respectively

b) $i : DecJump(r_j, s)$

Finally, a tuple $c^0 =< v_1, v_2, ..., v_m, k >\in \mathbb{N}^{m+1}$ serves as the starting configuration of $\hat{M}$.

Definition 2.41

The ***global behavior*** $T = (Q, \mathcal{I}, \rightarrow, c^0)$ of a Minsky machine $\hat{M}$ represents the deterministic behavior as follows.

- Q is the set of all configurations for $\hat{M}$, where a configuration of $\hat{M}$ is represented by a tuple $< v_1, v_2, ..., v_m, k >\in \mathbb{N}^{m+1}$, where v_i represents the value stored in register r_i and k is the number of the instruction that is to be computed next.
  We also call these tuples global states and the set of all such tuples the global state space.
- The label set $\mathcal{I} = \{I_1, \ldots, I_n\}$ is given by the instructions of $\hat{M}$.
- Let $c =< v_1, v_2, ..., v_m, k >$ be the present configuration of $\hat{M}$. Then we distinguish the following three cases to describe the possible transitions:

  1) $k > n$ means that $\hat{M}$ halts, because the instruction that should be computed next does not exist. This happens after computing instruction I_n and passing on to I_{n+1} or by simply jumping to a nonexistent instruction.
  2) if $k \in \{1, ..., n\} \wedge I_k = Succ(r_j)$ then v_j and k are incremented, i.e., we increment the value in register r_j and proceed with the next instruction.
  3) if $k \in \{1, ..., n\} \wedge I_k = DecJump(r_j, s)$ then $\hat{M}$ checks whether the

value v_j of r_j is > 0. If this is the case, we decrement it and proceed with the next instruction (i.e., we increment k). Else (i.e., if $v_j = 0$) we simply jump to instruction I_s, (i.e., we assign $k := s$).

For the cases 2) and 3) let $(c, I_k, c') \in \rightarrow$, where c' is the configuration that we receive if we perform the described modifications on c.

We say a Minsky machine $\hat{M}$ with starting configuration $< v_1, v_2, ..., v_m, k >$ terminates if its computation reaches a configuration that belongs to case 1). If such a configuration is never reached, the computation never stops and we say that $\hat{M}$ diverges. This coincides with our notions of termination and divergence presented in Section 2.2. Due to the determinism in a Minsky machine's behavior, it will always either terminate or diverge.
It is well-known [Min67] that the question whether a Minsky machine diverges or terminates under the starting configuration $< 0, ..., 0, 1 >$ is undecidable for the class MM of all Minsky machines.

2.7 Equivalences

As already described in the introduction, the main purpose of formal methods is to formally describe a system on a mathematic level in order to reason about its properties. Equivalences are a very important notion in this matter: We often want to decide whether two systems (which might be different abstractions of the same implementation) behave "the same way" on a certain level of detail. Depending on the level of abstraction, the aspects one wants to focus on and last but not least the aim one is pursuing one may define a variety of different equivalences. In this work, we consider well-known equivalences, namely bisimilarity, trace equivalence and isomorphism as well as equivalences that we define for our own specific purposes, namely weak step simulation and a somewhat relaxed notion of isomorphism.

2.7.1 Bisimilarity

Let $T_i = (Q_i, Lab, \rightarrow_i, q_i^0)$, $i \in \{1, 2\}$ be two labeled transition systems using the same label set.

A relation $R_B \subseteq Q_1 \times Q_2$ is a **bisimulation** if $\forall (q_1, q_2) \in R_B$ the following two conditions hold:

1) $q_1 \xrightarrow{l}_1 q_1' \quad \Rightarrow \quad \exists q_2' \in Q_2 \; q_2 \xrightarrow{l}_2 q_2' \wedge (q_1', q_2') \in R_B$.

2) $q_2 \xrightarrow{l}_2 q_2' \quad \Rightarrow \quad \exists q_1' \in Q_1 \; q_1 \xrightarrow{l}_1 q_1' \wedge (q_1', q_2') \in R_B$.

We call two transition systems T_1 and T_2 **bisimilar** (denoted by $T_1 \sim T_2$) iff there exists a bisimulation R_B with $(q_1^0, q_2^0) \in R_B$.

2.7.2 Trace Equivalence

Let $T_i = (Q_i, Lab, \rightarrow_i, q_i^0)$, $i \in \{1, 2\}$ be two labeled transition systems using the same label set.

We call T_1 and T_2 trace equivalent iff $\text{Traces}(T_1) = \text{Traces}(T_2)$.

2.7.3 Weak Step Simulation

Let $T_1 = (Q_1, Lab_1, \rightarrow_1, q_1^0)$ and $T_2 = (Q_2, Lab_2, \rightarrow_2, q_2^0)$ be two labeled transition systems. We write $T_1 \preceq T_2$ iff the following properties hold:

1) T_1 and T_2 are trace equivalent.

2) Sys_2 is able to weakly simulate Sys_1, i.e., $\exists R \subseteq Q_1 \times Q_2$ such that:

  2.1) $(q_1^0, q_2^0) \in R$ and

  2.2) $(q_1, q_2) \in R \wedge q_1 \xrightarrow{a} q_1' \Rightarrow \exists q_2' \in Q_2 : q_2 \xrightarrow{a}^{+} q_2' \wedge (q_1', q_2') \in R$.

2.7.4 Isomorphism

We define the notion of isomorphism, which we use to establish a relation between transition systems that use different label sets Lab_1 and Lab_2.

Let $T_i = (Q_i, Lab_i, \rightarrow_i, q_i^0)$, $i \in \{1, 2\}$ be two labeled transition systems. We say that T_1 and T_2 are **isomorphic** ($T_1 \cong T_2$) iff there exist bijective functions $f : Q_1 \rightarrow Q_2$ and $g : Lab_1 \rightarrow Lab_2$, such that $f(q_1^0) = q_2^0$ and $\forall q_1, q_1' \in Q_1, l_1 \in Lab_1, q_2, q_2' \in Q_2, l_2 \in Lab_2$ the following two conditions hold:
1) $q_1 \xrightarrow{l_1}_1 q_1' \quad \Rightarrow \quad f(q_1) \xrightarrow{g(l_1)}_2 f(q_1')$.
2) $q_2 \xrightarrow{l_2}_2 q_2' \quad \Rightarrow \quad f^{-1}(q_2) \xrightarrow{g^{-1}(l_1)}_1 f^{-1}(q_2')$.

2.7.5 Isomorphism up to a Label Relation R

We define a modified notion of isomorphism, which we use to establish a relation between transition systems that use different label sets Lab_1 and Lab_2 if the transition systems are not isomorphic to each other. R then defines which labels in Lab_1 we want to correspond to which labels in Lab_2.

Let $T_i = (Q_i, Lab_i, \rightarrow_i, q_i^0)$, $i \in \{1, 2\}$ be two labeled transition systems. Given a *label relation* $R \subseteq (Lab_1 \times Lab_2)$, that relates labels of Lab_1 to labels of Lab_2, we say that T_1 and T_2 are **isomorphic up to *R*** ($T_1 \cong_R T_2$) iff there exists a bijective function $f : Q_1 \rightarrow Q_2$, such that $f(q_1^0) = q_2^0$ and $\forall q_1, q_1' \in Q_1, l_1 \in Lab_1, q_2, q_2' \in Q_2, l_2 \in Lab_2$ the following two conditions hold:
1) $q_1 \xrightarrow{l_1}_1 q_1' \quad \Rightarrow \quad \exists l_2 \in Lab_2$, s.t. $(l_1, l_2) \in R \ \wedge \ f(q_1) \xrightarrow{l_2}_2 f(q_1')$.
2) $q_2 \xrightarrow{l_2}_2 q_2' \quad \Rightarrow \quad \exists l_1 \in Lab_1$, s.t. $(l_1, l_2) \in R \ \wedge \ f^{-1}(q_2) \xrightarrow{l_1}_1 f^{-1}(q_2')$.

Remark 2.13
As already mentioned in Remark 2.1, we will, for ease of notation, often identify a system with its behavior. Hence, we will call two systems equivalent (in any of the above defined meanings) iff the labeled transition systems that are induced by their semantics are equivalent.

Chapter 3

Mappings

In this chapter, we present mappings between some of the models for concurrency that we introduced in Chapter 2. We do so in order to establish a relation between the models (respectively the semantics) and to compare them with respect to expressiveness.

We start out with the contribution that has the greatest impact on our studies, namely a translation from 1-safe nets to interaction systems. This translation is computable in polynomial time (and as a consequence does not yield an exponential blow-up) and it preserves reachability. Therefore, it allows us to deduce from the PSPACE-hardness of reachability in 1-safe nets (cf. [CEP93]) that reachability in interaction systems is also PSPACE-hard to decide.

Then we present a translation for the opposite direction, i.e., from interaction systems to 1-safe nets. This translation assures that the respective global transition systems of an interaction system and its corresponding net are isomorphic up to a label relation. However, the translation may yield an exponential blow-up. This blow-up is unavoidable to assure the desired relation, as will be seen in the consecutive sections, where we consider other translations from interaction systems to 1-safe nets. We show that there is no translation that yields bisimilarity. We also show that isomorphism up to a label relation cannot be acquired with a polynomial mapping.

In the second part of this chapter, we discuss the relation between $\mathit{LinCa}_{\mathit{ITS}}$, respectively $\mathit{LinCa}_{\mathit{MTS}}$ and the semantics $\mathit{LinCa}_{\mathit{MTS\text{-}mp}}$ that requires *maximum progress*. We give a transformation[1] from $\mathit{LinCa}_{\mathit{ITS}}$ to $\mathit{LinCa}_{\mathit{MTS\text{-}mp}}$ and from $\mathit{LinCa}_{\mathit{MTS}}$ to $\mathit{LinCa}_{\mathit{MTS\text{-}mp}}$, respectively, such that a process can be *weakly step simulated* by its image process, i.e., they are trace equivalent and the *MTS-mp* semantics is able to weakly simulate the other semantics.

3.1 Interaction Systems and 1-safe Nets

In this section we establish a relation between the model of interaction systems and the well-studied model of 1-safe Petri nets for which complexity results have been investigated in [CEP93]. We show that anything described by a 1-safe net can be described by an interaction system without a blow-up in notation. Similarly, interaction systems can be translated into 1-safe nets. However, it is unavoidable to have a (worst case) exponential blow-up for this translation.

The results with the greatest impact are PSPACE-hardness of the problems of reachability and liveness for interaction systems. These are the first PSPACE-hardness results concerning interaction systems and they partially[2] outrun the complexity results given in [Min07]. The established results provide an essential basis for Section 5.3, where we use these "master-reductions"

[1]We find it convenient to refer to these mappings as transformations (rather than translations) because we do not map from one syntax to another.

[2]In [Min07] we prove NP-hardness of local and global deadlock. While our translation from *1SN* to *IS* yields PSPACE-hardness for global deadlock, it does not outrun the result given for local deadlocks.

to extend the PSPACE-hardness results (by polynomial reductions) to almost all behavioral questions for interaction systems.

Furthermore, these results suggest that there is no polynomial time algorithm for solving the questions of reachability or liveness in interaction systems. Thus, they provide further motivation for approaches to establish algorithms that test properties of interaction systems in polynomial time by avoiding state space explosion. In Chapter 6 we will see that the model of interaction systems is particularly suited for applying these approaches because they exploit local information about components, whose identities are preserved when being composed. Petri nets, by contrast, lack compositionality and the identity of a component is lost when a composite system is modeled as a Petri net.

3.1.1 Translating 1-safe Nets to Interaction Systems

In this section we define a mapping $\mathit{trans}_1 : \mathit{1SN} \rightarrow \widetilde{IS}$ from 1-safe nets to interaction systems in such a way that a 1-safe net N and its image $\mathit{trans}_1(N)$ are isomorphic.

Idea and Explanation:
Let $N = (P, T, F, M_0)$ be a 1-safe net. We introduce a component $\hat{p}$ for each place $p \in P$. The transition system $T_{\hat{p}}$ has only two states, one state $q^f_{\hat{p}}$ to reflect the fact that p is full (i.e., it contains a token) and one state $q^e_{\hat{p}}$ to reflect that it is empty (i.e., it does not contain a token). For the events represented by the arcs in N we introduce actions in order to refer to them in $\mathit{trans}_1(N)$. For the transitions t adjacent to p we distinguish three cases:

a) $t \in (\mathit{preset}(p) \setminus \mathit{postset}(p))$. When such a transition is performed in N, this means that p is empty before the performance of t and contains a

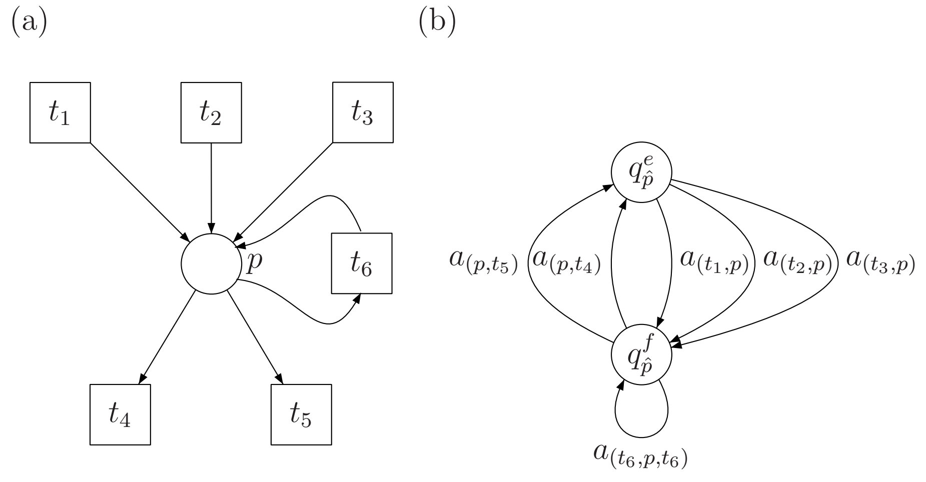


Figure 3.1: A place with ingoing and outgoing transitions and its corresponding component

token afterwards. Thus, we introduce an edge from $q^e_{\hat{p}}$ to $q^f_{\hat{p}}$ labeled by the action $a_{(t,p)}$.

b) $t \in (\mathit{postset}(p) \setminus \mathit{preset}(p))$. When such a transition is performed in N, this means that p contains a token before the performance of t and is empty afterwards. Thus, we introduce an edge from $q^f_{\hat{p}}$ to $q^e_{\hat{p}}$ labeled by the action $a_{(p,t)}$.

c) $t \in (\mathit{preset}(p) \cap \mathit{postset}(p))$. This means there has to be a token in p to perform t which will still be contained afterwards. In this case, we introduce a loop at $q^f_{\hat{p}}$ labeled by the action $a_{(t,p,t)}$.

For an example of a place with pre- and postset and its corresponding component, see Figure 3.1 (a) resp. (b). (Note that only transitions adjacent to p are depicted.) Now we define a connector $c(t)$ for each transition t. For the places adjacent to t we do again distinguish three cases:

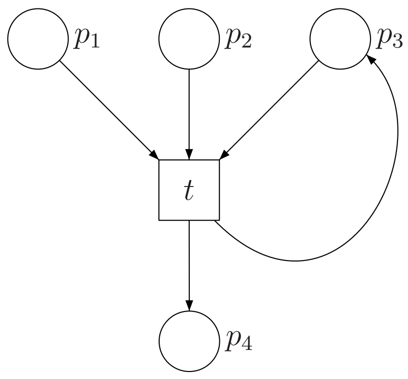
a) $p \in (\mathit{preset}(t) \setminus \mathit{postset}(t))$. This means that in order to perform t, there has to be a token in p, and there will be no token in p after performing t. Thus, we include the action $a_{(p,t)}$ in $c(t)$ which already occurs in the component $\hat{p}$ in such a way that this fact is perfectly reflected.

b) $p \in (\mathit{postset}(t) \setminus \mathit{preset}(t))$. This means that in order to perform t, there must not to be a token in p, and there will be a token in p after performing t. Thus, we include the action $a_{(t,p)}$ in $c(t)$ which already occurs in the component $\hat{p}$ in the corresponding way.

c) $p \in (\mathit{preset}(t) \cap \mathit{postset}(t))$. This means that in order to perform t, there has to be a token in p, and p will still contain one token after performing t. Thus, we include the action $a_{(t,p,t)}$ in $c(t)$ which already occurs in the component $\hat{p}$ in the corresponding way.

For an example of a transition with pre- and postset, respectively its corresponding connector, see Figure 3.2[3] (a) respectively (b).
Finally, let $C := \{c(t) \mid t \in T\}$ and $\mathit{Comp} := \emptyset$.

(a)



(b)

$$c(t) = \{a_{(p_1,t)}, a_{(p_2,t)}, a_{(t,p_3,t)}, a_{(t,p_4)}\}$$

Figure 3.2: A transition with its pre- and postset and its corresponding connector

[3]Note that only places adjacent to t are depicted.

Formal definition:

Let $N \in \mathit{1SN}$, then $\mathit{trans}_1(N) = \{K, \{A_i\}_{i \in K}, C, \mathit{Comp}, \{T_i\}_{i \in K}\}$, where

$$\boldsymbol{K} \quad := \quad \{\hat{p} \mid p \in P\}$$

$$\begin{aligned}
\text{For } \hat{p} \in K : A^{in}_{\hat{p}} \quad &:= \quad \{a_{(t,p)} \mid t \in (\mathit{preset}(p) \setminus \mathit{postset}(p))\}, \\
A^{out}_{\hat{p}} \quad &:= \quad \{a_{(p,t)} \mid t \in (\mathit{postset}(p) \setminus \mathit{preset}(p))\}, \\
A^{inout}_{\hat{p}} \quad &:= \quad \{a_{(t,p,t)} \mid t \in (\mathit{preset}(p) \cap \mathit{postset}(p))\}, \text{ and} \\
\boldsymbol{A_{\hat{p}}} \quad &:= \quad A^{in}_{\hat{p}} \cup A^{out}_{\hat{p}} \cup A^{inout}_{\hat{p}}.
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{T_{\hat{p}}} \quad &:= \quad (\{q^e_{\hat{p}}, q^f_{\hat{p}}\}, A_{\hat{p}}, \rightarrow_{\hat{p}}, q^0_{\hat{p}}), \text{ where } A_{\hat{p}} \text{ has already been given,} \\
\rightarrow_{\hat{p}} \quad &:= \quad \{(q^e_{\hat{p}}, a_{(t,p)}, q^f_{\hat{p}}) \mid a_{(t,p)} \in A^{in}_{\hat{p}}\} \\
&\cup \quad \{(q^f_{\hat{p}}, a_{(p,t)}, q^e_{\hat{p}}) \mid a_{(p,t)} \in A^{out}_{\hat{p}}\} \\
&\cup \quad \{(q^f_{\hat{p}}, a_{(t,p,t)}, q^f_{\hat{p}}) \mid a_{(t,p,t)} \in A^{inout}_{\hat{p}}\} \\
q^0_{\hat{p}} \quad &:= \quad q^e_{\hat{p}} \text{ if } M_0(p) = 0 \text{ and } q^0_{\hat{p}} := q^f_{\hat{p}} \text{ if } M_0(p) = 1.
\end{aligned}$$

In order to define a connector for a transition in a natural way we now relate the actions in $\bigcup_{i \in K} A_i$ to the transitions in the way described informally above.

$$\begin{aligned}
\text{For } t \in T\text{: } A^{in}_t \quad &:= \{a_{(p,t)} \mid p \in (\mathit{preset}(t) \setminus \mathit{postset}(t))\}, \\
A^{out}_t \quad &:= \{a_{(t,p)} \mid p \in (\mathit{postset}(t) \setminus \mathit{pretset}(t))\}, \\
A^{inout}_t \quad &:= \{a_{(t,p,t)} \mid p \in (\mathit{preset}(t) \cap \mathit{postset}(t))\} \\
c(t) \quad &:= A^{in}_t \cup A^{out}_t \cup A^{inout}_t \\
\boldsymbol{C} \quad &:= \{c(t) \mid t \in T\} \\
\boldsymbol{Comp} \quad &:= \emptyset
\end{aligned}$$

It remains to prove that C is indeed a sound connector set.

We observe that $\{A^{sup}_t \mid t \in T, \mathit{sup} \in \{\mathit{in}, \mathit{out}, \mathit{inout}\}\}$ is a disjoint decomposition of $\bigcup_{i \in K} A_i$. This is due to the fact that the sets A^{sup}_t are defined

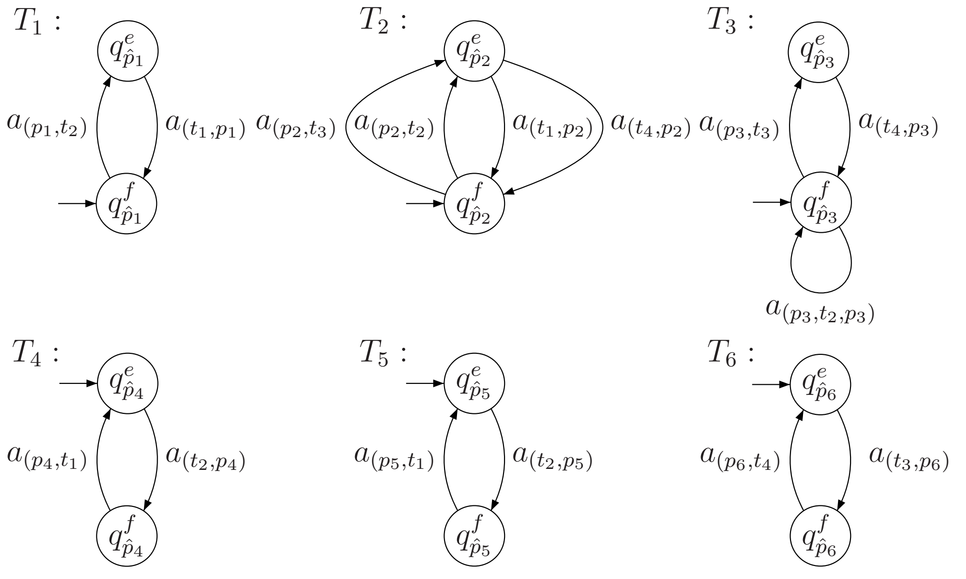


Figure 3.3: The local transition systems T_i for $\mathit{trans}_1(N)$

inversely to the sets $A^{sup}_{\hat{p}}$. Thus, C, which is obtained from that disjoint decomposition by taking the union of some of the subsets, is still a disjoint decomposition of $\bigcup_{i\in K} A_i$. Thereby, we may conclude that all connectors consist of actions in $\bigcup_{i\in K} A_i$ and each action occurs exactly once, i.e., in at least one connector, and no connector can be a subset of another connector. Also, as $\mathit{Comp} = \emptyset$, we have upwards-closedness of Comp w.r.t. C.

Example 3.1
Let $N = (P, T, F, M_0)$ be the 1-safe net from Example 2.3 (p. 36). The corresponding interaction system is $\mathit{trans}_1(N)$ =$\{\{1,\ldots,6\}$, $\{A_i\}_{1\le i\le 6}$, C, $\emptyset$, $\{T_i\}_{1\le i\le 6}\}$, where $C = \{\{a_{(p_4,t_1)}, a_{(p_5,t_1)}, a_{(t_1,p_1)}, a_{(t_1,p_2)}\}, \{a_{(p_1,t_2)}, a_{(p_2,t_2)}, a_{(t_2,p_4)}, a_{(t_2,p_5)}, a_{(p_3,t_2,p_3)}\}, \{a_{(p_2,t_3)}, a_{(p_3,t_3)}, a_{(t_3,p_6)}\}, \{a_{(p_6,t_4)}, a_{(t_4,p_3)}, a_{(t_4,p_2)}\}\}$, and the local transition systems T_i (and implicitly the port sets A_i) are given in Figure 3.3.

Theorem 3.1

Let $N \in$ 1SN and $Sys = trans_1(N)$.

We define $f : \mathcal{M} \to Q$ by $f(M) = q$ with $q(\hat{p}) = q^f_{\hat{p}}$ if $p \in M$ and $q(\hat{p}) = q^e_{\hat{p}}$ otherwise.

Further let $g : T \to C$ be defined by $g(t) := c(t)$ as given above.

Then, $N \cong Sys$, i.e., N is isomorphic to Sys.

Proof: Clear by construction of $trans_1$.

Due to Theorem 3.1 we can clearly answer the question of reachability of a marking M in a 1-safe net by computing $trans_1(N)$ in polynomial time and answering whether $f(M) \in Reach(trans_1(N))$. So we deduce, from the PSPACE-hardness of reachability in *1SN* (cf. [CEP93]) PSPACE-hardness of reachability in $\widetilde{IS}$ as stated in the following corollary.

Corollary 3.1

The Reachability problem for (original) interaction systems is PSPACE-hard.

We also know that the question of liveness for 1-safe nets, i.e., the question whether every transition can always occur again in the behavior of a net N, is PSPACE-hard [CEP93].

As liveness in 1-safe nets concerns transitions (cf. Definition 2.36, p. 38), which are translated to interactions, and, by contrast, liveness in interaction systems concerns components (cf. Definition 2.25, p. 31), we introduce a place p_t for each transition t, such that the component corresponding to the place, i.e., $\hat{p}_t$ will be live iff t can always occur again. This can be done by employing a (polynomial) preencoding map_{pre} on N before applying $trans_1$. More formally, let $map_{pre}(N) = (P \cup \{p_t \mid t \in T\}, T, F \cup \{(p_t, t), (t, p_t) \mid t \in T\}, M_0 \cup \{p_t \mid t \in T\})$.

Now N is live iff every $\hat{p}_t$ $(t \in T)$ is live in $trans_1(map_{pre}(N))$. As a consequence, we deduce PSPACE-hardness of liveness in (original) interaction systems as stated in the following corollary.

Corollary 3.2
The Liveness problem for (original) interaction systems is PSPACE-hard.

3.1.2 Translating Interaction Systems to 1-safe Nets

In this section we define a mapping $\mathit{trans}_2 : IS \rightarrow \mathit{1SN}$ from generalized interaction systems to 1-safe Petri nets in such a way that an interaction system *Sys* and its image $\mathit{trans}_2(Sys)$ are isomorphic up to a label relation R_L.

Idea and Explanation:
In this section, we present the encoding from interaction systems to 1-safe nets. Our interest in such a translation is mainly based on the theoretical point of view, i.e., we want to obtain a more profound understanding of the capabilities of these two models. Still, as interaction systems are a relatively young model for which so far not many tools have been developed, there is some practical benefit: One could translate a system into a net and apply Petri net tools in order to investigate behavioral questions of the system.
Let $Sys = (K, \{A_i\}_{i \in K}, Int, \{T_i\}_{i \in K}) \in IS$ be an interaction system. We introduce a place $\hat{q}_i$ for each local state $q_i \in Q_i$ of each component $i \in K$. A global state of *Sys* is a tuple that consists of the current local states of the components. Thus, for every reachable marking in N, there will for each $i \in K$ always be exactly one place $\hat{q}_i$ that contains a token. A token in $\hat{q}_i$ reflects the fact that q_i is the current state of component i.
It remains to translate the glue code given by the interactions in *Int* to the notion of transition in Petri nets. An action $a_i \in A_i$ may occur multiple times

in the local transition system T_i of component i. Thus, the performance of an interaction α may cause different state changes in Sys.

As a consequence, we are going to map an interaction α not to a single transition but to a set of transitions $T(\alpha)$. Each transition in $T(\alpha)$ represents one of these possible global state changes and will shift the tokens in N according to the local state changes that are caused for the components that participate in α.

Formal definition:

$\mathit{trans}_2(\mathit{Sys}) = (P, T, F, M_0)$, where

$\boldsymbol{P} = \bigcup_{i \in K} \{\hat{q}_i \mid q_i \in Q_i\}$.

For $\alpha = \{a_{i_1}, a_{i_2}, \ldots, a_{i_k}\} \in \mathit{Int}$, we introduce a set of transitions

$T(\alpha) := \{\{(q_{i_1}, a_{i_1}, q'_{i_1}), \ldots, (q_{i_k}, a_{i_k}, q'_{i_k})\} \mid \forall 1 \leq j \leq k (q_{i_j}, a_{i_j}, q'_{i_j}) \in \rightarrow_{i_j}\}$.

Then we define

$\boldsymbol{T} = \bigcup_{\alpha \in \mathit{Int}} T(\alpha)$.

For each α and each transition $t = \{(q_{i_1}, a_{i_1}, q'_{i_1}), \ldots, (q_{i_k}, a_{i_k}, q'_{i_k})\}$ in $T(\alpha)$ we introduce arcs as follows:

$$\begin{aligned} F(t) = &\quad \{(\hat{q}_{i_1}, t), \ldots, (\hat{q}_{i_k}, t)\} \cup \{(t, \hat{q}'_{i_1}), \ldots, (t, \hat{q}'_{i_k})\} \\ F(\alpha) = &\quad \textstyle\bigcup_{t \in T(\alpha)} F(t). \end{aligned}$$

$\boldsymbol{F} = \bigcup_{\alpha \in \mathit{Int}} F(\alpha)$.

$\boldsymbol{M}_0 = \{\hat{q}_i \in P \mid q_i = q_i^0\}$.

This means that in the initial marking exactly those places that correspond to the local starting states of the components contain a token.

Remark: Let T_i be the local transition system of component i and let $a_i \in A_i$ be an action of i. We denote the number of ***occurrences*** of a_i in T_i by $occ(a_i)$. Note that for one interaction $\alpha = \{a_{i_1}, \ldots, a_{i_k}\}$ there are $occ(a_{i_1}) \cdot \ldots \cdot occ(a_{i_k})$ instances of α. This means we might have exponentially (in n) many instances for a single interaction α, which will result in

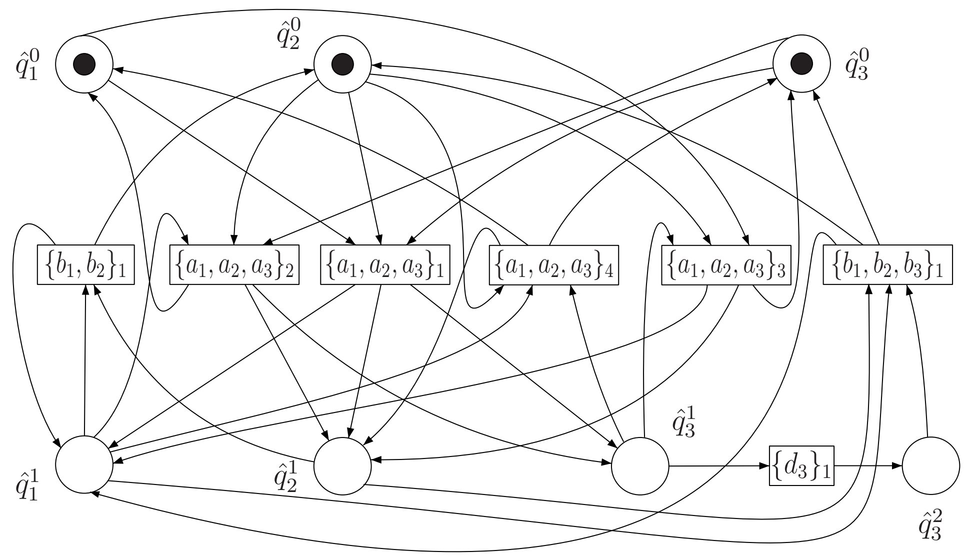


Figure 3.4: The corresponding 1-safe Petri-net $\mathit{trans}_2(\mathit{Sys})$

an exponential blow-up in our mapping from interaction systems to 1-safe nets. See, e.g., Example 2.1 (p. 28), where we get $\mathit{occ}(a_1) \cdot \mathit{occ}(a_2) \cdot \mathit{occ}(a_3)$ $= 2 \cdot 1 \cdot 2 = 4$ transitions for the interaction $\{a_1, a_2, a_3\}$ in $T(\{a_1, a_2, a_3\})$.

Example 3.2

Let Sys be the interaction system from Example 2.1 (p. 28). The corresponding net $\mathit{trans}_2(\mathit{Sys})$ is given in Figure 3.4. For better readability we use the following abbreviations:

$\{a_1, a_2, a_3\}_1 := \{(q_1^0, a_1, q_1^1), (q_2^0, a_2, q_2^1), (q_3^0, a_3, q_3^1)\}$,
$\{a_1, a_2, a_3\}_2 := \{(q_1^0, a_1, q_1^1), (q_2^0, a_2, q_2^1), (q_3^1, a_3, q_3^0)\}$,
$\{a_1, a_2, a_3\}_3 := \{(q_1^1, a_1, q_1^0), (q_2^0, a_2, q_2^1), (q_3^0, a_3, q_3^1)\}$,
$\{a_1, a_2, a_3\}_4 := \{(q_1^1, a_1, q_1^0), (q_2^0, a_2, q_2^1), (q_3^1, a_3, q_3^0)\}$.
$\{b_1, b_2, b_3\}_1 := \{(q_1^1, b_1, q_1^1), (q_2^1, b_2, q_2^0), (q_3^2, b_3, q_3^0)\}$.
$\{d_3\}_1 := \{(q_3^1, d_3, q_3^2)\}$ and $\{b_1, b_2\}_1 := \{(q_1^1, b_1, q_1^1), (q_2^1, b_2, q_2^0)\}$.

Theorem 3.2

Let $Sys \in IS$ and $N = trans_2(Sys)$.

We define $f : Q \to \mathcal{M}$ by $f(q_1, \ldots, q_n) = \{\hat{q}_1, \ldots, \hat{q}_n\}$.

Further let $R_L = \bigcup_{\alpha \in Int}(\{\alpha\} \times T(\alpha))$.

Then, $Sys \cong_{R_L} N$, i.e., Sys is isomorphic up to R_L to N.

Proof: Clear by construction of *trans*$_2$.

3.1.3 Considering other Relations between IS and 1SN

In Sections 3.1.1 and 3.1.2, we discussed mappings between interaction systems and 1-safe nets. Particularly, the relation we established by the translation *trans*$_2$: *IS* $\to$ *1SN* in the previous section, and that yielded isomorphism up to a 1-to-many relabeling was rather tailored to our purpose. This makes it a legitimate question to ask what other relations could be conceived, especially taking into account that our translation included an exponential blow-up. We will in the following answer the questions:

- As we allow for an exponential blow-up anyway, would it not be possible to give a translation from *IS* to *1SN* that establishes bisimilarity or even an isomorphism between the respective global transition systems?

- Would it not be possible to give a polynomial translation that yields the relation that is established here (bisimilarity or even isomorphism up to a 1-to-many relabeling)?

The answer to either question is "no" as we will show in the following subsections.

3.1.3.1 The Gap between Interaction Systems and 1-Safe Nets

There is no translation from *IS* to *1SN* that yields bisimilarity between the respective global behaviors. The reason for this fact becomes obvious when we consider the following simple example of an interaction system:

Example 3.3
Let $Sys = (K, \{A_i\}_{i\in K}, C, Comp, \{T_i\}_{i\in K})$, where
$K = \{1,2\}, C = \{\alpha_1, \alpha_2\}$, $\alpha_1 = \{a_1, a_2\}$, $\alpha_2 = \{b_1, b_2\}$, $Comp = \emptyset$ and the local behaviors of 1 and 2 are given in Figure 3.5.

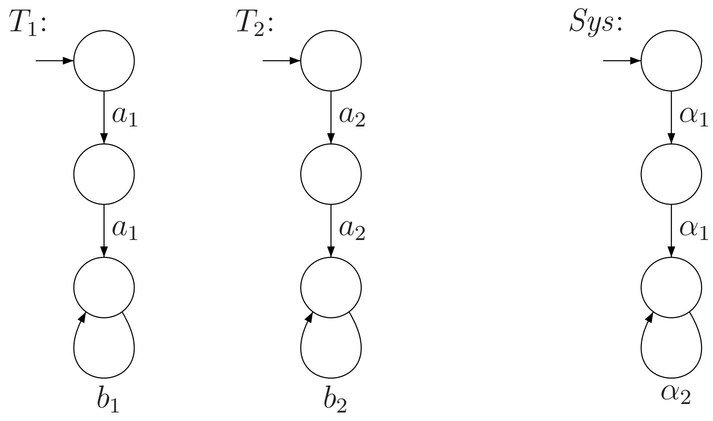


Figure 3.5: The local transition systems T_i and the global behavior *Sys* of Example 3.3

The global behavior of this system is given in Figure 3.5. A bisimilar 1-safe net must have two transitions (α_1 and α_2) and α_1 must be performable exactly twice from the net's initial marking. In 1-safe nets, however, a transition t that can be executed two times successively can be executed arbitrarily often successively, because the preset of t must be included in the postset of t in order to allow for successive execution.

3.1.3.2 Considering a polynomial Version of $trans_2$

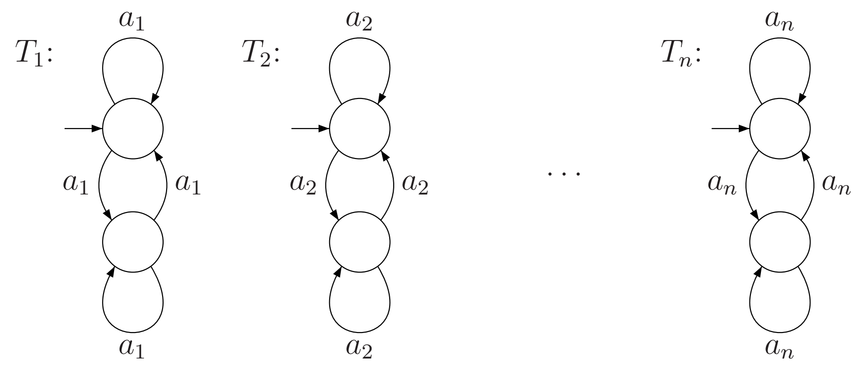
With $trans_2$ we gave an exponential translation from *IS* to *1SN*. One might ask whether it is possible to give a polynomial translation that yields the relation that is established by $trans_2$. To prove that this is indeed impossible we show that there is generally no polynomial translation that yields an isomorphism for the unlabeled versions of the respective global behaviors. Let us consider the following parametrized example of an interaction system.

Example 3.4
Let $Sys_n = (K_n, \{A_i\}_{i \in K_n}, C_n, Comp_n, \{T_i\}_{i \in K_n})$, where:
$K_n = \{1, \ldots, n\}, C_n = \{\{a_1, \ldots, a_n\}\}$, $Comp_n = \emptyset$ and the local behaviors T_i are given in Figure 3.6.

The underlying unlabeled graph of the global behavior of this system is called (in graph theory terminology) K_{2^n}. It has 2^n states and 2^{2n} edges (cf. Figure 3.6). Let N be a 1-safe net whose unlabeled global behavior is isomorphic to K_{2^n}. For any marking M there must be 2^n different transitions enabled as for each node there are 2^n outgoing edges with pairwise different successor states. However, the existence of 2^n different transitions clearly implies that N can not be polynomial in n.

3.2 The various Semantics of the Linda Calculus

In Section 2.5 we defined different types of semantics for the Linda Calculus, namely *ITS*, *MTS* and *MTS-mp*. Here, we discuss the relationships between the semantics by taking a closer look at the respective behavior they imply for a LinCa process P. In order to do so, we establish transformations that augment a *LinCa* process by certain auxiliary actions that allow us to sim-



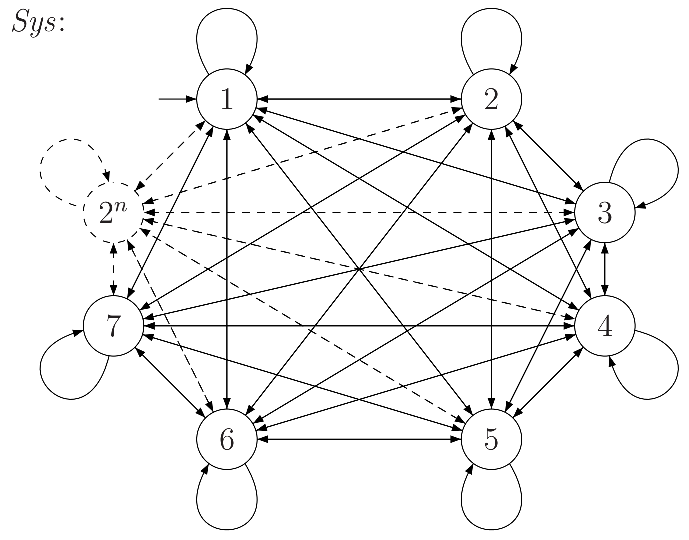


Figure 3.6: The local transition systems T_i and the global behavior *Sys* of Example 3.4

ulate one semantics by another. We express our considerations in terms of trace equivalence and weak step simulation (cf. Section 2.7).

To begin with, we formulate the following straight-forward observations that concern the subgraph-relations between the global transition systems.

- *ITS*[*P*] is always a subgraph of *MTS*[*P*], as the *pure syntax* rules for *ITS* (cf. Figure 2.8) are a subset of those for *MTS* (cf. Figure 2.10) and the way the semantics (cf. Figures 2.9 and 2.11) build on the *pure syntax* rules is the same.

- *MTS-mp*$[P]$ is always a subgraph of *MTS*$[P]$, as the *pure syntax* rules for *MTS* and *MTS-mp* are the same but for the *MTS-mp* semantics (cf. Figure 2.13) we apply a stronger precondition than for the *MTS* semantics (cf. Figure 2.11).

By $LinCa_{cde}$ we denote the *LinCa* language based on an extended *Tuple Space*. That is, we assume the existence of three designated tuples *c,d,e* that are not elements of the original *LinCa Tuple Space*. We extend our *MTS-mp* semantics to treat actions on these tuples just like any other actions in the purely syntactic description. However, in transition systems whenever (I, O, R) consists of nothing but designated tuples we replace it by τ, the *internal* label. Whenever some *internal* actions are performed concurrently with some *observable* actions, the label of the resulting transition will simply consist of the *observable* ones.

By *MTS-mp*$[P]$ where $P \in LinCa_{cde}$ we denote the semantics of P as described above.

3.2.1 Translating $LinCa_{ITS}$ to $LinCa_{MTS\text{-}mp}$

In this subsection we define an encoding $trans_3$: $LinCa \to LinCa_{cde}$ and prove that $ITS[P] \preceq MTS\text{-}mp[trans_3(P)]$ holds. The presented encoding artificially sequentializes processes by initially writing one instance of the designated tuple c into the tuple space and surrounds every original action with an enclosing pair $in(c)$ and $out(c)$. Thus, at most one action is enabled at every point of time.

The formal definition of $trans_3$ consists of a recursive definition enc_{ITS}, finally composed with the prefix $out(c)$:

$enc_{ITS}(0) = 0$

$enc_{ITS}(\text{act(t).P}) = \text{in(c).act(t).out(c).}enc(\text{P})$

$enc_{ITS}(\text{P} \mid \text{Q}) = enc(\text{P}) \mid enc(\text{Q})$

$enc_{ITS}(!\, in(t).P) = !\, in(c).in(t).out(c).enc(P)$

$trans_3(P) = out(c).enc_{ITS}(P)$

Theorem 3.3

$ITS[P] \preceq MTS\text{-}mp[trans_3(P)]$

Proof:

1) *Weak Similarity*

$enc_{ITS}(P)$ puts a prefix $in(c)$ in front of and a suffix $out(c)$ behind every action in P. The weak step simulation deterministically starts by performing the *internal* action $out(c)$ and subsequently simulates every step of the ITS Transition System by performing three steps as follows:
First, we remove the guarding $in(c)$-prefix which is produced by the encoding from the *observable* action we want to simulate (henceforth, we call this *unlocking an action*). Then we perform this action. Finally, we perform the suffix $out(c)$ to supply the *Tuple Space* configuration with the tuple c for the simulation of the next action. As all described steps are indeed maximal, the transitions are valid for *MTS-mp*.

2) *Equality of Traces*

Traces$(ITS[P]) \subseteq$ *Traces*$(MTS\text{-}mp[trans_3(P)])$ follows immediately from weak similarity. As for the reverse inclusion: $MTS\text{-}mp[trans_3(P)]$ can either unlock an action that can be performed under the present *Tuple Space* configuration. In this case, $ITS[P]$ can perform the same action directly. $MTS\text{-}mp[trans_3(P)]$ could also unlock an action that is blocked under the present *Tuple Space* configuration. In this case the computation (and thus the trace) halts due to the total blocking of the process $trans_3(P)$ (as the single instance of tuple c has been consumed without leaving an opportunity to provide a new one). However, the hitherto observed trace also exists in $ITS[P]$.

3.2.2 Translating $LinCa_{MTS}$ to $LinCa_{MTS\text{-}mp}$

In this subsection, we define an encoding $trans_4$: $LinCa \to LinCa_{cde}$ and prove that $MTS[P] \preceq MTS\text{-}mp[trans_4(P)]$ holds. The presented encoding tweaks a given process P in such a way, that the maximum progress requirement of the *MTS-mp* semantics is discarded. For this purpose, we prefix every original action of P with the action $in(c)$ and rely on an additional process Q to produce an arbitrary number of instances of the tuple c. As a result, any number k of actions can be performed in parallel by producing exactly k instances of c first.

The formal definition of $trans_4$ consists of a recursive definition enc_{MTS}, composed with Q and an initial $out(c)$:

Firstly, we introduce the recursive encoding enc_{MTS}: $LinCa \to LinCa_{cde}$, that simply prefixes every action of a process with an additional blocking $in(c)$ action.

$enc_{MTS}(0) = 0$
$enc_{MTS}(act(t).P) = in(c).act(t).enc_{MTS}(P)$
$enc_{MTS}(P \mid Q) = enc_{MTS}(P) \mid enc_{MTS}(Q)$
$enc_{MTS}(!\, in(t).P) = !\, in(c).in(t).enc_{MTS}(P)$

Secondly, we introduce the process Q. All actions performed by Q are *internal* actions, and Q will be able to produce an arbitrary number of instances of the tuple c simultaneously.

$$\text{We define: } Q := \quad !\, in(d).[rd(e).out(c) \mid out(d)] \\ \mid \quad !\, in(d).out(e).wait.in(e).wait.out(d)$$

Remark 3.1

Strictly speaking, the wait-operator used in Q is not included in LinCa. We nevertheless use it because a wait-action (which has no effect on the rest of the process and is not observable*) can be implemented by a rd-action in the following way: Let t^* be a designated tuple that is not used for other purposes. If P is a LinCa-process except for the fact, that it may contain some wait-actions, then we consider it as the process $P[wait/rd(t^*)] \mid out(t^*)$. Additionally, we want to point out that the wait-actions are not essential for the correctness of the encoding $trans_4$. They only keep things synchronized for ease of proofs and understanding.*

Finally, we define $trans_4(P) := enc_{MTS}(P) \mid Q \mid out(d)$. The parallel process $out(d)$ puts a tuple d into the initially empty *Tuple Space* configuration to activate the process Q.

Theorem 3.4

$MTS[P] \preceq MTS\text{-}mp[trans_4(P)]$

Proof:

1) *Weak similarity*

The proof is similar to the one of Theorem 3.3. Whenever we want to simulate some step $< P, M > \overset{(I,O,R)}{\rightarrow}_{MTS} < P', M' >$ (where $|I| + |O| + |R| = z$) Q first produces z processes $rd(e).out(c)$ by subsequently performing z times $in(d)$ and $out(d)$ in line 1 of Q. Then, line 2 of Q is performed, i.e., the tuple e is provided and then simultaneously read by the z $rd(e).out(c)$-processes (and deleted by $in(e)$ immediately afterwards). This causes the simultaneous production of z instances of c, which are used to unlock the desired actions in

$enc_{MTS}(P)$ in the subsequent step. As the step we want to simulate is valid in MTS and as all other actions (besides the second *internal wait*-action of Q that is in fact performed simultaneously) are still blocked by their prefixes $in(c)$, the step is also *maximal* and thus it is valid in *MTS-mp*.

2) *Equality of Traces*

Again, *Traces*$(MTS[P]) \subseteq$ *Traces(MTS−mp*$[trans_4(P)]$) follows immediately from weak similarity. We give a sketch of the proof of the reverse inclusion:

The process Q finds itself in a "loop" in which it continuously produces arbitrary numbers of instances of the tuple c (let the number of produced c-tuples be z). In the subsequent step (due to our maximality-request) as many actions $in(c)$ as possible are performed. The actual number of these *unlockings* is restricted either by the number of enabled $in(c)$ processes (let this number be x, i.e., $(c, x) \in ea_{IN}(enc_{MTS}(P)))$ if $x \leq z$ or by the number of instances of c that we have produced if $x > z$.

In the next step, we perform as many unlocked actions as possible. That might be all of them, if the present *Tuple Space* configuration M allows for it, or a subset of them. In any of those cases, the same set of actions can instantly be performed in $MTS[P]$. It simply remains to show that neither the over-production of c-tuples, nor the unlocking of more actions than we can simultaneously perform under M will ever enable any *observable* actions that are not already enabled in $MTS[P]$.

The proof for this proposition is straightforward and can be done by defining a relation S that includes all pairs $(< P, M >, < trans_4(P), M \uplus \{d\} >)$ as well as any pair $(< P, M >, q')$ where q' is a derivation from $< trans_4(P), M \uplus \{d\} >$ by τ-steps. Based on S, we can show, that whenever $(q_1, q_2) \in S$ and q_2 performs an *observable* step in *MTS-mp*$[trans_4(P)]$, q_1 will be ready to imitate it in $MTS[P]$. For an example that displays this analogy between P under $LinCa_{MTS}$ and $trans_4(P)$ under $LinCa_{MTS\text{-}mp}$ see Appendix A.1.

Chapter 4

Undecidability

In this chapter, we present undecidability results for *LinCa* under the *MTS-mp* semantics. We derive these results from the class *MM* of Minsky machines, for which the problem of termination is known to be undecidable [Min67]. For this purpose, we present property-preserving translations from *MM* to $LinCa_{MTS-mp}$. In contrast to the mappings presented in Chapter 3, the ones presented here are to a much smaller degree generic and rather result-oriented. In other words, we do not aim at a close relation between states of the systems' behaviors and their respective outgoing transitions. Instead, we confine with preserving the respective property of interest.

4.1 Overview

The expressive power of the Linda calculus and its dialects has been thoroughly discussed by Bravetti et al. [BGZ00, BGLZ05a]. In standard *LinCa*, termination is decidable [BGLZ05a] by defining an encoding of *LinCa* systems into finite Petri nets that preserves the existence of a finite computation and by exploiting the fact that the deadlock problem is decidable [Reu90] in finite Petri nets.

On the other hand, *PrioLinCa* (a dialect of *LinCa* that features priorization) can be proven to be Turing complete by a Minsky machine encoding, even with the use of only two types of priorities. As a consequence, termination and divergence are undecidable for *PrioLinCa*.

There exist *LinCa*-dialects that include a predicative *in*-operator $inp(t)?P\_Q$, that has the semantical meaning "*if* $t \in$ *TS then P else Q*" (cf. [BGM00]). It has been shown [BGZ00] that the questions of termination and divergence are undecidable for such dialects, as for any Minsky machine there is an obvious deterministic encoding (i.e., these dialects, like *PrioLinCa*, are even Turing complete).

However, the original *Linda calculus* [CJY95] that we discuss here does not include such an operator which makes the proof that neither termination nor divergence are decidable under the *MTS-mp* semantics more difficult. Given a Minsky machine $\hat{M}$, we will assign to $\hat{M}$ a *LinCa* process P. While the behavior of $\hat{M}$ is deterministic, the *MTS-mp*-behavior of P features nondeterminism. That is, in contrast to the transition system for $\hat{M}$, the transition system for P contains branchings. While one of the paths in the behavior of P imitates the behavior of $\hat{M}$, there will be other paths that compute something "useless". The trick is to prevent useless computations from messing up the result of the Minsky machine imitation. We will define encodings *term* and *div* that map *Minsky machine*s to *LinCa*-processes such that a *Minsky machine* $\hat{M}$ terminates (respectively diverges) iff the corresponding transition system $MTS\text{-}mp[term(\hat{M})]$ (respectively $MTS\text{-}mp[div(\hat{M})]$) terminates (respectively diverges).

Remark 4.1
Whenever a process P (respectively its behavior) performs a nondeterministic choice, there will be one transition describing the simulation of $\hat{M}$ and one

transition that will compute something useless. For ease of explanations in Sections 4.2 and 4.3, we call the first type right *and the second type* wrong*.*

To guarantee that the part of the transition system that is reached by a *wrong* transition (that deviates from the simulation) does not affect the question of termination (respectively divergence), we will make sure that all traces of the corresponding subtree are infinite (respectively finite). This approach guarantees that the whole transition system terminates (respectively diverges) iff the single computation sequence that we obtain by keeping to the *right* transitions is finite (respectively infinite).
Our encodings establish a natural correspondence between *Minsky machine* configurations and *Tuple Space* configurations, i.e., the *Minsky machine*-configuration $< v_1, v_2, ..., v_m, k >$ belongs to the *Tuple Space* configuration $\{(r_1, v_1), ..., (r_m, v_m), p_k\}$. For a *Minsky machine* configuration c we refer to the corresponding *Tuple Space* configuration by $TS(c)$.

4.2 Termination is undecidable in MTS-mp-LinCa

Let *term*: $MM \rightarrow LinCa$ be the following mapping:

$$term(\hat{M}) = \prod_{i\in\{1,...,n\}} [I_i] \mid !\, in(div).out(div) \mid in(loop).out(div) \mid out(p_1)$$

where the encoding $[I_i]$ of a *Minsky machine*-Instruction in *LinCa* is:

$$\begin{array}{rcl} [i : Succ(r_j)] & = & !\, in(p_i).out(r_j).out(p_{i+1}) \\ [i : DecJump(r_j, s)] & = & !\, in(p_i).[\ out(loop) \mid in(r_j).in(loop).out(p_{i+1})\ ] \\ & \mid & !\, in(p_i).[\ in(r_j).out(loop) \\ & & \qquad \mid wait.wait.out(r_j).in(loop).out(p_s)\ ] \end{array}$$

Note that the first (deterministic) step of $term(\hat{M})$ will be the initial $out(p_1)$.

The resulting *Tuple Space* configuration is $\{p_1\} = TS(<0, ..., 0, 1>)$. For ease of notation, we will henceforth also denote the above defined process where $out(p_1)$ has already been executed by $term(\hat{M})$.

Theorem 4.1
For every Minsky machine $\hat{M}$ the transition system MTS-mp$[term(\hat{M})]$ terminates iff $\hat{M}$ terminates under starting configuration $<0, ..., 0, 1>$.

Proof: To prove Theorem 4.1 we describe (given some *Minsky machine* $\hat{M}$ and configuration c) the possible transition sequences from some state $<term(\hat{M}), TS(c)>$ in *MTS-mp*$[term(\hat{M})]$ by three cases:
In cases 1 and 2, the computation in our transition system is completely deterministic and performs the calculation of $\hat{M}$. In case 3, the transition sequence that simulates *DecJump(r_j,s)* includes nondeterministic choice. As already mentioned, performing only *right* choices (cases 3.1.1 and 4.1.1) results in an exact simulation of $\hat{M}$'s transition $c \rightarrow_{\hat{M}} c'$, i.e., the transition sequence leads to the corresponding state $<term(\hat{M}), TS(c')>$. Performing at least one *wrong* choice (cases 3.1.2, 3.2, 4.1.2 and 4.2) causes the subprocess $!\ in(div).out(div)$ to be activated, thus assuring that any computation in the corresponding subtree diverges (denoted by $\rightsquigarrow$). In this case, other subprocesses are not of concern because they cannot interfere by removing the tuple *div*, so we substitute these subprocesses by "...".

1. $k > n$, i.e., $\hat{M}$ has terminated. Then $<term(\hat{M}), TS(c)>$ is totally blocked.

2. $k \in \{1, ..., n\} \wedge I_k = k : Succ(r_j)$, then $\hat{M}$ increments both r_j and k. The corresponding transition sequence in *MTS-mp*$[term(\hat{M})]$ is:

$$\begin{array}{rl} & <term(\hat{M}), TS(c)> \\ \rightarrow & <term(\hat{M}) \mid out(r_j).out(p_{k+1}), TS(c) \setminus \{p_k\}> \\ \rightarrow & <term(\hat{M}) \mid out(p_{k+1}), TS(c) \setminus \{p_k\} \uplus \{r_j\}> \\ \rightarrow & <term(\hat{M}), TS(c) \setminus \{p_k\} \uplus \{r_j, p_{k+1}\}> \\ = & <term(\hat{M}), TS(c')> \end{array}$$

3. $k \in \{1, ..., n\} \wedge I_k = k : DecJump(r_j, s) \wedge v_j \neq 0$, then $\hat{M}$ decrements r_j and increments k. The possible transition sequences in *MTS-mp*$[term(\hat{M})]$ are: $<term(\hat{M}), TS(c)> \overset{nondet.}{\rightarrow}$

3.1 ***right***:

$$\begin{array}{rl} & <term(\hat{M}) \mid out(loop) \mid in(r_j).in(loop).out(p_{k+1}), TS(c) \setminus \{p_k\}> \\ \rightarrow & <term(\hat{M}) \mid in(loop).out(p_{k+1}), TS(c) \setminus \{p_k, r_j\} \uplus \{loop\}> \overset{nondet.}{\rightarrow} \end{array}$$

3.1.1 ***right - right***:

$$\begin{array}{rl} & <term(\hat{M}) \mid out(p_{k+1}), TS(c) \setminus \{p_k, r_j\}> \\ \rightarrow & <term(\hat{M}), TS(c) \setminus \{p_k, r_j\} \uplus \{p_{k+1}\}> \\ = & <term(\hat{M}), TS(c')> \end{array}$$

3.1.2 ***right - wrong***:

$$\begin{array}{rl} & <term(\hat{M}) \mid in(loop).out(p_{k+1}), TS(c) \setminus \{p_k, r_j\} \uplus \{loop\}> \\ \rightarrow & <... \mid out(div), TS(c) \setminus \{p_k, r_j\}> \rightsquigarrow \end{array}$$

3.2 ***wrong***:

$$\begin{array}{rl} & <term(\hat{M}) \mid in(r_j).out(loop) \mid \\ & \quad wait.wait.out(r_j).in(loop).out(p_s), TS(c) \setminus \{p_k\}> \\ \rightarrow & <term(\hat{M}) \mid out(loop) \mid wait.out(r_j).in(loop).out(p_s), TS(c) \setminus \{p_k, r_j\}> \\ \rightarrow & <term(\hat{M}) \mid out(r_j).in(loop).out(p_s), TS(c) \setminus \{p_k, r_j\} \uplus \{loop\}> \\ \rightarrow & <... \mid out(div), TS(c) \setminus \{p_k\}> \rightsquigarrow \end{array}$$

4. $k \in \{1, ..., n\} \wedge I_k = k : DecJump(r_j, s) \wedge v_j = 0$,

   then $\hat{M}$ assigns $k := s$

   $<term(\hat{M}), TS(c)> \overset{nondet.}{\rightarrow}$

4.1 ***right***:

$\rightarrow <term(\hat{M}) \mid in(r_j).out(loop) \mid$

$wait.wait.out(r_j).in(loop).out(p_s), TS(c) \setminus \{p_k\}>$

$\rightarrow <term(\hat{M}) \mid in(r_j).out(loop) \mid wait.out(r_j).in(loop).out(p_s), TS(c) \setminus \{p_k\}>$

$\rightarrow <term(\hat{M}) \mid in(r_j).out(loop) \mid out(r_j).in(loop).out(p_s), TS(c) \setminus \{p_k\}>$

$\rightarrow <term(\hat{M}) \mid in(r_j).out(loop) \mid in(loop).out(p_s), TS(c) \setminus \{p_k\} \uplus \{r_j\}>$

$\rightarrow <term(\hat{M}) \mid out(loop) \mid in(loop).out(p_s), TS(c) \setminus \{p_k\}>$

$\rightarrow <term(\hat{M}) \mid in(loop).out(p_s), TS(c) \setminus \{p_k\} \uplus \{loop\}> \overset{nondet.}{\rightarrow}$

4.1.1 ***right - right***:

$$\begin{aligned} & <term(\hat{M}) \mid out(p_s), TS(c) \setminus \{p_k\}> \\ \rightarrow\ & <term(\hat{M}), TS(c) \setminus \{p_k\} \uplus \{p_s\}> \\ =\ & <term(\hat{M}), TS(c')> \end{aligned}$$

4.1.2 ***right - wrong***:

$<... \mid out(div), TS(c) \setminus \{p_k\}> \rightsquigarrow$

4.2 ***wrong***:

$$\begin{aligned} & <term(\hat{M}) \mid out(loop) \mid in(r_j).in(loop).out(p_{k+1}), TS(c) \setminus \{p_k\}> \\ \rightarrow\ & <term(\hat{M}) \mid in(r_j).in(loop).out(p_{k+1}), TS(c) \setminus \{p_k\} \uplus \{loop\}> \\ \rightarrow\ & <... \mid out(div), TS(c) \setminus \{p_k\}> \rightsquigarrow \end{aligned}$$

4.3 Divergence is undecidable in MTS-mp-LinCa

Let *div*: $MM \rightarrow LinCa$ be the following mapping:

$$div(\hat{M}) = \prod_{i \in \{1,...,n\}} [I_i] \mid in(flow) \mid out(p_1)$$

where the encoding $[I_i]$ of a *Minsky machine* instruction in *LinCa* is:

$$\begin{array}{rcl} [i : Succ(r_j)] & = & !\, in(p_i).out(r_j).out(p_{i+1}) \\ [i : DecJump(r_j, s)] & = & \quad !\, in(p_i).in(r_j).out(p_{i+1}) \\ & & \mid \ !\, in(p_i).\ [\, in(r_j).out(flow) \\ & & \qquad\qquad \mid wait.wait.out(r_j).in(flow).out(p_s)\,] \end{array}$$

Note that the first (deterministic) step of $div(\hat{M})$ will be the initial $out(p_1)$. The resulting *Tuple Space* configuration is $\{p_1\} = TS(< 0, ..., 0, 1 >)$. For ease of notation, we will henceforth also denote the above defined process where $out(p_1)$ has already been executed by $div(\hat{M})$.

Theorem 4.2
For every Minsky machine $\hat{M}$ *the transition system MTS-mp[div(*$\hat{M}$*)] diverges iff* $\hat{M}$ *diverges under starting configuration* $< 0, ..., 0, 1 >$.

Proof: To prove Theorem 4.2 we describe (given some *Minsky machine* $\hat{M}$ and configuration c) the possible transition sequences from some state $< div(\hat{M}), TS(c) >$ in *MTS-mp*$[div(\hat{M})]$. In cases 1 and 2, the computation in our transition system is completely deterministic and performs the calculation of $\hat{M}$. We omit these cases because they are analogous to the corresponding steps in Section 4.2. In case 3, the transition sequence that simulates *DecJump(*r_j*,s)* includes nondeterministic choice. As described in

Section 4.1, performing only *right* choices (cases 3.1 and 4.1.1) results in an exact simulation of $\hat{M}$'s transition $c \rightarrow_{\hat{M}} c'$, i.e., the transition sequence leads to the corresponding state $< div(\hat{M}), TS(c') >$. Performing at least one *wrong* choice (cases 3.2, 4.1.2 and 4.2) causes the tuple *flow* to be removed from the *Tuple Space* configuration, thus leading to some state $< P, M >$ where P is totally blocked under M, denoted by $< P, M > \not\rightarrow$.

3. $k \in \{1, ..., n\} \wedge I_k = k : DecJump(r_j, s) \wedge v_j \neq 0$, then $\hat{M}$ decrements r_j and increments k. The possible transition sequences in $MTS\text{-}mp[div(\hat{M})]$ are:
$<div(\hat{M}), TS(c)> \overset{nondet.}{\rightarrow}$

3.1 ***right****:*

$$
\begin{array}{ll}
 & <div(\hat{M}) \mid in(r_j).out(p_{k+1}), TS(c) \setminus \{p_k\}> \\
\rightarrow & <div(\hat{M}) \mid out(p_{k+1}), TS(c) \setminus \{p_k, r_j\}> \\
\rightarrow & <div(\hat{M}), TS(c) \setminus \{p_k, r_j\} \uplus \{p_{k+1}\}> \\
= & <div(\hat{M}), TS(c')>
\end{array}
$$

3.2 ***wrong****:*

$$
\begin{array}{ll}
 & <div(\hat{M}) \mid in(r_j).out(flow) \mid \\
 & \quad wait.wait.out(r_j).in(flow).out(p_s), TS(c) \setminus \{p_k\}> \\
\rightarrow & <div(\hat{M}) \mid out(flow) \mid wait.out(r_j).in(flow).out(p_s), TS(c) \setminus \{p_k, r_j\}> \\
\rightarrow & <div(\hat{M}) \mid out(r_j).in(flow).out(p_s), TS(c) \setminus \{p_k, r_j\} \uplus \{flow\}> \\
\rightarrow & <\Pi\ [I_i] \mid in(flow).out(p_s), TS(c) \setminus \{p_k\}> \not\rightarrow
\end{array}
$$

4. $k \in \{1, ..., n\} \wedge I_k = k : DecJump(r_j, s) \wedge v_j = 0$, then $\hat{M}$ assigns $k := s$
$<div(\hat{M}), TS(c)> \overset{nondet.}{\rightarrow}$

4.1 ***right***:

$$\begin{aligned}
& <div(\hat{M}) \mid in(r_j).out(flow) \mid \\
& \quad wait.wait.out(r_j).in(flow).out(p_s), TS(c) \setminus \{p_k\}> \\
\rightarrow \ & <div(\hat{M}) \mid in(r_j).out(flow) \mid wait.out(r_j).in(flow).out(p_s), TS(c) \setminus \{p_k\}> \\
\rightarrow \ & <div(\hat{M}) \mid in(r_j).out(flow) \mid out(r_j).in(flow).out(p_s), TS(c) \setminus \{p_k\}> \\
\rightarrow \ & <div(\hat{M}) \mid in(r_j).out(flow) \mid in(flow).out(p_s), TS(c) \setminus \{p_k\} \uplus \{r_j\}> \\
\rightarrow \ & <div(\hat{M}) \mid out(flow) \mid in(flow).out(p_s), TS(c) \setminus \{p_k\}> \\
\rightarrow \ & <div(\hat{M}) \mid in(flow).out(p_s), TS(c) \setminus \{p_k\} \uplus \{flow\}> \overset{nondet.}{\rightarrow}
\end{aligned}$$

4.1.1 ***right - right***:

$$\begin{aligned}
& <div(\hat{M}) \mid out(p_s), TS(c) \setminus \{p_k\}> \\
\rightarrow \ & <div(\hat{M}), TS(c) \setminus \{p_k\} \uplus \{p_s\}> \\
= \ & <div(\hat{M}), TS(c')>
\end{aligned}$$

4.1.2 ***right - wrong***:

$$<\Pi\ [I_i] \mid in(flow).out(p_s), TS(c) \setminus \{p_k\}> \not\rightarrow$$

4.2 ***wrong***:

$$<div(\hat{M}) \mid in(r_j).out(p_{k+1}), TS(c) \setminus \{p_k\}> \not\rightarrow$$

Chapter 5

Complexity

5.1 Overview

In this chapter, we consider complexity issues for interaction systems. We start out with a reduction of the classic 3-SAT problem to local, respectively global deadlock in interaction systems. This result, as published in [Min07], was to the best of our knowledge the first one concerning complexity issues of interaction systems. Then we present an exact classification of the problems of reachability, local and global deadlock, progress and availability in interaction systems by proving them all to be PSPACE-complete. The PSPACE-hardness results are based on the function $trans_1$ (cf. Section 3.1.1) that yields PSPACE-hardness for reachability in interaction systems. We then extend this result by a chain of reductions to the problems mentioned above and prove that the last problem in this reduction chain is in PSPACE.

5.2 Reducing 3-SAT to LDIS and GDIS

In interaction systems, (local) deadlocks may arise where a group of components is engaged in a cyclic waiting and will thus no longer participate in the progress of the global system. We show that deadlock-detection is NP-hard by encoding the classic 3-SAT problem [GJ79] to (deadlock detection for) interaction systems. For this purpose, we apply two ideas: First, we ensure that in all situations where a deadlock arises, a global deadlock arises[1]. Second, the components we introduce for a clause of a 3-CNF formula will always be able to participate in some interaction while the clause evaluates to false. So at the time a deadlock occurs, no further interactions can be performed and, i.e., no clause evaluates to false.

Let $F = k_1 \wedge \ldots \wedge k_n$ with $k_i = (l_{(i,1)} \vee l_{(i,2)} \vee l_{(i,3)})$ be a propositional formula in 3-CNF, where $l_{(i,1)}, l_{(i,2)}$ and $l_{(i,3)}$ are positive literals (i.e., variables) or negative literals (i.e., negated variables).

In the following, we construct an interaction system $Sys(F)$, such that
$(F \in 3\text{-}SAT) \Leftrightarrow (Sys(F) \in GDIS) \Leftrightarrow (Sys(F) \in LDIS)$.
We represent each clause k_i by a component $(i,0)$ and each literal $l_{(i,j)}$ by a component (i,j). By $i+1$ we mean $i+1$, if $1 \leq i \leq n-1$ and 1 if $i = n$.

Let $Sys(F) = (K, \{A_{(i,j)}\}_{(i,j)\in K}, C, Comp, \{T_{(i,j)}\}_{(i,j)\in K})$, where:
$K = \{(i,j) \mid 1 \leq i \leq n, 0 \leq j \leq 3\}$.
$A_{(i,0)} = \{init_{(i,0)}, false_{(i,0)}\}$ for $1 \leq i \leq n$
$A_{(i,j)} = \{init_{(i,j)}, \textit{set-to-1}_{(i,j)}, \textit{set-to-0}_{(i,j)}, \mathrm{true}_{(i,j)}, \mathrm{false}_{(i,j)}\}$ for
$1 \leq i \leq n$ and $1 \leq j \leq 3$
$C := \{\{init_{(i+1,0)}, init_{(i,1)}, init_{(i,2)}, init_{(i,3)}\} \mid 1 \leq i \leq n\}$
$\cup \{\{\textit{set-to-1}_{(i_1,j_1)}, \textit{set-to-1}_{(i_2,j_2)}, \ldots, \textit{set-to-1}_{(i_a,j_a)}\} \mid$
$\exists$ variable x that occurs in $l_{(i_1,j_1)}, \ldots, l_{(i_a,j_a)}$ and only there$\}$
$\cup \{\{\textit{set-to-0}_{(i_1,j_1)}, \textit{set-to-0}_{(i_2,j_2)}, \ldots, \textit{set-to-0}_{(i_a,j_a)}\} \mid$
$\exists$ variable x that occurs in $l_{(i_1,j_1)}, \ldots, l_{(i_a,j_a)}$ and only there$\}$

[1]Remember that a global deadlock is a special case of a local deadlock. This also means that we reduce 3-Sat to both local and global deadlock analysis.

$\cup \{\{false_{(i,0)}, false_{(i,1)}, false_{(i,2)}, false_{(i,3)}\} \mid 1 \leq i \leq n\}$

$\cup \{\{true_{(i,j)}, init_{(i+1,0)}\} \mid 1 \leq i \leq n, 1 \leq j \leq 3\}$

$Comp = \emptyset$

The local transition systems $T_{(i,0)}$ for $1 \leq i \leq n$ are given in Figure 5.1 (a). The local transition systems $T_{(i,j)}$ for $1 \leq i \leq n$, $1 \leq j \leq 3$ where $l_{(i,j)}$ is a positive (resp. negative) literal are given in Figure 5.1 (b) (resp. (c)).

We call components $(i,0)$ clause-components and components (i,j) with $1 \leq j \leq 3$ literal-components. For a component (i,j) we call the state $q^f_{(i,j)}$ its *false-state* and, if it exists, the state $q^t_{(i,j)}$ its *true-state*. We call both $q^t_{(i,j)}$ and $q^f_{(i,j)}$ *local final states*. We call a global state $q \in Q$ *global final state*, if all components are in local final states in q.

Remark 5.1
There is a natural 1-to-1-correspondence between assignments and reachable global final states:
An assignment σ for F corresponds to the global final state $q^{end} := state(\sigma)$, where all clause-components are in their false-states (they have no other local final state) and any literal-component (i,j) that represents a literal of variable x with $\sigma(x) = 1$ ($\sigma(x) = 0$) is in the local final state that is reachable by the set-to-1-action (by the set-to-0-action).
A global final state q^{end} that is in fact reachable starting in q^0 (i.e., all literal-components for the same variable have been set conjointly) corresponds to the assignment $\sigma := ass(q^{end})$, where for each variable x, $\sigma(x) = 1$ ($\sigma(x) = 0$) if the literal-components in which x occurs are in their local final states that are reached by the set-to-1-action (by the set-to-0-action).

Example 5.1
Let $F = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$.
F is satisfiable, namely $\sigma(F) = 1$ for $\sigma(x_1) = 1, \sigma(x_2) = 1, \sigma(x_3) = 0$.
Consider the corresponding interaction system $Sys(F) = (K, \{A_i\}_{i \in K}, C,$

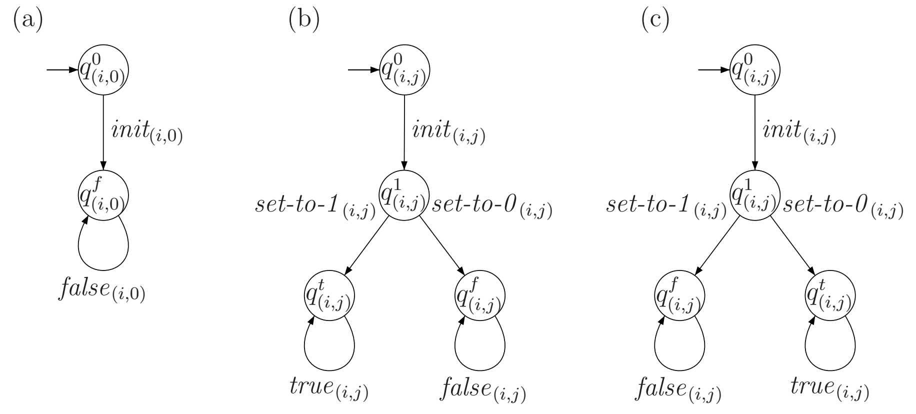


Figure 5.1: The local transition systems $T_{(i,j)}$ for clause-components (a) and positive and negative literal-components (b) and (c)

$\{T_i\}_{i \in K}$), where $K = \{(1,0),(1,1),(1,2),(1,3),(2,0),\ldots,(3,3)\}$ and the port sets $\{A_i\}_{i \in K}$ as well as the local transition systems $\{T_i\}_{i \in K}$ are given in Figure 5.2.

$$
\begin{aligned}
C := \{&\{init_{(2,0)}, init_{(1,1)}, init_{(1,2)}, init_{(1,3)}\}, \{init_{(3,0)}, init_{(2,1)}, init_{(2,2)}, init_{(2,3)}\},\\
&\{init_{(1,0)}, init_{(3,1)}, init_{(3,2)}, init_{(3,3)}\}\}\\
\cup \{&\{\textit{set-to-1}_{(1,1)},\ \textit{set-to-1}_{(2,1)},\ \textit{set-to-1}_{(3,1)}\},\\
&\{\textit{set-to-1}_{(1,2)},\ \textit{set-to-1}_{(2,2)},\ \textit{set-to-1}_{(3,2)}\},\\
&\{\textit{set-to-1}_{(1,3)},\ \textit{set-to-1}_{(2,3)},\ \textit{set-to-1}_{(3,3)}\}\}\\
\cup \{&\{\textit{set-to-0}_{(1,1)},\ \textit{set-to-0}_{(2,1)},\ \textit{set-to-0}_{(3,1)}\},\\
&\{\textit{set-to-0}_{(1,2)},\ \textit{set-to-0}_{(2,2)},\ \textit{set-to-0}_{(3,2)}\},\\
&\{\textit{set-to-0}_{(1,3)},\ \textit{set-to-0}_{(2,3)},\ \textit{set-to-0}_{(3,3)}\}\}\\
\cup \{&\{false_{(1,0)}, false_{(1,1)}, false_{(1,2)}, false_{(1,3)}\},\\
&\{false_{(2,0)}, false_{(2,1)}, false_{(2,2)}, false_{(2,3)}\},\\
&\{false_{(3,0)}, false_{(3,1)}, false_{(3,2)}, false_{(3,3)}\}\}\\
\cup \{&\{true_{(1,1)}, init_{(2,0)}\}, \{true_{(1,2)}, init_{(2,0)}\}, \{true_{(1,3)}, init_{(2,0)}\},\\
&\{true_{(2,1)}, init_{(3,0)}\}, \{true_{(2,2)}, init_{(3,0)}\}, \{true_{(2,3)}, init_{(3,0)}\},
\end{aligned}
$$

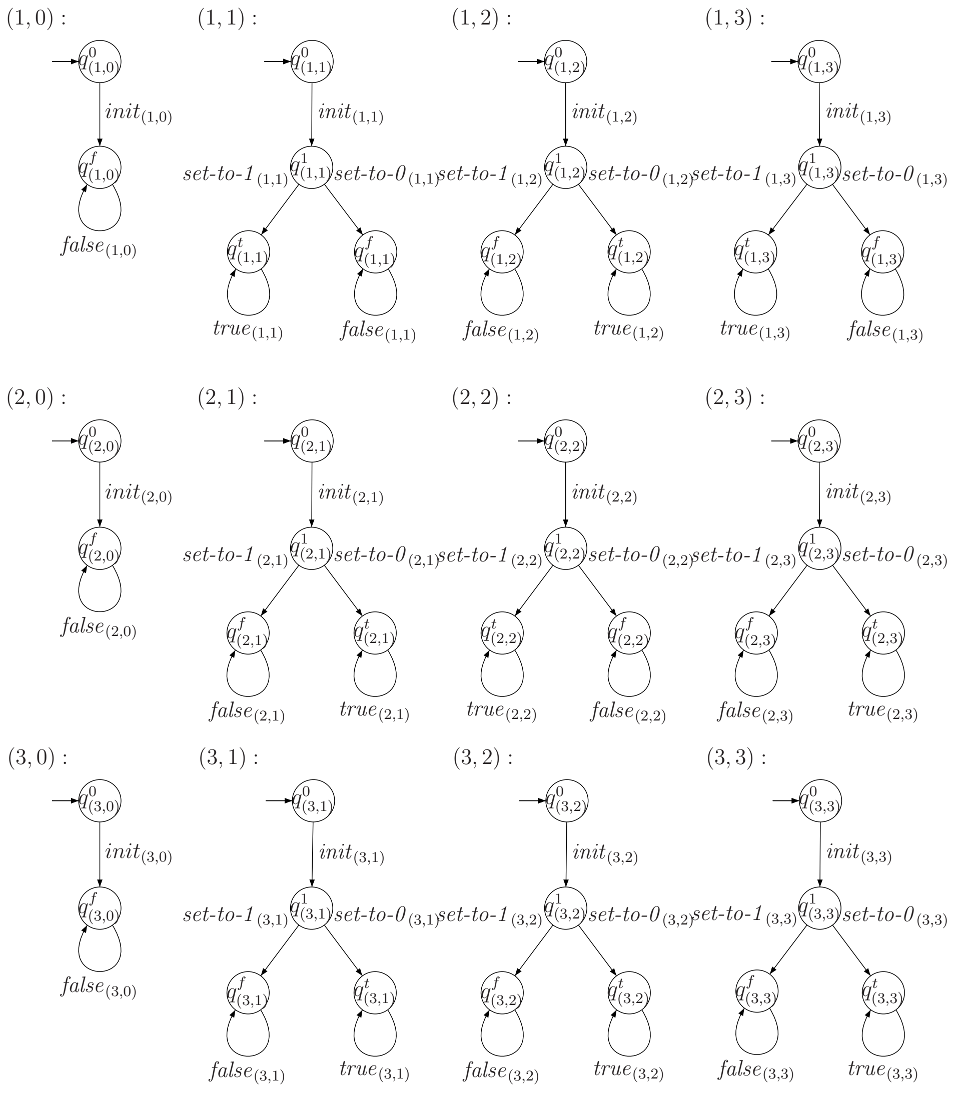


Figure 5.2: The local transition systems $\{T_{(i,j)}\}_{(i,j)\in K}$ for Example 5.1

$$\{true_{(3,1)}, init_{(1,0)}\}, \{true_{(3,2)}, init_{(1,0)}\}, \{true_{(3,3)}, init_{(1,0)}\}\}$$

$q^0 = (q^0_{(1,0)}, q^0_{(1,1)}, q^0_{(1,2)}, q^0_{(1,3)}, q^0_{(2,0)}, q^0_{(2,1)}, q^0_{(2,2)}, q^0_{(2,3)}, q^0_{(3,0)}, q^0_{(3,1)}, q^0_{(3,2)}, q^0_{(3,3)})$

As mentioned above, F is satisfiable by σ. We will show that $Sys(F)$ can reach the global final state $state(\sigma)$, where K is a deadlock:

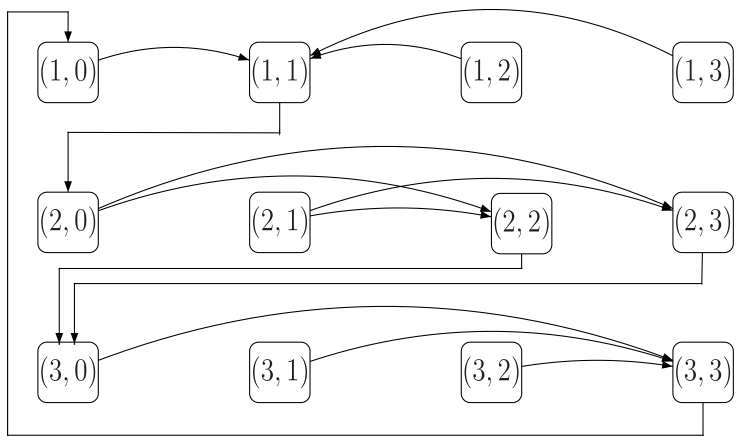


Figure 5.3: A graphical representation of the global deadlock in q^{end} in Example 5.1

We subsequently perform the interactions $\{init_{(i+1,0)}, init_{(i,1)}, init_{(i,2)}, init_{(i,3)}\}$ for all $1 \le i \le 3$.

Then, the clause-components $(i, 0)$ are in their states $q^f_{(i,0)}$ and the literal-components (i, j) are in their states $q^1_{(i,j)}$.

Now, we perform :

$\{$*set-to-1* $_{(1,1)}$,*set-to-1* $_{(2,1)}$,*set-to-1* $_{(3,1)}\}$, $\{$*set-to-1* $_{(1,2)}$,*set-to-1* $_{(2,2)}$,*set-to-1* $_{(3,2)}\}$ and $\{$*set-to-0* $_{(1,3)}$,*set-to-0* $_{(2,3)}$,*set-to-0* $_{(3,3)}\}$.

Then, K is a deadlock in the global state

$q^{end} = (q^f_{(1,0)}, q^t_{(1,1)}, q^f_{(1,2)}, q^f_{(1,3)}, q^f_{(2,0)}, q^f_{(2,1)}, q^t_{(2,2)}, q^t_{(2,3)}, q^f_{(3,0)}, q^f_{(3,1)}, q^f_{(3,2)}, q^t_{(3,3)})$

The global deadlock situation is displayed in Figure 5.3, where the nodes (i, j) represent the components (not their local states) and an edge from node (i_1, j_1) to (i_2, j_2) means that (i_1, j_1) waits for (i_2, j_2).

Polynomiality of the reduction:

There is no critical blow-up in notation when we go from F to $Sys(F)$. The four transition systems we introduce for each clause are of constant size. Also, the set-to-1- and set-to-0-connectors have an overall size which is linear in the number of literals in F and the other $(5n)$ connectors in C are of constant size.

Proposition 5.1

$D \subseteq K$ *is a local deadlock in a reachable state* q *and* $(i, j) \in D$

$\Rightarrow$ (i, j) *is in a local final state.*

Proof:

Assume that component $(i, 0)$ $(1 \leq i \leq n)$ is part of a deadlock $D \subseteq K$ and in its local non-final state $q^0_{(i,0)}$. Obviously in any case, the enabled $init_{(i,0)}$-action can be performed together with the $init_{(i-1,j)}$-actions of the corresponding literal-components, as those cannot have left their starting states, so $(i, 0)$ cannot be part of a deadlock.

Assume that component (i, j) $(1 \leq i \leq n, 1 \leq j \leq 3)$ is part of a deadlock $D \subseteq K$ and in one of its local non-final states:

If (i, j) is in $q^0_{(i,j)}$, then the $\{init_{(i+1,0)}, init_{(i,1)}, init_{(i,2)}, init_{(i,3)}\}$-interaction can still be performed because the actions $init_{(i,j)}(1 \leq j \leq 3)$ occur in no other connector and the action $init_{(i+1,0)}$ occurs in other connectors $\{true_{(i,j)}, init_{(i+1,0)}\}$ but only together with the true-actions of the discussed components (i, j) which they do not offer until they have left their starting states which is not the case as we assumed that (i, j) is in $q^0_{(i,j)}$. So (i, j) cannot be part of a deadlock and in particular (i, j) can still proceed to $q^1_{(i,j)}$.

If (i, j) is in $q^1_{(i,j)}$, then the *set-to*-1- or *set-to*-0-actions can still be performed in the future, because no other literal-component of the same variable can have reached a local final state. This is obvious, as they can only transit

conjointly (see definition of C). Also, any of these literal-components can proceed to $q^1_{(i,j)}$ as explained above, if it is not in this state already.
So (i,j) can still perform some action in the future and thus cannot be part of a deadlock. Hence, (i,j) must be in a local final state.

Lemma 5.1
$(Sys(F) \in GDIS) \Leftrightarrow (Sys(F) \in LDIS)$

Proof $\Rightarrow$:
By definition, a global deadlock is a special case of a local deadlock.

Proof $\Leftarrow$:
1) Let q be a reachable state in $Sys(F)$, s.t. $D \subseteq K$ is a local deadlock in q. Then a literal-component (i,j) $(1 \leq j \leq 3)$ participates in D (because the clause-components do not communicate with each other directly).
2) Due to Proposition 5.1, (i,j) is in a final state. We show that at least one of the literal-components of clause i must be in its true-state: Assume that (i,j) is in $q^f_{(i,j)}$ (otherwise we are done). Then, $ea(q^f_{(i,j)}) = \{\mathit{false}_{(i,j)}\}$, which occurs in the connector $\{\mathit{false}_{(i,0)}, \mathit{false}_{(i,1)}, \mathit{false}_{(i,2)}, \mathit{false}_{(i,3)}\}$. If $(i,0) \in D$, then $(i,0)$ is in its local final state $q^f_{(i,0)}$. Therefore, (i,j) does not wait for $(i,0)$. Hence, one of the literal-components of clause i must participate in D. Due to Proposition 5.1 it must be in a final state where it does not offer the false action, i.e., its true-state.
3) The literal-component of clause i, which is in its true-state can only wait for the clause-component $(i+1,0)$. So we have $(i+1,0) \in D$ and due to Proposition 5.1 $(i+1,0)$ has to be in its only local final state, i.e., its false-state.
4) As $(i+1,0) \in D$ offers $\mathit{false}_{(i+1,0)}$, at least one of the literal-components

of clause $i+1$ has to be in D and in its true-state. From here, we apply induction by going to 3) and conclude the same for all clauses.
5) It is possible that some variables have not yet been set to 0 or 1, i.e., the corresponding literal-components are not yet in their final states, so the deadlock D would not be global. It is however quite obvious, that we still may perform interactions such that these components finally reach local final states. We call the thus reached state q' and in q', $D = K$ is a global deadlock.

Corollary 5.1
From the observations described under "$\Leftarrow$" we may deduce:
If $Sys(F)$ is a global deadlock, at least one of the literal-components of each clause is in its true-state.

Lemma 5.2
(F is satisfiable) $\Leftrightarrow$ ($Sys(F) \in GDIS$)

Proof $\Rightarrow$:
Let $F = k_1 \wedge \ldots \wedge k_n$ with $k_i = (l_{(i,1)} \vee l_{(i,2)} \vee l_{(i,3)})$ be a satisfiable 3-CNF formula and let $\sigma(F) = 1$ for an assignment σ.
The starting state of $Sys(F)$ is $q^0 := (q^0_{(1,0)}, q^0_{(1,1)}, q^0_{(1,2)}, q^0_{(1,3)}, q^0_{(2,0)}, \ldots, q^0_{(n,3)})$.
Let $Sys(F)$ perform the following transitions:
1) For all $1 \leq i \leq n$ perform the interactions $\{init_{(i+1,0)}, init_{(i,1)}, init_{(i,2)}, init_{(i,3)}\}$.
Then all clause-components $(i,0)$ $(1 \leq i \leq n)$ are in their false-states $q^f_{(i,0)}$ and all literal-components (i,j) $(1 \leq i \leq n, 1 \leq j \leq 3)$, are in their states $q^1_{(i,j)}$.
2) Let x be a variable that occurs in F at the positions (i_1, j_1), (i_2, j_2), ..., (i_a, j_a) (and only there), and let $\sigma(x) = 1$ (or $\sigma(x) = 0$, respectively).
Then perform the interaction $\{$*set-to-1* $_{(i_1,j_1)}$, *set-to-1* $_{(i_2,j_2)}$, ..., *set-to-1* $_{(i_a,j_a)}\}$

(or $\{$*set-to-0*$_{(i_1,j_1)}$, *set-to-0*$_{(i_2,j_2)}, \ldots,$*set-to-0*$_{(i_a,j_a)}\}$, respectively).
After having performed the corresponding interaction for each variable that occurs in F, we reached the global final state $q^{end} = state(\sigma)$ that we described in Remark 5.1.
As $\sigma(F) = 1$, we have $\sigma(k_i) = 1$ for all $1 \leq i \leq n$, i.e., in each clause there is at least one literal that evaluates to 1 under σ. This means there is at least one positive literal $l_{(i,j)} = x$ with $\sigma(x) = 1$ or a negative literal $l_{(i,j)} = \overline{x}$ with $\sigma(x) = 0$. In both cases, the corresponding transition system $T_{(i,j)}$ has reached its local state $q^t_{(i,j)}$ (cf. Figure 5.1, (b) and (c)).

Hence, we have $\forall 1 \leq i \leq n \; q^{end}(i,0) = q^f_{(i,0)}$ and $ea(q^f_{(i,0)}) = \{false_{(i,0)}\}$.
Furthermore, $\forall 1 \leq i \leq n \; \exists j \in \{1,2,3\}$, s.t. $q^{end}(i,j) = q^t_{(i,j)}$ and $ea(q^t_{(i,j)}) = \{true_{(i,j)}\}$.
Obviously, $Sys(F)$ is a global deadlock in q^{end} (or in other words $D = K$ is a deadlock in q^{end} in $Sys(F)$), as every clause-component $(i,0)$ waits for at least one of its literal-components $(i,1),(i,2),(i,3)$. Those literal-components in $(i,1),(i,2),(i,3)$ that are in their q^f-states, also wait for the ones that are in their q^t-states. Meanwhile, those literal-components that are in their q^t-states wait for the clause-component $(i+1,0)$. Hence, we observe a cyclic waiting over all clauses (cf. Example 5.1, Figure 5.3), that includes all components.

Proof $\Leftarrow$:

$F \in GDIS$ means that there is a reachable global state, where K is a deadlock.
By Corollary 5.1 we know that at least one component of every clause must be in its true-state.

Due to the one-to-one correspondence of literal-components to literals (cf. Remark 5.1), the fact that all occurrences of a variable x are consistently set to 1 or 0 and the fact that in each clause at least one literal evaluates to "true", we may conclude the existence of a satisfying assignment σ.

5.3 *Everything is PSPACE-complete in Interaction Systems*

In this section, we first prove the problems of reachability, progress, global and local deadlock and availability to be PSPACE-hard[2]. We build on the result established in Section 3.1.1, where we gave a polynomial translation from 1-safe Petri nets to interaction systems which yielded PSPACE-hardness for reachability in interaction systems. Now we give four polynomial reductions $f_1, \dots, f_4$ that build a reduction chain as depicted in Figure 5.4. The chain allows to derive the PSPACE-hardness result for all considered properties from the PSPACE-hardness of reachability as well as PSPACE-solvability for all properties in the chain from the PSPACE-solvability of availability. Hence, we formally prove all problems in the chain to be PSPACE-complete. Although the reductions vary strongly in their degree of difficulty, they also

$$\begin{tikzcd}
\textit{Reachability} \arrow[r, "f_1"] \arrow[d, dash, dashed, "\text{Corollary 3.1}"] & \textit{Progress} \arrow[r, "f_2"] & \textit{GDIS} \arrow[r, "f_3"] & \textit{LDIS} \arrow[r, "f_4"] & \textit{Availability} \arrow[d, dash, dashed, "\text{Theorem 5.5}"'] \\
\boxed{\text{PSPACE-hard}} & & & & \boxed{\in \text{PSPACE}}
\end{tikzcd}$$

Figure 5.4: The polynomial time reductions $f_i \quad (1 \leq i \leq 4)$

[2]This result outruns our earlier published result that we presented in Section 5.2.

have one basic idea in common. In each of the reductions, we add a component *main* to the system. However, the purpose (and thus the behavior) of *main* will be a different one for each reduction.
For each reduction, we present its formal definition followed by a short explanation and sometimes a proof of correctness. The logspace computability is obvious, so proofs are omitted.

5.3.1 Reachability is polynomially reducible to Progress

Theorem 5.1

Reachability is polynomially reducible to Progress

Proof: Let $Sys \in \widetilde{IS}$ and $q = (q_1, \ldots, q_n) \in Q[Sys]$. We associate with ($Sys$,$q$) an interaction system $f_1(Sys, q)$ (which is free of global deadlocks) s.t.

$$((Sys,q) \in Reachability) \Leftrightarrow ((f_1(Sys,q),main) \notin Progress).$$

Formal definition of f_1 :

Let $Sys = \{K, \{A_i\}_{i\in K}, C, Comp, \{T_i\}_{i\in K}\}$, then

$f_1(Sys, q) = \{K', \{A'_i\}_{i\in K'}, C', Comp', \{T'_i\}_{i\in K'}\}$, where

$$\begin{aligned}
\boldsymbol{K'} &:= K \cup \{main\}, \\
\text{For } i \in K: \boldsymbol{A'_i} &:= A_i \cup \{run_i\}, \\
\boldsymbol{A'_{main}} &:= \{dummy_{main}, check_{main}\}, \\
\text{For } i \in K: \boldsymbol{T'_i} &:= (Q_i, A'_i, \rightarrow'_i, q_i^0), \text{ where} \\
\rightarrow'_i &:= \rightarrow_i \cup \{(q_i, run_i, q_i)\}, \\
\boldsymbol{T'_{main}} &:= (\{q^0_{main}\}, A'_{main}, \rightarrow'_{main}, q^0_{main}), \text{ where} \\
\rightarrow'_{main} &:= \{(q^0_{main}, check_{main}, q^0_{main}), (q^0_{main}, dummy_{main}, q^0_{main})\}.
\end{aligned}$$

$$\begin{aligned} \boldsymbol{C'} &:= \{c \cup \{check_{main}\} \,|\, c \in C\} \cup \{\{run_i \,|\, 1 \leq i \leq n\}\} \cup \{\{dummy_{main}\}\}, \\ \boldsymbol{Comp'} &:= \{\alpha \cup \{check_{main}\} \mid \alpha \in Comp\}. \end{aligned}$$

Explanation: We add a component *main* whose local transition system consists of a single state with two loops. For each local transition system T_i we add a loop in each state q_i labeled by run_i. Clearly $f_1(Sys, q) \in \widetilde{IS}$ holds. The loop of *main* labeled by $dummy_{main}$ can be performed independently (i.e., $\{dummy_{main}\}$ is a connector) and assures that $f_1(Sys,q) \notin GDIS$ (which is a precondition for asking for progress). The second loop is labeled by the action $check_{main}$, which is added to every interaction $\alpha \in C \cup Comp$. Hence, the only interaction in $C \cup Comp$ in which *main* does not participate is $\{run_1, \ldots, run_n\}$.

This fact, together with the obvious observation that q is reachable in *Sys* iff q extended by q^0_{main} is reachable in $f_1(Sys, q)$ allows us to conclude that in $f_1(Sys, q)$ there is a run from q in which *main* does not participate iff q is reachable in *Sys*.

5.3.2 *Progress is polynomially reducible to GDIS*

Preliminaries: We present a parameterized counter-system in order to build an interaction system for an m^n-counter, $m, n \in \mathbb{N}$:

$Count_{(m,n)} = (\{n+1, \ldots, 2n\}, \{A_i\}_{n+1 \leq i \leq 2n}, C, Comp, \{T_i\}_{n+1 \leq i \leq 2n})$,

$$\begin{aligned} \text{where } A_i &= \{inc_i, dec_i\} \text{ for } n+1 \leq i \leq 2n-1 \text{ and } A_{2n} = \{inc_{2n}, dummy_{2n}\} \\ C &= \{\{inc_{n+1}, dummy_{2n}\}\} \cup \textstyle\bigcup_{i=n+2}^{2n} \{c(inc_i)\} \\ &\qquad\qquad\qquad \text{where } c(inc_i) = \{inc_i\} \cup \textstyle\bigcup_{j=n+1}^{i-1} \{dec_j\}, \\ Comp &= \{\{inc_{n+1}\}, \{inc_{n+1}, dummy_{2n}\}\}, \\ T_i &= (Q_i, A_i, \rightarrow_i, q_i^0), \text{ where } Q_i = \{q_i^0, \ldots, q_i^{m-1}\} \text{ and} \end{aligned}$$

$$\to_i = \begin{cases} \{(q_i^j, inc_i, q_i^{j+1}) \mid 0 \leq j \leq m-2\} \cup \{(q_i^{m-1}, dec_i, q_i^0)\} \ ; n+1 \leq i \leq 2n-1 \\ \{(q_i^j, inc_i, q_i^{j+1}) \mid 0 \leq j \leq m-2\} \cup \{(q_i^{m-1}, dummy_{2n}, q_i^{m-1})\} \ ; i = 2n \end{cases}$$

A system $Count_{(m,n)}$ behaves deterministically and simply performs $m^n - 1$ ("counting") interactions before stopping. It nicely demonstrates the capability of interaction systems to synchronize with different numbers of participants.

Example 5.2

$Count_{(3,4)} = (\{5, 6, 7, 8\}, \{A_i\}_{5 \leq i \leq 8}, C, Comp, \{T_i\}_{5 \leq i \leq 8})$, where

$$\begin{aligned} A_i &= \{inc_i, dec_i\} \ (5 \leq i \leq 7),\ A_8 = \{inc_8, dummy_8\}, \\ C &= \{\ \{inc_5, dummy_8\}, \{inc_6, dec_5\}, \{inc_7, dec_6, dec_5\}, \\ &\quad \{inc_8, dec_7, dec_6, dec_5\}\}, \\ Comp &= \{\{inc_5\}, \{inc_5, dummy_8\}\}, \text{ and the } T_i\text{'s are given in Figure 5.5.} \end{aligned}$$

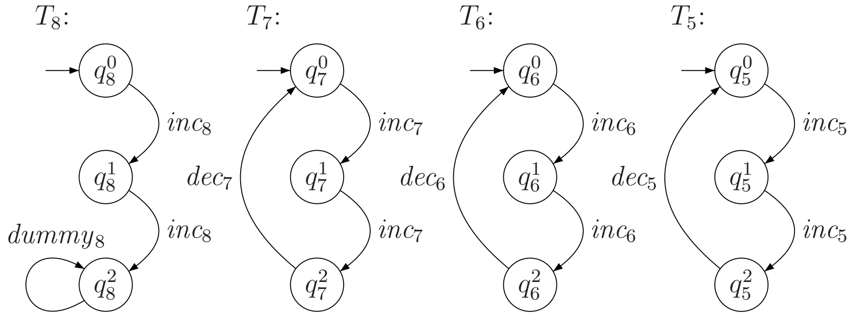


Figure 5.5: The local transition systems for $Count_{(3,4)}$

The behavior[3] of our example system $Count_{(3,4)}$ is as follows: It performs a deterministic computation starting in $(q_5^0, q_6^0, q_7^0, q_8^0)$. The system describes a 3^4-counter that counts from 0 to 3^4-1=80 and then cannot perform any further interaction.

We refer to the local transition system T_i of a component i of some pre-

[3] $dummy_8$ is introduced only to ensure that T_8 is non-terminating

viously defined system Sys by $T_i[Sys]$. The same notation is used for the other elements of the interaction system tuple. E.g., $Comp[Count_{(3,4)}] = \{\{inc_5\}, \{inc_5, dummy_8\}\}$. Whenever it is obvious by the context to which system we refer (as, e.g., in the next subsection), we may simply write Q instead of $Q[Sys]$, etc. for ease of notation.

Theorem 5.2

Progress is polynomially reducible to GDIS.

Proof:

Let $Sys \in (\widetilde{IS} \setminus GDIS)$ and $k \in K[Sys]$. If k participates in every $\alpha \in C \cup Comp$, then k makes progress[4]. Otherwise, we associate with (Sys,k) an interaction system $f_2(Sys,k)$ s.t.

$$(Sys,k) \in Progress \Leftrightarrow f_2(Sys,k) \notin GDIS.$$

In the following, let $m := max\{|Q_i| \mid i \in K[Sys]\}$.

Formal definition of f_2 :

Let $Sys = \{K, \{A_i\}_{i\in K}, C, Comp, \{T_i\}_{i\in K}\}$, then

$f_2(Sys,k) = \{K', \{A'_i\}_{i\in K'}, C', Comp', \{T'_i\}_{i\in K'}\}$, where

$$\begin{array}{rcl}
\boldsymbol{K'} & := & K \cup \{n+1, \ldots, 2n, main\}, \\
\text{For } i \in K\text{: } \boldsymbol{A'_i} & := & A_i, \\
\text{For } i \in \{n+1, \ldots, 2n\}\text{: } \boldsymbol{A'_i} & := & A_i[Count_{(m,n)}], \\
\boldsymbol{A'_{main}} & := & \{check_{main}, exclude_{main}, count_{main}\}, \\
\text{For } i \in K\text{: } \boldsymbol{T'_i} & := & T_i, \\
\text{For } i \in \{n+1, \ldots, 2n\}\text{: } \boldsymbol{T'_i} & := & T_i[Count_{(m,n)}], \\
\text{and } \boldsymbol{T'_{main}} & \text{is} & \text{depicted in Figure 5.6.}
\end{array}$$

[4]We have to consider this case explicitly because $f_2(Sys,k) \notin \widetilde{IS}$ for such an input.

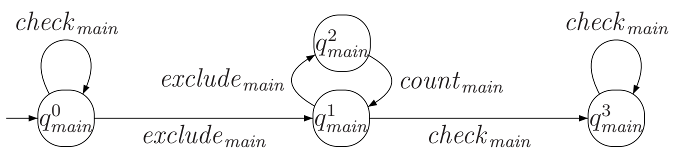


Figure 5.6: The local transition system T'_{main}

$$\begin{aligned}
\boldsymbol{C_{check}} &:= \{c \cup \{check_{main}\} \mid c \in C\} \\
\boldsymbol{Comp_{check}} &:= \{\alpha \cup \{check_{main}\} \mid \alpha \in Comp\} \\
\boldsymbol{C_{exclude}} &:= \{c \cup \{exclude_{main}\} \mid c \in C \wedge k(c) = \emptyset\} \\
\boldsymbol{Comp_{exclude}} &:= \{\alpha \cup \{exclude_{main}\} \mid \alpha \in Comp \wedge k(\alpha) = \emptyset\} \\
\boldsymbol{C_{counter}} &:= \{c \cup \{count_{main}\} \mid c \in C[Count_{(m,n)}]\} \\
\boldsymbol{Comp_{counter}} &:= \{\alpha \cup \{count_{main}\} \mid \alpha \in Comp[Count_{(m,n)}]\} \\
&= \{\{inc_{n+1}, count_{main}\}, \{inc_{n+1}, dummy_{2n}, count_{main}\}\} \\
\boldsymbol{C'} &:= C_{check} \cup C_{exclude} \cup C_{counter} \\
\boldsymbol{Comp'} &:= Comp_{check} \cup Comp_{exclude} \cup Comp_{counter}
\end{aligned}$$

Explanation: First, we observe that $f_2(Sys, k) \in \widetilde{IS}$ holds. *Sys* is globally deadlock-free and we want to know whether it contains a run from q^0, in which k does not participate infinitely often. According to Remark 2.8 (p. 31), this amounts to the question, whether there is a reachable global state, that lies on a cycle that does not involve k. As m^n is an upper bound for the size of the global state space of *Sys*, this is equivalent to asking whether it is possible to perform (not necessarily starting in q^0) m^n consecutive interactions in which k does not participate.

Lemma 5.3

(Sys,k) $\in$ *Progress* $\Leftrightarrow$ f_2*(Sys,k)* $\notin$ *GDIS*

Proof:

$\Leftarrow$ If there is a run from q^0 in which k does not participate infinitely often, then there is a global deadlock in $f_2(Sys, k)$:

In $f_2(Sys,k)$, we may perform arbitrarily many "original" interactions of *Sys* in the beginning, which are accompanied by the action $check_{main}$. Assuming that there is a run, in which k does not participate infinitely often, we will be able to reach a global state q of $f_2(Sys, k)$ which lies on a cycle whose transitions are labeled by interactions in which k does not participate. Thus, we may perform m^n consecutive[5] interactions of *Sys* in which k does not participate. These interactions will be accompanied by the action $exclude_{main}$ and all of them (except for the first) will be followed by a count-interaction. Once we have performed the $(m^n - 1)$-th count-interaction, the counter components are in deadlock causing a global deadlock in $f_2(Sys, k)$.

$\Rightarrow$ If there is a global deadlock in $f_2(Sys, k)$ then there is a run in *Sys* in which k does not participate infinitely often:

Note that there is no global deadlock in *Sys* and our construction does not interfere with the original computation of *Sys*. (I.e., the reachable global state space of $f_2(Sys, k)$ projected to the components $K = \{1, \ldots, n\}$ of *Sys* is equal to the reachable global state space of *Sys*.) As a consequence, $f_2(Sys, k)$ can proceed, as long as *main* offers its action $check_{main}$ which simply accompanies the original interactions of *Sys*. However, *main* offers this action in all but one of its local states. So a global deadlock is only possible when T'_{main} is in its local state

[5]consecutive in the sense of interactions that belonged to *Sys*, i.e., we disregard the interleaved count-interactions

q^2_{main} and offers its action $count_{main}$. The only case when $count_{main}$ is no longer possible occurs after $m^n - 1$ executions of $count_{main}$. In this case, m^n consecutive (i.e., interleaved only with the count-interactions) instances of $exclude_{main}$ have just occurred. As k does not participate in the corresponding exclude-interactions and as there are at most m^n global states in $|Q|$ we have visited at least one global state at least twice. Thus, we could (keeping to such a cycle) perform interactions excluding k forever and this corresponds to a run from q^0 in *Sys* in which k does not participate infinitely often. Thus, k does not make progress in *Sys*.

5.3.3 GDIS is polynomially reducible to LDIS

Theorem 5.3

GDIS is polynomially reducible to LDIS

Proof:

Let $Sys \in \widetilde{IS}$. We associate with *Sys* an interaction system $f_3(Sys)$ s.t.

$$(Sys \in GDIS) \Leftrightarrow (f_3(Sys) \in LDIS).$$

Formal definition of f_3 :

Let $Sys = \{K, \{A_i\}_{i\in K}, C, Comp, \{T_i\}_{i\in K}\}$, then

$f_3(Sys) = \{K', \{A'_i\}_{i\in K'}, C', Comp', \{T'_i\}_{i\in K'}\}$, where

$$\begin{aligned} \boldsymbol{K'} &:= K \cup \{main\}, \\ \text{For } i\in K\text{: } \boldsymbol{A'_i} &:= A_i \cup \{dummy_i\}, \\ \boldsymbol{A'_{main}} &:= \{dummy_{main}, check_{main}\}, \\ \text{For } i\in K\text{: } \boldsymbol{T'_i} &:= (Q_i, A'_i, \to'_i, q^0_i), \text{ where} \end{aligned}$$

$$\begin{aligned}
\rightarrow'_i &:= \rightarrow_i \cup \{(q_i, dummy_i, q_i) \mid q_i \in Q_i\}.\\
\boldsymbol{T'_{main}} &:= (\{q^0_{main}, q^1_{main}\}, A'_{main}, \rightarrow'_{main}, q^0_{main}), \text{ where}\\
\rightarrow'_{main} &:= \{(q^0_{main}, check_{main}, q^1_{main}), (q^1_{main}, dummy_{main}, q^0_{main})\},\\
\boldsymbol{C'} &:= \{c \cup \{check_{main}\} \mid c \in C\} \cup\\
&\quad\ \{\{dummy_1, \ldots, dummy_n, dummy_{main}\}\},\\
\boldsymbol{Comp'} &:= \{\alpha \cup \{check_{main}\} \mid \alpha \in Comp\}.
\end{aligned}$$

Explanation: Clearly, $f_3(Sys) \in \widetilde{IS}$. We add an additional component *main* which alternatingly accompanies original interactions of *Sys* in one step and then allows the system to perform a connector including all components in a second step. This preserves global deadlocks but resolves local ones.

Lemma 5.4

(Sys ∈ GDIS) $\Leftrightarrow$ *(f_3(Sys) ∈ LDIS)*

Proof:

$\Rightarrow$ Let *Sys* $\in$ *GDIS*. Let $q = (q_1, \ldots, q_n)$ a global state of *Sys*, which is reachable by a path ϕ from $q^0[Sys]$ such that $q \not\rightarrow$. By construction of f_3, the global state $(q_1, \ldots, q_n, q^1_{main})$ is reachable in $f_3(Sys)$ by starting in $q^0[f_3(Sys)]$ and performing interactions according to ϕ (extended by $check_{main}$) interleaved with instances of the *dummy*-connector. In $(q_1, \ldots, q_n, q^1_{main})$ the *dummy*-connector is enabled and by performing it, we reach $(q_1, \ldots, q_n, q^0_{main})$ which is a global deadlock state in $f_3(Sys)$.

$\Leftarrow$ Let $f_3(Sys) \in LDIS$, namely let $q = (q_1, \ldots, q_n, q_{main})$ in $Reach(f_3(Sys))$ and let $\emptyset \neq \tilde{K} \subseteq K'$ be a deadlock in q. $q_{main} = q^1_{main}$ is not possible because in this case the *dummy*-connector (in which all components participate, i.e., also $\tilde{K}$) may be performed which would be a contradiction to our deadlock assumption. So *main* is in q^0_{main} and offers $check_{main}$. Now if any $\alpha \in C' \cup Comp'$ with $check_{main} \in \alpha$ can be performed, *main* would also reach q^1_{main} and again the *dummy*-connector could be performed yielding the same contradiction. Hence, we conclude that in q no interaction $\alpha \in C' \cup Comp'$ with $check_{main} \in \alpha$ can be performed. By construction of f_3 this yields that in $(q_1, \ldots, q_n)$ no interaction in $C \cup Comp$ is possible and as $(q_1, \ldots, q_n)$ is reachable in *Sys*, $Sys \in GDIS$ follows.

5.3.4 LDIS is polynomially reducible to Availability

Theorem 5.4

LDIS is polynomially reducible to Availability

Proof:

Let $Sys \in \widetilde{IS}$. We associate with *Sys* an interaction system $f_4(Sys)$ (which is free of global deadlocks) s.t.

$$(Sys \in LDIS) \Leftrightarrow ((f_4(Sys), main) \notin Availability).$$

Formal definition of f_4 :

Let $Sys = \{K, \{A_i\}_{i \in K}, C, Comp, \{T_i\}_{i \in K}\}$,

then $f_4(Sys) = \{K', \{A'_i\}_{i \in K'}, C', Comp', \{T'_i\}_{i \in K'}\}$, where[6]

[6]For ease of notation we use sets of actions as edge labels in the definition of $\rightarrow'_i$ as well as in Figure 5.7. When we write $(q, A, q') \in \rightarrow'_i$ we mean $(q, a, q') \in \rightarrow'_i \ \forall a \in A$. Note that by $ea(q_i)$ we refer to the enabled actions of the local state q_i in *Sys* (not in $f_4(Sys)$).

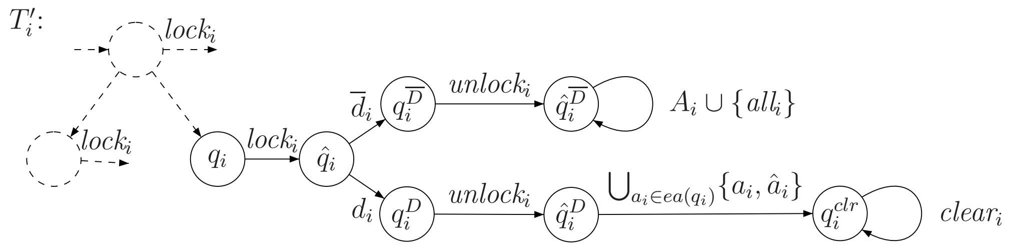


Figure 5.7: The modification for a local state q_i in the transition system T'_i

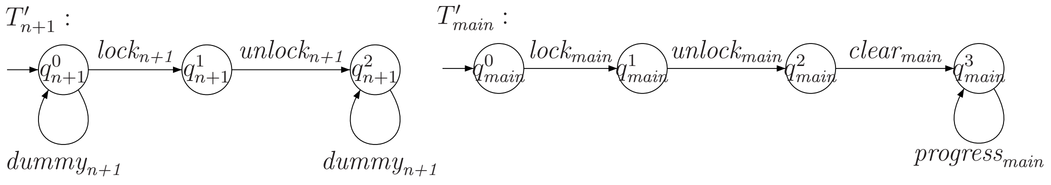


Figure 5.8: The local transition systems T'_{main} and T'_{n+1}

$$
\begin{aligned}
\boldsymbol{K'} &:= K \cup \{n{+}1\} \cup \{main\} \\
\text{For } i \in K\text{: } \boldsymbol{A'_i} &:= A_i \cup \{\hat{a}_i \mid a_i \in A_i\} \cup \{lock_i, unlock_i, d_i, \overline{d}_i, clear_i\} \\
\boldsymbol{A'_{n+1}} &:= \{dummy_{n+1}, lock_{n+1}, unlock_{n+1}\} \\
\boldsymbol{A'_{main}} &:= \{lock_{main}, unlock_{main}, clear_{main}, progress_{main}\} \\
\text{For } i \in K\text{: } \boldsymbol{T'_i} &:= (Q'_i, A'_i, \rightarrow'_i, q^0_i), \text{ where} \\
Q'_i &:= \textstyle\bigcup_{q_i \in Q_i}\{q_i, \hat{q}_i, q_i^D, \hat{q}_i^D, q_i^{\overline{D}}, \hat{q}_i^{\overline{D}}, q_i^{clr}\} \\
\rightarrow'_i &:= \textstyle\bigcup_{q_i \in Q_i}\{\ (q_i, lock_i, \hat{q}_i), (\hat{q}_i, d_i, q_i^D), (q_i^D, unlock_i, \hat{q}_i^D), \\
&\qquad\qquad (\hat{q}_i, \overline{d}_i, q_i^{\overline{D}}), (q_i^{\overline{D}}, unlock_i, \hat{q}_i^{\overline{D}}), \\
&\qquad\qquad (\hat{q}_i^D, \textstyle\bigcup_{a_i \in ea(q_i)}\{a_i, \hat{a}_i\}, q_i^{clr}), \\
&\qquad\qquad (\hat{q}_i^{\overline{D}}, A_i \cup \{all_i\}, \hat{q}_i^{\overline{D}}), (q_i^{clr}, clear_i, q_i^{clr})\} \\
&\quad\ \cup \rightarrow_i
\end{aligned}
$$

$\boldsymbol{T'_{n+1}}$ and $\boldsymbol{T'_{main}}$ are given in Figure 5.8.

The result of our modifications is sketched for a state $q_i \in Q_i$ in Figure 5.7.

$$\begin{array}{rcl}
\boldsymbol{C'} & := & \{\{\mathit{dummy}_{n+1}\}, \{\mathit{lock}_1, \ldots, \mathit{lock}_n, \mathit{lock}_{n+1}, \mathit{lock}_{\mathit{main}}\}\} \\
& \cup & \{\{\mathit{unlock}_1, \ldots, \mathit{unlock}_n, \mathit{unlock}_{n+1}, \mathit{unlock}_{\mathit{main}}\}\} \\
& \cup & \{\{d_1\}, \ldots, \{d_n\}, \{\overline{d}_1\}, \ldots, \{\overline{d}_n\}\} \\
& \cup & \{\{\mathit{all}_1, \ldots, \mathit{all}_n, \mathit{clear}_{\mathit{main}}\}\} \\
& \cup & \{\{\mathit{clear}_1, \mathit{clear}_{\mathit{main}}\}, \ldots, \{\mathit{clear}_n, \mathit{clear}_{\mathit{main}}\}\} \\
& \cup & \{\{\mathit{progress}_{\mathit{main}}\}\} \\
& \cup & C \cup C^{\mathit{clear}}, \text{ where} \\
C^{\mathit{clear}} & := & \{\{\mathit{clear}_{\mathit{main}}, \hat{a}\} \cup (c \setminus a) \mid a \in c \in C\} \\
\boldsymbol{\mathit{Comp}'} & := & \mathit{Comp} \cup \mathit{Comp}^{\mathit{clear}}, \text{ where} \\
\mathit{Comp}^{\mathit{clear}} & := & \{\{\mathit{clear}_{\mathit{main}}, \hat{a}\} \cup (\alpha \setminus a) \mid a \in \alpha \in \mathit{Comp}\}
\end{array}$$

Explanation: Clearly, $f_4(Sys) \in \widetilde{IS}$ holds. Component $n+1$ guarantees $f_4(Sys) \notin GDIS$. The idea of our reduction is as follows: In the beginning *main* offers in any reachable state an action $\mathit{lock}_{\mathit{main}}$, which can participate in the *lock*-interaction which includes all components. As a result, *main* will always be able to participate in an interaction as long as this action is not performed. Now in any reachable state q of *Sys* we want to be able to check whether there is a local deadlock in q. For this purpose in any reachable state $(q_1, \ldots, q_n, q^0_{n+1}, q^0_{main})$, the interaction $\{\mathit{lock}_1, \ldots, \mathit{lock}_n, \mathit{lock}_{n+1}, \mathit{lock}_{\mathit{main}}\}$ can be performed leading to a state where for every $i \in K$ a choice between d_i and $\overline{d}_i$ takes place. Those components j that select d_j form a subset $\tilde{K} \subseteq K$. Then the component *main* will not be able to participate in any further interaction if and only if $\tilde{K}$ is a local deadlock in $(q_1, \ldots, q_n)$ in *Sys*.

Lemma 5.5

$(Sys \in LDIS) \Leftrightarrow ((f_4(Sys), main) \notin Availability)$

Proof:

$\Rightarrow$ If there is a local deadlock in *Sys*, then *main* is not available in $f_4(Sys)$.
Let $\tilde{K}$ be a local deadlock in some reachable state $q = (q_1, \ldots, q_n)$. In $f_4(Sys)$ we perform an interaction sequence from the global starting state $(q_1^0, \ldots, q_n^0, q_{n+1}^0, q_{main}^0)$ to $(q_1, \ldots, q_n, q_{n+1}^0, q_{main}^0)$. Now we perform the *lock*-interaction followed by $\{d_i\}$ for $i \in \tilde{K}$ and $\{\overline{d_i}\}$ for $i \in (K \setminus \tilde{K})$. Finally, we perform the *unlock*-interaction. The components $i \in \tilde{K}$ may not perform any interaction, because they are a local deadlock, even though all components in $(K \setminus \tilde{K})$ offer all their respective actions (and might indeed perform further interactions). As a consequence, *main*'s action $clear_{main}$ will not be enabled, so *main* is "stuck" and will never again be enabled.

$\Leftarrow$ If there is no local deadlock in *Sys*, then *main* is available in $f_4(Sys)$.
As long as we simply perform interactions $\alpha \in C \cup Comp$ in $f_4(Sys)$, *main* will always be enabled. Now, even if we choose in some thus reached global state to perform the lock interaction, *main* will end up enabled again:
The next n transitions have to be the d_i resp. $\overline{d_i}$ connectors. They may be performed in an arbitrary order, but this is of no relevance. Then we have no other choice but to preform the *unlock*-interaction. Let $\tilde{K} \subseteq K$ be the set of components for which we performed the actions d_i. If $\tilde{K} = \emptyset$ we may perform the *all*-interaction and *main* will reach its local state q_{main}^3 and be enabled forever. If $\tilde{K} \neq \emptyset$, remember that $\tilde{K}$ is no local deadlock. This fact, together with the fact that all components in $(K \setminus \tilde{K})$ offer all their actions, means that one or more of the components in $\tilde{K}$ are able to participate in an interaction. As all components of $\tilde{K}$ offer their respective actions a in both forms a and

$\hat{a}$, some $\alpha \in (C^{clear} \cup Comp^{clear})$ is enabled. This means that *main* is enabled as long as we perform interactions in which only components in $(K \setminus \tilde{K})$ participate.
Now, if we perform an interaction $\alpha \in (C^{clear} \cup Comp^{clear})$ again *main* will reach its local state q^3_{main} and be enabled forever.
Note that as a third possibility we might also perform an original interaction $\alpha \in C \cup Comp$ in which at least one of the components in $\tilde{K}$ participates. However, this will result in a local state change of the respective components such that they would afterwards offer their actions $clear_i$, which would again enable *main*. Finally, if an interaction $\{clear_i, clear_{main}\}$ is performed, *main* reaches q^3_{main} and is, as already mentioned, enabled forever.

5.3.5 *Availability is in PSPACE*

In this section, we show that *Availability* is in *PSPACE*. We first prove that the problem's complement is in *NPSPACE*. For this, we give a nondeterministic (polynomially space bounded) algorithm which receives as input a pair (*Sys*,*k*), where $Sys \notin GDIS$ and $k \in K[Sys]$, and outputs "yes" iff k is not available in *Sys*.
Then we invoke *NPSPACE=PSPACE* [Sav70] to deduce that *Availability* is also in *PSPACE*.

Theorem 5.5
Availability is in PSPACE

Let $Sys \in \widetilde{IS}$ and $k \in K[Sys]$. Let $m := max\{|Q_i| \mid i \in K[Sys]\}$. Then, Algorithm 1 - Non-Availability(*Sys*,*k*) (which is nondeterministic) has the possibility to produce the output *yes* iff k is not available in *Sys*. It is obvious that the algorithm works in polynomial space because it contains only six variables for global states respectively integers with upper bound m^n (which can be stored in $\log(m^n) = n \cdot \log(m)$).

Proof of Correctness:

k is not available in *Sys*

$\overset{(1)}{\Leftrightarrow}$ $\exists z \in \{0, \dots, m^n - 1\}\ \exists \alpha_0, \dots, \alpha_z \in C \cup Comp\ \exists q^1, \dots, q^z \in Q$
$\exists j \in \{0, \dots, z\}\ q^0 \xrightarrow{\alpha_0} q^1 \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_{j-1}} q^j \xrightarrow{\alpha_j} \dots \xrightarrow{\alpha_{z-1}} q^z \xrightarrow{\alpha_z} q^j$ and
$\forall i \in \{j, \dots, z\}:\ q^i$ does not enable k

$\overset{(2)}{\Leftrightarrow}$ The call Non-Availability(*Sys*,k) (cf. Algorithm 1) allows for the output "yes"

(1) ($\Rightarrow$) The global transition system is finite. If k is not available, there is a run (i.e., an infinite sequence) and thus a cycle such that in no global state on the cycle an interaction is enabled in which k participates. A shortest reachable cycle with this property is completed after at most m^n steps which is an upper bound for the number of (reachable) global states.

($\Leftarrow$) On the other hand, the existence of any reachable cycle with this property implies the existence of a run in which k is only finitely often enabled, so k is not available.

(2) ($\Leftarrow$) If the algorithm returns *yes*, then *cycle_complete* has at some time been set to *true* while *violation* never was. Furthermore, *cycle_complete* can only be set to *true* within the second while-loop (15-24), which is only performed as long as *interaction_counter* $< m^n + 1$ holds. Interaction counter is incremented once in both while-loops, so we conclude that we have performed both loops at most $m^n + 1$ times altogether. *violation* has never been set to *true* so we have not visited a global state in which k is enabled in the second while-loop. So there is a reachable state (namely the state *cycle_state*

which complies with the state q^j in the formal statement above) from which we may follow edges that build a cycle of states (back to *cycle_state*) such that no state on the cycle enables k.

($\Rightarrow$) Now we assume $\exists z \in \{0, \dots, m^n - 1\}$ $\exists j \in \{0, \dots, z\}$ $\exists q^1, \dots, q^z \in Q$ (pairwise distinct) $\exists \alpha_0, \dots, \alpha_z \in C \cup \mathit{Comp}$ with
$q^0 \xrightarrow{\alpha_0} q^1 \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_{j-1}} q^j \xrightarrow{\alpha_j} \dots \xrightarrow{\alpha_{z-1}} q^z \xrightarrow{\alpha_z} q^j$ and
$\forall i \in \{j, \dots, z\}$ q^i does not enable k.
Then we can obviously perform the transitions $q^0 \xrightarrow{\alpha_0} q^1, \dots, q^{j-1} \xrightarrow{\alpha_{j-1}} q^j$ in the first loop until *present_state*$=q^j$. Only then we would choose to set *cycle_reached* := true.
Thus, we would set *cycle_state*:$=q^j$. By successively performing the transitions $q^j \xrightarrow{\alpha_j} \dots \xrightarrow{\alpha_{z-1}} q^z$ we would then (without setting *violation* to true) obtain *present_state* $= q^z$. Now we choose the transition (q^z, α_z, q^j) and with *next_state* $= q^j =$ *cycle_state* we set *cycle_complete* := true and thereby quit the second loop. We do not quit the loop before because we performed an overall sum of z loop cycle executions in which we incremented *interaction_counter* and we know that $z \leq m^n$. Thus as we quit the second loop, we have *cycle_complete* = *true* and *violation* = *false* and the algorithm returns "*yes*".

Algorithm 1 Non-Availability(Sys, k)

```
1: global_state $present\_state := q^0[Sys]$
2: global_state $cycle\_state$;
3: integer $interaction\_counter := 0$;
4: boolean $cycle\_reached :=$ false;
5: boolean $cycle\_complete :=$ false;
6: boolean $violation :=$ false;
7: while (not $cycle\_reached$) and ($interaction\_counter < m^n + 1$) do
8:     choose an edge ($present\_state$,$\alpha$,$next\_state$);
9:     $interaction\_counter{+}{+}$;
10:    $present\_state := next\_state$;
11:    choose ($cycle\_reached :=$ false) resp. ($cycle\_reached :=$ true);
12: end while
13: $cycle\_state := present\_state$;
14: while (not $cycle\_complete$) and ($interaction\_counter < m^n + 1$) do
15:    if ($present\_state$ enables $k$) then
16:        $violation :=$ true
17:    end if
18:    choose an edge ($present\_state$,$\alpha$,$next\_state$);
19:    $interaction\_counter{+}{+}$;
20:    if $next\_state = cycle\_state$ then
21:        $cycle\_complete :=$ true;
22:    end if
23:    $present\_state := next\_state$;
24: end while
25: if ($cycle\_complete$) and (not $violation$) then
26:    return "$yes$";
27: else
28:    return "$no$";
29: end if
```

Chapter 6

An efficient Approach

6.1 Overview

In the previous chapter, we pointed out that problems like deadlock-detection are not likely to be solvable in polynomial time for interaction systems. We take this as a motivation for an approach based on a polynomial-time computable sufficient condition for deadlock-freedom. The ideas presented in this chapter apply to component-based systems in general and are not restricted to the model of interaction systems, which serves as our means of demonstration.

The general idea of component-based systems is that systems are built from reusable computation units that retain their identity after composition, i.e., they can still be identified, e.g., to be exchanged or modified. This leaves the possibility to build subsystems of a system by decomposing it and then recomposing parts of it while neglecting other parts. If the nature of the model's synchronization pattern includes the idea of multi-party communication (which is the case for interaction systems) where neglecting certain components implies a relaxation of the global behavior we can thus establish subsystems whose transition systems in a way approximate the global transition system.

We start out in Section 6.2 by giving a formal definition of the notion of subsystem and some related ideas. In Section 6.3 we state a basic sufficient condition that investigates subsystems of a parametrized size d but only applies a static (i.e., independent of d) local predicate. In Section 6.4, we generalize the idea of proving global properties by local predicates and thereafter we introduce the *Cross-Checking* technique in Section 6.5 that improves the quality of our state space approximation given by the subsystems. In Section 6.6, we present a more sophisticated sufficient condition for deadlock-freedom. We finally improve this sufficient condition in Section 6.7 by applying another variant of the Cross-Checking idea and speed it up by restricting our investigations to a subset of subsystems in Section 6.8. Along with our considerations, we present examples that point out the applicability of our approach and refer to the appendix for empiric data from our case studies.

6.2 Subsystem Reachability

As described above, our ideas build on the analysis of subsystems of interaction systems (or component-based systems in general). In this section, we formalize the notion of subsystems and along with it establish the terminology that is needed in later sections. For the following definitions, let $Sys = (K, \{A_i\}_{i\in K}, Int, \{T_i\}_{i\in K})$ be an interaction system.

Definition 6.1
Let $K' \subseteq K$ and q be a global state. Then $q \downarrow K' := (q_i)_{i\in K'}$ denotes the **projection** of q to the components in K'. We also use the $\downarrow$-operator to denote projections of projections. For a projected state q' we refer to the components that occur in q' by $K(q')$.

Definition 6.2

For $K' \subseteq K$ let $Q_{K'} = \prod_{i \in K'} Q_i$ be the **state space induced** by K' and $Subs(K') = \bigcup_{K'' \subseteq K'} Q_{K''}$ be the **substates induced** by K'.

Definition 6.3

For an interaction $\alpha \in Int$ and $K' \subseteq K$ we define the **projection** of α to the components in K' by $\alpha \downarrow K' := \alpha \cap (\bigcup_{i \in K'} A_i)$.
For an interaction set Int and $K' \subseteq K$ we define the **projection** of Int to the components in K' by $Int_{K'} := \{\alpha \downarrow K' \mid \alpha \in Int\} \setminus \{\emptyset\}$.

Definition 6.4

Let $K' \subseteq K$. The **subsystem** $Sys_{K'}$ is given by $(K', \{A_i\}_{i \in K'}, Int_{K'}, \{T_i\}_{i \in K'})$. Note that $Sys_{K'}$ accords to our definition of generalized interaction systems[1], so all definitions for interaction systems apply. We refer to the transition relation of the behavior of $Sys_{K'}$ by $\rightarrow_{K'}$.

For the definition of $\rightarrow_{K'}$, we restrict interactions to actions of components in K'. This amounts to assuming (for reachability) that actions of components in $K \setminus K'$ are always available. This definition of a subsystem implies that if a state q is reachable in the global transition system, then for every $K' \subseteq K$ the state $q \downarrow K'$ is reachable in the corresponding subsystem. We formalize this observation in the following lemma.

Lemma 6.1

Let $Sys = (K, \{A_i\}_{i \in K}, Int, \{T_i\}_{i \in K})$.
$q \in Reach(Sys) \Rightarrow \forall K' \subseteq K,\ (q \downarrow K') \in Reach(Sys_{K'})$.

[1]On the other hand, for the the notion of original interaction systems this may not be the case, i.e., for $Sys \in \widetilde{IS}$ there may be $K' \subseteq K$, such that $Sys_{K'} \notin \widetilde{IS}$, because the interaction set of our projection is not necessarily splittable into sets C and $Comp$ that meet the requirements for $\widetilde{IS}$.

Lemma 6.1 implies that the following notion of *extending* a set of substates yields an over-approximation of the reachable global state space. In fact, we will never explicitly compute or store such an extension, because this would ruin the benefits that we obtain by projecting in the first place. Instead, we introduce this notion for formal reasoning.

Definition 6.5
Let q' be a substate. Then $Ext(q', K')$ for $K' \subseteq K$ denotes the set of **extensions** of q' in K' and is defined by $Ext(q', K') = \{q'\} \times \prod_{i \in K' \setminus K(q')} Q_i$. If $K' \subseteq K(q')$ let $Ext(q', K') = \{q'\}$. We say that a substate $\hat{q}'$ is an extension of a substate q' if $K(q') \subseteq K(\hat{q}')$ and $\hat{q}' \downarrow K(q') = q'$.
We overload Ext to also work on sets by $Ext(Q', K') := \bigcup_{q' \in Q'} Ext(q', K')$. Often, we extend substates to the set K of all components, so for ease of notation we define a function f by $f(q') = Ext(q', K)$ resp. $f(Q') = Ext(Q', K)$.

Remark 6.1
The approach that we present in this chapter does not rely on the reachability information that can be derived from a single subsystem. Instead, we build on the combined reachability information of all subsystems of a certain size d. According to our definition of interaction systems in Section 2.3.1 (cf. p. 26) (where we denoted the number of components by n and the maximum size of a component's state space by m), there are $\binom{n}{d}$ such subsystems and m^d is an upper bound for a subsystem's global state space size.
Please note that $\binom{n}{d} \cdot m^d$ is hence an upper bound for the number of states that we have to explore.

Lemma 6.1 together with our definition of the extension function f implies that the intersection of these over-approximations again yields an over-approximation. Although we will not be able to compute this intersection in polynomial time, it will become relevant in Section 6.5 where we will try to approximate it.

Corollary 6.1
$Reach(Sys) \subseteq \bigcap_{K' \subseteq K, \text{ with } |K'|=d} f(Reach(Sys_{K'}))$.

Given a set $K' \subseteq K$ and the induced subsystem $Sys_{K'}$ we denote by the term $Reach(Sys_{K'})[j]$ the set of states that can be reached in $Sys_{K'}$ by a transition sequence in which the last step causes a proper state change for component j.

Definition 6.6
We define the set of states that are **reachable by a proper state change of a component** j in a subsystem $Sys_{K'}$ by

$$Reach(Sys_{K'})[j] := \{q' \in Reach(Sys_{K'}) \mid \exists q \in Reach(Sys_{K'}) \; \exists \alpha \in Int_{K'} \; q \xrightarrow{\alpha} q' \wedge q_j \neq q'_j\}.$$

Similarly to Lemma 6.1, the (global) reachability of a state q by a proper state change of a component j implies that the projection of q is reachable by a proper state change of j in all subsystems $Sys_{K'}$ that include j.

Lemma 6.2
$q \in Reach(Sys)[j] \Rightarrow \forall K' \subseteq K \text{ with } j \in K' : q \downarrow K' \in Reach(Sys_{K'})[j]$

Remark 6.2
For three components i, j, k and $d \geq 3$, there are various (in fact exactly $\binom{|K|-3}{d-3}$) subsystems $Sys_{K'}$ of size d in which these components occur. Due to Lemma 6.2 we may conclude that for a state q that is globally reachable by a proper state change of j, the projection of q to i, j, k is reachable in every subsystem $Sys_{K'}$ with $\{i, j, k\} \subseteq K'$ by a proper state change of j.

We denote the intersection of these subsystems' projections to $\{i, j, k\}$ by $Reach_d(Sys_{i,j,k})[j]$.

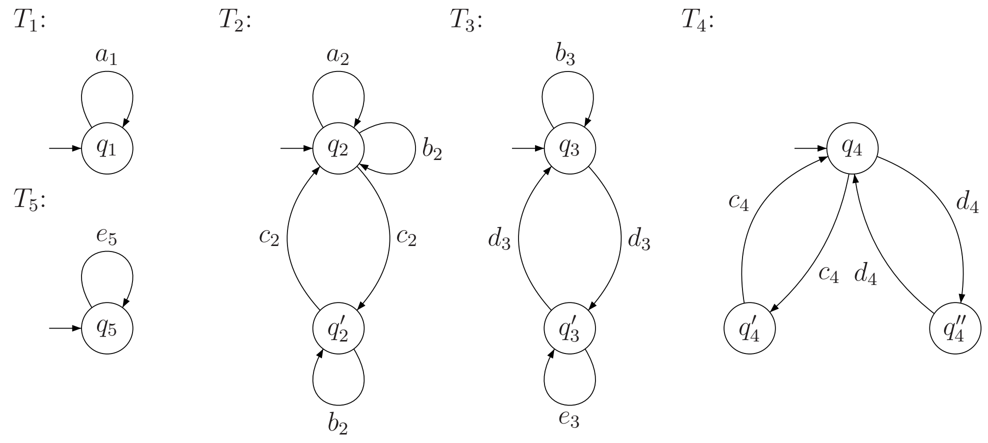


Figure 6.1: The local transition systems T_i for Example 6.1

Definition 6.7

Let $i, j, k \in K$ and $3 \leq d \leq |K|$. Then we define the set of states that are reachable **by component** j **for all subsystems that observe** $i, j, k \in K$ by $Reach_d(Sys_{i,j,k})[j] := \bigcap_{K' \subseteq K, \texttt{s.t.} i,j,k \in K' \wedge |K'| = d} (\text{Reach}(Sys_{K'})[j] \downarrow \{i, j, k\})$.

Example 6.1

We consider a system $Sys = (K, \{A_i\}_{i \in K}, Int, \{T_i\}_{i \in K})$, where $K = \{1, ..., 5\}$, $A_1 = \{a_1\}$, $A_2 = \{a_2, b_2, c_2\}$, $A_3 = \{b_3, d_3, e_3\}$, $A_4 = \{c_4, d_4\}$ and $A_5 = \{e_5\}$. $Int = \{\{a_1, a_2\}, \{b_2, b_3\}, \{c_2, c_4\}, \{d_3, d_4\}, \{e_3, e_5\}\}$. The local transition systems are given in Figure 6.1. Consider the following exemplary reachabilities, where "$-$" stands for an arbitrary state of the corresponding component: $(q_1, q_2', q_3, q_4', q_5) \in \text{Reach}(Sys)[2]$; $\forall j \in K : (-, q_2', q_3', -, -) \notin \text{Reach}(Sys)[\text{j}]$; $(q_1, q_2', q_3') \in \text{Reach}_3(Sys_{\{1,2,3\}})[2]$, whereas $(q_1, q_2', q_3') \notin \text{Reach}_4(Sys_{\{1,2,3\}})[2]$ even though $(q_1, q_2', q_3', q_5) \in Reach(Sys_{\{1,2,3,5\}})[2]$, because $(q_1,\ q_2',\ q_3',\ -) \notin Reach(Sys_{\{1,2,3,4\}})[2]$.

6.3 A basic sufficient Condition

We present a parametrized polynomial-time computable sufficient condition that can confirm local deadlock-freedom. Our algorithm is based on two ideas: Firstly, a necessary condition for the existence of local deadlocks. If a component j is involved in causing a local deadlock in the reachable global state q then there must be two other components satisfying certain properties referring to their respective enabled actions in the state q. This is similar to an idea presented in [AC05] for systems communicating via shared variables. The second idea is to check this predicate on an over-approximation of the set of reachable states: As already mentioned, we consider the states that can be reached by projecting the state space to any subsystem of size d, where d is a parameter of the algorithm (and the degree of the polynomial describing the cost of the algorithm). If local deadlock-freedom cannot be verified, the algorithm reports so, in which case one has to apply other methods to further clarify the situation. We present a nontrivial example where our algorithm confirms deadlock-freedom while a global state space analysis would indeed take exponential time.

Idea:

We investigate the formation of deadlock situations in a system Sys. We assume that Sys is initially deadlock-free[2], i.e., there is no deadlock in the global starting state q^0. We derive a necessary condition for deadlocks that can be checked within subsystems and thus can be used to avoid exponential time complexity. Then, we present the parametrized verification algorithm in pseudo code and a short complexity analysis.

[2]This is a natural assumption w.r.t. reasonable system design. Anyway, a check of this proposition is possible within polynomial time.

Remark 6.3
Given a system Sys, we assume that the $\binom{n}{d}$ subsystems of size d have already been analyzed for reachability and the information (including the reachability of a state q via a proper state change of a component j) is stored in corresponding arrays[3].

Let *Sys* be a system that is initially deadlock-free but contains a reachable deadlock. This implies that in each transition sequence from the global starting state q^0 to a state q with $DL(q)$, there is a first transition $q^1 \xrightarrow{\alpha} q^2$ from a deadlock-free state q^1 to a deadlock-containing state q^2. We will detect this implication (or in other words, this necessary condition) in our subsystems and thus formulate a sufficient condition for deadlock-freedom for the case that no such situation can be detected.
According to Definition 2.18 (p. 29), D is a local deadlock in q if $\forall i \in D \; \forall \alpha \in Int : (ea(q_i) \cap \alpha \neq \emptyset) \Rightarrow (\exists j \in D \; j(\alpha) \not\subseteq ea(q_j))$. We can formulate a locally detectable consequence of a deadlock as described above by defining a binary relation between local states of different components as follows.

Definition 6.8
For components $i, j \in K$ and local states $q_i \in Q_i, q_j \in Q_j$, we say that q_i **waits for** q_j **with respect to** $\alpha \in Int$ if $ea(q_i) \cap \alpha \neq \emptyset$ and $j(\alpha) \neq \emptyset$ but $ea(q_j) \cap \alpha = \emptyset$.

Definition 6.9
For components $i, j \in K$ and local states $q_i \in Q_i, q_j \in Q_j$, we say that q_i **waits for** q_j if $\exists \alpha \in Int$ s.t. q_i waits for q_j with respect to α.

[3] For a more thorough discussion of the complexity of computing the reachability information for the subsystems, see Remark 6.7 (p. 130).

Remark 6.4

Definition 6.9 may be delusive to some extent: It allows for a situation, where for components $i, j \in K$ and local states $q_i \in Q_i, q_j \in Q_j$ we say q_i waits for q_j although on the other hand, there may be some interaction $\{a_i, a_j\}$ with $a_i \in ea(q_i)$ and $a_j \in ea(q_j)$ that i and j could perform together.
In fact, the existence of an interaction α such that q_i waits for q_j with respect to α implies some dependency of i on j whose evaluation should take other aspects (like the existence of other possible synchronizations) into account. Nevertheless, we apply the reduced terminology "q_i waits for q_j" for two reasons: Firstly, it is part of the nature of sufficient conditions to assume the worst case to ease complexity. Secondly, it helps us to keep definitions and propositions that build on Definition 6.9 simple.

Lemma 6.3

Let Sys be an initially deadlock-free interaction system that contains a minimal local deadlock $D \subseteq K$ in some reachable global state $\tilde{q}$. Then $\exists j \in D$ $\exists q \in Reach(Sys)[j]$, such that the following conditions hold:
1) $\forall \alpha \in Int$, s.t. $\alpha \cap ea(q_j) \neq \emptyset$ $\exists k \in D$ such that $k(\alpha) \not\subseteq ea(q_k)$
(For every interaction α, in which j could in its present state participate, j waits for some component $k \in D$ w.r.t. α).
2) $\exists i \in D$ $\exists \alpha \in Int$, s.t. $\alpha \cap ea(q_i) \neq \emptyset \wedge j(\alpha) \not\subseteq ea(q_j)$
(In return, at least one component $i \in D$ waits for j w.r.t. some interaction in which i could in its present state participate.)

Proof:

For a reachable state $\tilde{q}$, there is a global transition sequence $q^0 \xrightarrow{\alpha_1} q^1 \xrightarrow{\alpha_2} ... \xrightarrow{\alpha_r} q^r = \tilde{q}$. Let $q := q^l$, where l is the minimal index with $1 \leq l \leq r$ such that D is a deadlock in q^l. If none of the components in D changed their local states in the transition from q^{l-1} to q, then D would have been a deadlock in the preceding state q^{l-1} already, which would be a contradiction to the choice of

l. Thus, let j be one of the components in D with $q_j^{l-1} \neq q_j$.

It remains to show *Conditions 1)* and *2)*:

Condition 1 follows directly from the definition of deadlocks as $j \in D$. Assume that *Condition 2* does not hold, i.e., j in q_j does not block any other component $i \in D$. Then $D \setminus \{j\}$ would be a deadlock in q in contradiction to our minimality assumption for D.

Definition 6.10

We say that a triple (q_i, q_j, q_k) is a **blocking chain** if i in q_i waits for j in q_j and j in q_j waits for k in q_k.

We relax[4] *Condition 1* by merely demanding the existence of some $\alpha \in Int$. Then we apply Remark 6.2 to formulate the following implication of the necessary condition in Lemma 6.3 that can be observed in subsystems.

Corollary 6.2

Let Sys be an initially deadlock-free interaction system that contains a minimal local deadlock $D \subseteq K$ in some reachable global state q. Then there exist[5] $i, j, k \in D$ and $\hat{q} \in Reach(Sys_{\{i,j,k\}})[j]$, s.t. $\forall\, 3 \leq d \leq n$:
$\hat{q} \in Reach_d(Sys_{\{i,j,k\}})[j]$ and $(\hat{q}_i, \hat{q}_j, \hat{q}_k)$ is a blocking chain.

Algorithm 2 tries to confirm the negation of the necessary condition in the corollary and outputs "Sys is deadlock-free" if and only if it is successful in doing so. Otherwise, it outputs "Sys might contain deadlocks".

Complexity:

[4]Please note that according to our definition of interaction systems, every local state has to enable at least one action and every action must occur in at least one interaction. Thus, demanding the existence of an interaction as described above is indeed a relaxation (also cf. Remark 2.6, p. 29)

[5]where possibly $i = k$ may hold

Algorithm 2 Deadlock-Freedom Verification(Sys, d)

```
1: for all $i, j, k \in K$ do
2:     for all $(q_i, q_j, q_k) \in \text{Reach}_d(Sys_{\{i,j,k\}})[j]$ do
3:         if $(q_i, q_j, q_k)$ is a blocking chain then
4:             write("Sys might contain deadlocks");
5:             break;
6:         end if
7:     end for
8: end for
9: write("Sys is deadlock-free")
```

We assume that before the application of Algorithm 2, a preprocessing is performed, that creates for every pair $(i, j) \in K \times K$ an $(m \times m)$-matrix $wait_{i,j}$ such that $wait_{i,j}[q_i, q_j] = 1$ if q_i waits for q_j and $wait_{i,j}[q_i, q_j] = 0$, otherwise. Given such a collection of matrices, the check, whether a state (q_i, q_j, q_k) is a blocking chain (line 3) comes down to looking up two values in a collection of matrices and thus can be performed in $O(1)$.
The outer loop (1-8) of Algorithm 2 is performed (up to) n^3 times. The inner loop (2-7) considers (up to) m^3 substates for each of (up to) n^{d-3} subsystems. Thus, we have an overall complexity of $O(m^3 \cdot n^d)$.

Examples

In the following, we apply Algorithm 2 to two example systems. We verify deadlock-freedom for a complex parametrized example system (cf. Example 6.2) that synchronizes triples of neighbors to perform trilaterations. Algorithm 2 is able to handle the example, even for arbitrarily large system size parameters by investigating subsystems of size $d = 3$. We also give an example system Sys that can be proven deadlock-free with $d = 4$ but not with $d = 3$ to display the limitations of Algorithm 2.

Remark 6.5

Trilateration is a method for determining the intersections of three sphere surfaces given the centers and radii of the three spheres. To accurately and uniquely determine the relative location of an object on a surface using trilateration, three reference points (in this case the vertices of the triangle surrounding the object) are needed.

Let us imagine a system of n transmitting stations that divide a surface into triangles, using an odd number y of rows and an odd number x of columns (cf. Figure 6.2). Three transmitting stations that form a triangle can cooperate in order to determine the position of an object within the triangle.

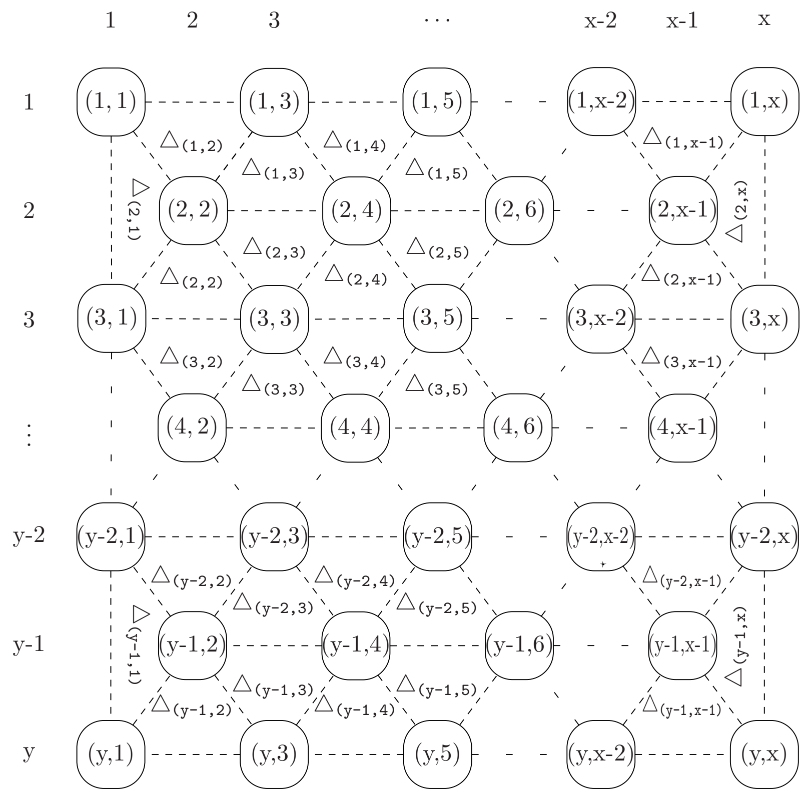


Figure 6.2: An area divided into triangles $\triangle_{(a,b)}$ by transmitting stations (u, v)

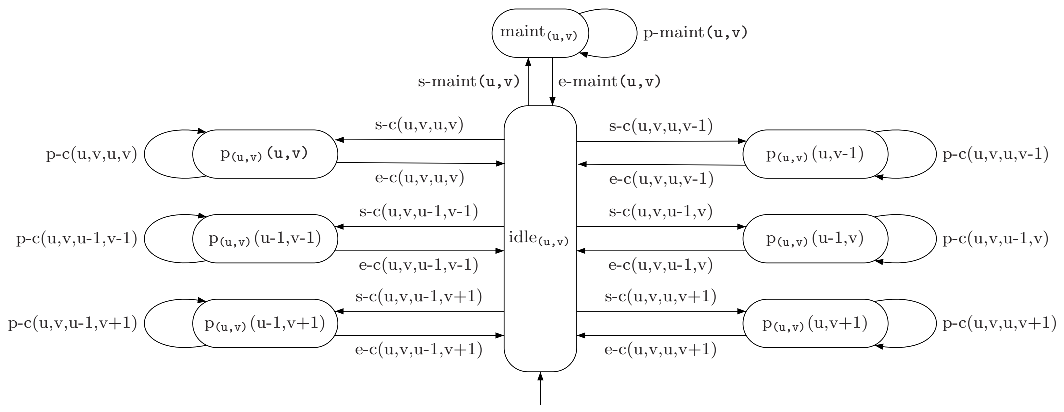


Figure 6.3: The local transition system for some (non-border) transmitting station (u, v)

This means, every transmitting station (u, v) can participate in a job (i.e., a trilateration) in one of its (up to) six adjacent triangle-areas at a time or participate in a maintenance synchronization together with the other ($\lfloor x/2 \rfloor$ or $\lfloor x/2 \rfloor - 1$) stations on the same horizontal line. Each transmitting station is a component (u, v) in our model and offers actions of type start, perform and end a cooperation in a triangle (a, b), which are abbreviated by s-c(u, v, a, b), p-c(u, v, a, b) and e-c(u, v, a, b), respectively. Also, each component (u, v) offers actions to start, perform and end a maintenance, which are abbreviated by s-maint(u, v), p-maint(u, v) and e-maint(u, v), respectively.

Example 6.2
The system is described by $Sys(y, x) = (K, \{A_{(u,v)}\}_{(u,v)\in K}, Int, \{T_{(u,v)}\}_{(u,v)\in K})$, where:
$K = \{(2u + 1, 2v + 1) \mid 0 \leq u \leq \frac{y-1}{2}, 0 \leq v \leq \frac{x-1}{2}\}$
$\quad \cup \{(2u, 2v) \mid 1 \leq u \leq \frac{y-1}{2}, 1 \leq v \leq \frac{x-1}{2}\}$
$A_{(u,v)} = \{$s-c(u, v, a, b),p-c(u, v, a, b),e-c$(u, v, a, b) | \triangle_{(a,b)}$ is a triangle adjacent

to $(u, v)\}$ $\cup$ $\{$s-maint(u, v),p-maint(u, v),e-maint$(u, v)\}$

Int : For each *op* $\in$ $\{$s-c,p-c,e-c$\}$ we include the interactions $\{op(u_1, v_1, a, b), op(u_2, v_2, a, b), op(u_3, v_3, a, b)\}$, where $(u_1, v_1), (u_2, v_2), (u_3, v_3)$ are vertices of $\triangle_{(a,b)}$.
Also, for *op* $\in$ $\{$s-maint,p-maint,e-maint$\}$, we include the interactions $\{op(u_1, 1), op(u_1, 3), ..., op(u_1, x)\}$, and $\{op(u_2, 2), op(u_2, 4), ..., op(u_2, x - 1)\}$ where u_1 resp. u_2 ranges over the odd resp. even numbers in $\{1, ..., y\}$.

The $T_{(u,v)}$'s are depicted in Figure 6.3. Note that the transmitting stations at the border of the area do not have 6 but less triangles to participate in, so Figure 6.3 is exemplary only.

In the following, we prove that the algorithm is indeed able to verify (for arbitrarily large x, y) that Example 6.2 is deadlock-free, by showing that no subsystem $Sys_{\{i,j,k\}}$ of components $i, j, k \in K$ will ever reach a state (q_i, q_j, q_k), such that (q_i, q_j, q_k) is a blocking-chain:

Remark 6.6
Let a component $l_1 \in \{i, j, k\}$ be in its maint$_{l_1}$- (or in its p$_{l_1}$(a,b)-) state. In this case, l_1 offers its p-maint and e-maint (or p-c(a,b) and e-c(a,b)) actions. For $l_2 \in \{i, j, k\}$ to block l_1, l_2 must possess an action that occurs in an interaction together with one of the actions offered by l_1, i.e., l_2 has to share a line with l_1 (or be one of the vertices of $\triangle_{(a,b)}$). However, as l_2 is observed in $Sys_{\{i,j,k\}}$ it must have moved to its maint$_{l_2}$- (or to its $p_{l_2}(a, b)$-) state conjointly with l_1. Thus, it offers the demanded action.

Now, assume that there is a state (q_i, q_j, q_k) that is a blocking-chain:

Due to Remark 6.6, we may assume $q_i = \text{idle}_i$. But then, for q_j to block q_i, we have $q_j \neq \text{idle}_j$. But by Remark 6.6, we know that j in $q_j \neq \text{idle}_j$ cannot be blocked by k in any q_k.

We showed that our algorithm verifies deadlock-freedom for the trilateration example in polynomial time. Note, that the example is a non-trivial system that could easily be modeled to contain deadlocks, e.g., $(3,3)$,$(2,4)$,$(3,5)$ could wait for each other when $(3,3)$ is in a state where it wants to do a job in $\triangle_{(2,3)}$ while $(2,4)$ wants to do a job in $\triangle_{(2,5)}$ and $(3,5)$ wants to do a job in $\triangle_{(3,4)}$. So firstly, it is not obvious by specification that the implementation is deadlock-free. Secondly, the number of reachable global states of the system is exponential (in n). Hence, any algorithm that checks some condition for every global state would need time exponential in n. Thirdly, the system scale is variable and it may contain arbitrarily large interactions (the maintenance interactions' size is linear in x). Nevertheless, to verify deadlock-freedom it suffices to choose the parameter $d = 3$, i.e., to observe subsystems of size 3 only.

Example 6.3
We are now going to investigate an example of a deadlock-free system Sys, for which our algorithm is not able to confirm deadlock-freedom when we observe subsystems of size 3. However, when observing subsystems of size 4, the algorithm yields the desired result.
Let us again consider Example 6.1 (p. 118), that was introduced at the end of section 6.2. When observing the subsystem $Sys_{\{1,2,3\}}$, we find $(q_1, q_2', q_3') \in Reach_3(Sys_{1,2,3})[2]$ where 1 is blocked by 2, which is, in turn, blocked by 3. However, no corresponding global state is reachable in the global system because the communication with component 4 prevents 2 and 3 from reaching q_2' and q_3' simultaneously.
The problem, of course, is the lack of observation of component 4. If we apply the algorithm with $d = 4$, we are indeed able to verify deadlock-freedom: The relation $R = \{(q_1, q_2'), (q_2, q_3'), (q_2, q_4''), (q_2', q_4''), (q_2', q_3'), (q_3, q_4'), (q_3', q_4'), (q_5, q_3)\}$ includes all pairs (q_i, q_j), where i in q_i is blocked by j in q_j. As a result, the set of possible blocking chains is $BC = \{(q_1, q_2', q_4''),\ (q_1, q_2', q_3'),\ (q_2, q_3', q_4'),$

(q'_2, q'_3, q'_4), $(q_5, q_3, q'_4)\}$. As stated at the end of section 6.2, $(q_1, q'_2, q'_3) \notin$ $\mathit{Reach}_4(\mathit{Sys}_{1,2,3})[2]$. Corresponding propositions hold for all other states in BC. The example also displays that it can be crucial to check whether a state is reachable by a state transition that affects a certain component:
Note that (q_5, q_3, q'_4), where 5 is blocked by 3 and 3 is blocked by 4 is indeed reachable in both subsystems of size 4 that include the components 3, 4 and 5, but it is *not* reachable by an interaction that causes a local state change of the middle component 3, so the state's reachability alone will not affect the algorithm's success.

The presented algorithm is (even with $d = 3$) able to handle the complex trilateration system (cf. Example 6.2) regardless of the choice of the parameters x, y. This means it can handle arbitrarily large synchronizations and a reachable state space of exponential size. Our algorithm profits from (cf. Example 6.2), but is not dependent on (cf. Example 6.1 with $d = 4$) symmetric constructs. Sys displays that the problem of "inherent information" can prevent our algorithm from verifying deadlock-freedom. This fact is, of course, not surprising w.r.t. the complexity results established so far. Nevertheless, the existence of non-trivial examples that cannot be verified in polynomial time by algorithms based on global state space exploration displays the benefit of the presented algorithm.

6.4 Generalizing sufficient Conditions based on local Predicates

In the previous section, we gave a sufficient condition for deadlock-freedom that was based on local predicates. While in this thesis we concentrate on approaches for deadlock detection, we still want to point out that other properties could be treated in an analogous manner. In this section, we generalize and to some extent formalize the presented approach.

Properties of interaction system are often defined to be equivalent to a certain predicate P (on global states) being valid on all states[6] in *Reach(Sys)*, i.e., $Prop(Sys) := \forall q \in Reach(Sys) P(q)$. As a consequence, the standard way to prove such a property is to compute the reachable global state space and check P for every global state therein. However, due to state space explosion, this is exponentially expensive (in the number of components) in non-trivial interaction systems. To avoid this exponential blow-up, we suggest to prove properties on ("compressed") over-approximations.

Remark 6.7
In Remark 6.3 (p. 120), we assume that the complexity analyses for the various subsystems have already been performed. Analogously, we assume for the generalization presented here that we may access for each $K' \subseteq K$ with $|K'| = d$ an array reach($Sys_{K'}$) (cf. Definition 6.11) that stores the reachability information Reach($Sys_{K'}$).
We want to emphasize that the costs of such an analysis would be polynomial in n and m. The exact costs will significantly depend on the parameter d which determines the polynomial degree of our asymptotic time bounds. To be more specific, if we abstract from the time costs[7] that are needed to check whether an interaction is enabled and which successor states can be reached by different interactions the analyses of all subsystems with d components can be performed in $O(\binom{n}{d} \cdot m^d)$ (i.e., the sum of the sizes of the state spaces,

[6]E.g., we call a system deadlock-free if and only if every reachable global state does not contain a deadlock.

[7]Our motivation for this abstraction is the fact that – depending on the data structures that we use to represent an interaction system – the costs for a reachability analysis of a system may vary. Also, as the benefit of our approach lies within the decrease of a time bound that is exponential in n to a polynomial with degree d, we are not interested in additional factors (like, e.g., $|Int|$) that occur analogously in both time bounds.

cf. Remark 6.1, p. 116).

It is an important aspect of our approach not to exceed these time bounds in any follow-up technique as increasing the degree of our overall time bounds would allow us to increase the parameter d instead, which may significantly enhance our approximation quality.

Definition 6.11

For reasons of efficient implementation of the algorithms presented in this chapter, we define for a subsystem $Sys_{K'}$ a boolean **reachability array** $reach(Sys_{K'})$ of length $|Q_{K'}|$ that will be initialized with the characteristic function of $Reach(Sys_{K'})$ w.r.t. $Q_{K'}$.

We will identify each position of $reach(Sys_{K'})$ with a state in $q' \in Q_{K'}$. We denote the access to the boolean that corresponds to q' by $reach(Sys_{K'})[q']$ where after our subsystem reachability analyses we have $reach(Sys_{K'})[q'] =$ true iff $q' \in Reach(Sys_{K'})$. It is easy to implement our arrays in such a way that the index of a state q' (of length d) can be computed in $O(d)$.

For ease of notation, we will (in our theoretic considerations) treat $reach(Sys_{K'})$ like a set and write "$q' \in reach(Sys_{K'})$" iff $reach(Sys_{K'})[q'] =$ true.

Remark 6.8

Our decision to represent $Reach(Sys_{K'})$ as an array (compared to a list representation) yields a worse lower space complexity bound, namely $\Omega(|Q_{K'}|)$ (compared to $\Omega(|Reach(Sys_{K'})|)$) but a better time bound for reachability look-ups, namely $O(1)$. The ability to efficiently look up the boolean $reach(Sys_{K'})[q']$ for a substate q' in our array representation will be important for our Cross-Checking implementations, namely Algorithms 3 (p. 139) and 5 (p. 158).

Idea:

The basic idea we want to apply is to prove a predicate P for a global state q by proving a predicate P' for each projection q' of q (of a certain size) such that $P(q)$ is implied. For a single state q, this might not be reasonable, because the workload of checking P' for the various projections may be larger than the workload of checking $P(q)$ directly.

However, we may exploit this idea in order to prove a property $Prop(Sys)$ in polynomial time by proceeding in three steps as follows:

- We choose a parameter $d \ll n$ and calculate the reachable states for each subsystem with d components. Each reachable substate $q' = (q_{i_1}, \ldots, q_{i_d})$ is a compact representation of $Ext(q', K)$.
- We formulate a local indicator predicate P' (checkable in time polynomial in n), i.e., a predicate on local states, such that the validity of P on a global state q is implied by the validity of P' on the projections of q (cf. Definition 6.12).
- We prove P' for all $q' \in Reach(Sys_{K'})$ to derive the validity of P on all reachable global states (cf. Lemma 6.4):
  $[\forall K' \subseteq K, |K'| = d \; \forall q \in Reach(Sys) : \; P'(q \downarrow K')] \Rightarrow P(Reach(Sys))$

Definition 6.12
Given a predicate P on global states, we call a predicate P' a **local indicator** if (for every $d \geq 3$) the validity of P on a global state q is implied by the validity of P' on every projection of q to d components:
$[\forall K' \subseteq K, |K'| = d \; P'(q \downarrow K')] \Rightarrow P(q)$

Definition 6.13
Let *Sys* be an interaction system, $Q' \subseteq Q$ a subset of the global state space and $K' \subseteq K$ a subset of the components. We call a set $A_{K'} \subseteq Q_{K'}$ an **approximation for Q' with respect to** K' if the projection (to K') of every state in Q' is in $A_{K'}$, i.e., if $q \in Q' \Rightarrow q \downarrow K' \in A_{K'}$.

The idea behind the approach sketched above is formalized in the following Lemma 6.4, where for the time being it is convenient to substitute the sets $A_{K'}$ by Reach($Sys_{K'}$), which is clearly an approximation according to Definition 6.13.

Lemma 6.4

Assume that we may access for each $K' \subseteq K$ with $|K'| = d$ an approximation $A_{K'} \subseteq Q_{K'}$ for Reach(Sys) w.r.t. K'. Also, let P' be a local indicator for P. Then the validity of P' on all substates in all $A_{K'}$ implies the validity of P on all states in Reach(Sys), i.e.:

$[\forall K' \subseteq K, |K'| = d\ \forall q' \in A_{K'}\ P'(q')] \Rightarrow P(Reach(Sys))$

Proof:

$$\begin{aligned} & \forall K' \subseteq K, |K'| = d\ \forall q' \in A_{K'}\ P'(q') \\ \overset{(1)}{\Rightarrow} \quad & \forall q \in Reach(Sys) \forall K' \subseteq K, |K'| = d\ P'(q \downarrow K') \\ \overset{(2)}{\Rightarrow} \quad & \forall q \in Reach(Sys) P(q) \end{aligned}$$

Clearly, (1) holds as $A_{K'}$ is an approximation of Reach(Sys) in K' and (2) holds as P' is a local indicator for P.

Complexity:

Lemma 6.4 allows us to abstain from handling global states explicitly. Instead, we may now prove predicates on global states by the over-approximations $Reach(Sys_{K'})$. The advantage of this approach is immense: Instead of a complexity that is exponential in n, we have a complexity that is polynomial (with degree d) in n and m.

Remark 6.9

There is one major drawback to our present approach:
Considering subsystems with $d \ll n$ components neglects a lot of information. Indeed, there will be many substates that are marked reachable in our subsystem reachability tables although they do not originate by projection from a globally reachable state. We call such substates "artifacts". If we

check condition P' on many such artifacts we run the risk that P' is violated and we cannot conclude P. We deal with this problem in the following section.

6.5 Cross-Checking for Reachability

In Section 6.3, we saw that the choice of the parameter d greatly influences the result of our approach. This is not surprising, as there is an obvious trade-off between the accuracy of our reachable state space approximations and the time needed to compute them. Given the fact that even polynomial-time algorithms are hardly applicable in practice for large inputs once the polynomial's degree exceeds 6 or 7, one may on the other hand deduce that given a certain interaction system and certain hardware, the degree of our polynomial is fixed. We take this observation as a motivation to try and improve the results that can be obtained by observing subsystems of a certain size *without* raising the degree (i.e., d in our case) of the polynomial time bound. In this section, we present the Cross-Checking idea that enhances the approximation quality without exceeding the polynomial time bound given by the reachability analyses.

Let us consider an interaction system that models Tanenbaum's solution [Tan08] to Dijkstra's Dining Philosophers problem. Tanenbaum suggests that each of the philosophers is provided with a separate semaphore that she has to set in order to leave her thinking state. A semaphore however can only be set if its "neighbor" semaphores are unset. Once a philosopher has eaten, she puts back the forks and resets her semaphore. This can be considered an elegant solution as it is symmetric and allows for maximum efficiency (meaning that it still allows for a global state where every second philosopher is in her eating state). On the other hand, this is a deadlock-free system with a natural interaction structure whose reachable global state space is exponential in the number of philosophers.

This solution can be modeled as an interaction system as follows, where p is the number of philosophers:

Example 6.4
$DP(p) = (K(p), \{A_i\}_{i \in K(p)}, \mathit{Int}(p), \{T_i\}_{i \in K(p)})$, where
$K(p) = \{\mathit{Phil}_0, \ldots, \mathit{Phil}_{p-1}, \mathit{Fork}_0, \ldots, \mathit{Fork}_{p-1}, \mathit{Sem}_0, \ldots, \mathit{Sem}_{p-1},\}$,
$\mathit{Int}(p) = \bigcup_{0 \le i \le p-1} \{\{\mathit{pickleft}_{\mathit{Phil}_i}, \mathit{occupy}_{\mathit{Fork}_i}\}, \{\mathit{pickright}_{\mathit{Phil}_i}, \mathit{occupy}_{\mathit{Fork}_{i-1}}\},$
$\{\mathit{priority}_{\mathit{Phil}_i}, \mathit{down}_{\mathit{Sem}_i}, \mathit{allow}_{\mathit{Sem}_{i-1}}, \mathit{allow}_{\mathit{Sem}_{i+1}}\},$
$\{\mathit{drop}_{\mathit{Phil}_i}, \mathit{up}_{\mathit{Sem}_i}, \mathit{vacate}_{\mathit{Fork}_{i-1}}, \mathit{vacate}_{\mathit{Fork}_i}\}\}$,
where calculation is modulo p, and the local behaviors T_i and (implicitly the) port sets A_i are given in Figure 6.4. For better readability we use the index i instead of Phil_i, Fork_i, respectively Sem_i.

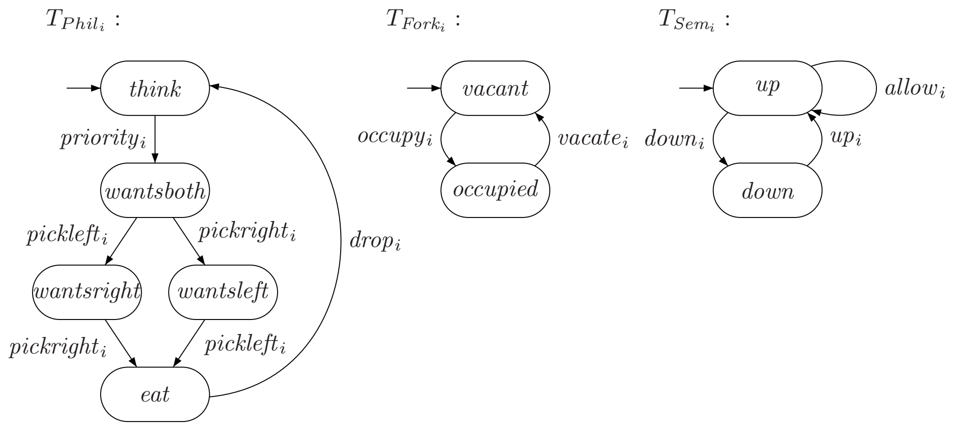


Figure 6.4: Tanenbaum's Dining Philosophers: Local transition systems

Example 6.5
For the dining philosophers example $DP(6)$ (i.e., $|K| = 18$) and $d = 4$ the sum of the sizes of the investigated substate spaces is 229,095 compared

to 64mio global states in the original system. Obviously, the advantage is much greater for a larger number of philosophers as the global state space grows exponentially in n while the sum of the sizes of the state spaces of the subsystems grows only polynomially (with degree d) in n (cf. Remark 6.1, p. 116).

Example 6.6
As already mentioned in the previous section, we have to deal with the problem of artifacts. For the dining philosophers example $DP(6)$ (i.e., $|K| = 18$) and $d = 4$, only 43,212 of the 229,095 states in the state spaces of the subsystems are unreachable, which corresponds to 18.85% (cf. Figure B.1, p. 200 Appendix).

Here, we introduce Cross-Checking as a technique to eliminate artifacts. Corollary 6.1 (p. 117) already pointed out that the straightforward approach to make use of multiple over-approximations would be computing (for some previously defined d) the set $\mathit{Intersection} := \bigcap_{K' \subseteq K, \text{ with } |K'|=d} f(\mathit{Reach}(\mathit{Sys}_{K'}))$ and thus distilling all the information that is available from our subsystem reachability analyses. However, in our case this is simply not feasible because firstly it seems to require the computation of the various sets $f(\mathit{Reach}(\mathit{Sys}_{K'}))$ (yielding an exponential blow-up) and secondly (even if there was a sophisticated way to avoid the application of f) the cardinality of *Intersection* is larger or equal than the cardinality of *Reach(Sys)* which we wanted to avoid in the first place.

Idea:
So basically we are bound to keep to our "encoded" approximations. Thus, we have to accept the fact that a single reachability table $\mathit{reach}(\mathit{Sys}_{K'})$ can (in the sense of its interpretation via f) only *exclude* the reachability of a

global state q by setting $reach(Sys_{K'})[q \downarrow K'] = false$. On the other hand, we want to maintain our over-approximation property, i.e., we may only do so if no global state in q's equivalence class w.r.t. K', namely $[q]_{K'} := f(q \downarrow K')$ is in *Reach(Sys)*. This implies that the smallest over-approximation of *Intersection* that can be encoded (w.r.t. to application of f) in a subset of $Q_{K'}$ is $f(Intersection \downarrow K')$.

Thus, we are interested in the set $Intersection \downarrow K'$ which reflects – in the sense just discussed – the very best piece of information that can be gathered (in the time available) from the information available so far. We use the following observations in order to receive an over-approximation of $Intersection \downarrow K'$:

$$\begin{array}{rl}
 & Intersection \downarrow K' \\
= & [\bigcap_{\tilde{K}\subseteq K, |\tilde{K}|=d} f(Reach(Sys_{\tilde{K}}))] \downarrow K'. \\
\supseteq & [\bigcap_{\tilde{K}\subseteq K, |\tilde{K}|=d} f(Reach(Sys_{\tilde{K}}) \downarrow K')] \downarrow K'. \\
=^1 & [\bigcap_{\tilde{K}\subseteq K, |\tilde{K}|=d} f(Ext(Reach(Sys_{\tilde{K}}) \downarrow K', K'))] \downarrow K'. \\
=^2 & f[\bigcap_{\tilde{K}\subseteq K, |\tilde{K}|=d} Ext(Reach(Sys_{\tilde{K}}) \downarrow K', K')] \downarrow K'. \\
=^3 & \bigcap_{\tilde{K}\subseteq K, |\tilde{K}|=d} Ext(Reach(Sys_{\tilde{K}}) \downarrow K', K'). \\
=^4 & \bigcap_{\tilde{K}\subseteq K, |\tilde{K}|=d, \tilde{K}\cap K'\neq\emptyset} Ext(Reach(Sys_{\tilde{K}}) \downarrow K', K'). \\
=: & Reach'(Sys_{K'})
\end{array}$$

The proofs of the relations $\supseteq$ and $=^1$ to $=^4$ are as follows:

$\supseteq$ as for all $\tilde{K}, K' \subseteq K$, $\tilde{Q} \subseteq Q_{\tilde{K}}$ we have $f(\tilde{Q} \downarrow K') \supseteq f(\tilde{Q})$.
Also, the operators intersection and projection preserve the subset-relation.

$=^1$ as for $K'' \subseteq K'$ and $Q'' \subseteq Q_{K''}$, $f(Q'') = f(Ext(Q'', K'))$

$=^2$ as for $Q_1, Q_2, \ldots, Q_k \subseteq Q_{K'}$,
$$\begin{array}{rcl}
\bigcap_{1\leq i\leq k} f(Q_i) & = & \bigcap_{1\leq i\leq k}(Q_{K\setminus K'} \times Q_i) \\
 & = & Q_{K\setminus K'} \times \bigcap_{1\leq i\leq k} Q_i \\
 & = & f(\bigcap_{1\leq i\leq k} Q_i)
\end{array}$$

$=^3$ as $\bigcap_{\tilde{K}\subseteq K,|\tilde{K}|=d} \mathit{Ext}(\mathit{Reach}(\mathit{Sys}_{\tilde{K}}) \downarrow K', K') \subseteq Q_{K'}$ and $(Q' \subseteq Q_{K'}) \Rightarrow (f(Q') \downarrow K' = Q')$

$=^4$ as for $\tilde{K}$ with $\tilde{K} \cap K' = \emptyset$ $\mathit{Reach}(\mathit{Sys}_{\tilde{K}}) \downarrow K' = \{()\}$ and thus $\mathit{Ext}(\mathit{Reach}(\mathit{Sys}_{\tilde{K}}) \downarrow K', K') = Q_{K'}$.

Definition 6.14
$\mathit{Reach}'(\mathit{Sys}_{K'}) := \bigcap_{\tilde{K}\subseteq K,|\tilde{K}|=d,\tilde{K}\cap K'\neq\emptyset} \mathit{Ext}(\mathit{Reach}(\mathit{Sys}_{\tilde{K}}) \downarrow K', K')$.

Besides correctness, another important property of an over-approximation is its quality, so we try to give the reader an idea to which degree our method comprises resp. neglects information: We were able to abstain from computing f by projecting reachable state spaces of the various subsystems to K', which accounts to restricting our view to explicit information about reachability for the components in K'. As mentioned above, we may only mark a state q' unreachable in $\mathit{Reach}'(\mathit{Sys}_{K'})$ if we are sure that $f(q') \cap \mathit{Reach}(\mathit{Sys}) = \emptyset$. This knowledge may either originate from the fact that the reachability analysis of some subsystem yields that no extension of a substate of q' is reachable (in which case our approximation covers that knowledge) or it may originate from the fact that there are different subsystems by which we derive the non-reachability of different states in $f(q')$ such that the whole set $f(q')$ is covered (in which case our approximation does not cover that knowledge).

Remark 6.10
We want to compute, for each $K' \subseteq K, |K'| = d$, the set $\mathit{Reach}'(\mathit{Sys}_{K'})$. The direct approach to do this would mean looping over all subsystems K' and for each K' loop over all subsystems $\tilde{K}$. As this would yield at least costs $\Omega(\binom{n}{d} \cdot \binom{n}{d})$ it would raise the degree of the polynomial that hitherto was an upper complexity bound. This in turn gives rise to the (legitimate) question

if it would not be more suitable to observe subsystems of size $2d$ in the first place and forget about this enhancement technique.
However, we will show that we can compute the various sets $Reach'(Sys_{K'})$ in strictly less time than it would take to even increment parameter d to $d+1$, i.e., our approach can be performed in $o(\binom{n}{d+1} \cdot m^{d+1})$ (cf. Remark 6.7, p. 130). The computation is performed by Algorithm 3 (p. 139) and we refer to this method by reachability Cross-Checking.

Explanation of Algorithm 3:

For reasons of efficiency, Algorithm 3 does not loop over the various sets $Reach(Sys_{K'})$ and the therein reachable substates but rather over all states in $\{q'' \in \mathit{Subs(K)} \mid |q''| < d\}$. Looping over these substates is realized via the three outer for-loops (Line 1, Line 2 and Line 3). For a state q'', we decide (by looking up the reachability flags of its extensions in the various tables $reach(Sys_{K'})$) whether its reachability can be refuted (Lines 5 to 11). If this is the case, we set all reachability flags of all extensions of q'' to false (Lines 13 to 17).

Example 6.7
Let $K' = \{Phil_1, Phil_2, Fork_1, Fork_2\}$. In $DP(6)_{K'}$, we are able (by performing the interactions $\{priority_1\}$ and $\{priority_2\}$) to reach the substate $q' = (priority_{Phil_1}, priority_{Phil_2}, vacant_{Fork_1}, vacant_{Fork_2})$. However, if we consider the projection $q'' = (priority_{Phil_1}, priority_{Phil_2})$ of q' and its occurrence in the subsystem that is induced by $K'' = \{Phil_1, Phil_2, Sem_1, Sem_2\}$ we learn that no extension of q'' is in $Reach(DP(6)_{K''})$. Thus, we can remove q' from $Reach(DP(6)_{K'})$ still preserving the fact that the set is an approximation for $Reach(DP(6))$ w.r.t. K'.

For $p = 6$ and $d = 4$, after the first application of Cross-Checking for the subsystem reachabilities, we will have marked 147,561 of the 229,095 substates unreachable. This corresponds to 64.41% (cf. Figure B.1, p. 200 Appendix).

Algorithm 3 Reachability Cross-Checking (Sys, d)

```
for $x := 1$ to $(d-1)$ do
    for all subsets $K'' = \{i_1, \ldots, i_x\}$ of $K$ do
        for all $q'' = (q_{i_1}, \ldots, q_{i_x}) \in Q_{K''}$ do
            reachable := true;
            for all subsystems $Sys_{K'}$ with $K'' \subseteq K'$ (and $|K'| = d$) do
                occurrence := false;
                for all $q' \in Ext(q'', K')$ do
                    occurrence := occurrence OR $reach(Sys_{K'})[q']$;
                end for
                reachable := reachable AND occurrence;
            end for
            if reachable = false then
                for all subsystems $Sys_{K'}$ with $K'' \subseteq K'$ (and $|K'| = d$) do
                    for all $q' \in Ext(q'', K')$ do
                        $reach(Sys_{K'})[q')]$ := false;
                    end for
                end for
            end if
        end for
    end for
end for
```

Lemma 6.5

Algorithm 3 – Reachability Cross-Checking – computes the sets $Reach'(Sys_{K'})$ for all subsystems $Sys_{K'}$ with d components in an overall amount of time that is in $O(d \cdot n^d \cdot m^d)$.

Proof:

Correctness of Algorithm 3 - *reachability Cross-Checking*:

Here, we show that after the application of Algorithm 3, we have $reach'(Sys_{K'}) = Reach'(Sys_{K'})$ for each subsystem $Sys_{K'}$.

$Reach'(Sys_{K'}) = \bigcap_{\tilde{K}\subseteq K, |\tilde{K}|=d, \tilde{K}\cap K'\neq\emptyset} Ext(Reach(Sys_{\tilde{K}}) \downarrow K', K')$ projects the various sets $Reach(Sys_{\tilde{K}})$ to K', extends them to K', and builds the intersection. In other words, a state $q' \in reach(Sys_{K'})$ is removed from $reach(Sys_{K'})$ if and only if there is a subsystem $Sys_{\tilde{K}}$ whose projection's extension does not include q'. This however is the case if and only if no state in $Reach(Sys_{\tilde{K}})$ is an extension of a projection of q'.

More formally put:

$$\begin{aligned}
& Reach'(Sys_{K'}) \\
= \quad & \textstyle\bigcap_{\tilde{K}\subseteq K, |\tilde{K}|=d, \tilde{K}\cap K'\neq\emptyset} Ext(Reach(Sys_{\tilde{K}}) \downarrow K', K') \\
= \quad & Reach(Sys_{K'}) \cap \textstyle\bigcap_{\tilde{K}\subseteq K, |\tilde{K}|=d, \tilde{K}\neq K', \tilde{K}\cap K'\neq\emptyset} Ext(Reach(Sys_{\tilde{K}}) \downarrow K', K') \\
=^1 \quad & Reach(Sys_{K'}) \setminus \textstyle\bigcup_{\tilde{K}\subseteq K, |\tilde{K}|=d, \tilde{K}\neq K', \tilde{K}\cap K'\neq\emptyset} \overline{Ext(Reach(Sys_{\tilde{K}}) \downarrow K', K')} \\
=^2 \quad & Reach(Sys_{K'}) \setminus \textstyle\bigcup_{\tilde{K}\subseteq K, |\tilde{K}|=d, \tilde{K}\neq K', \tilde{K}\cap K'\neq\emptyset} Ext(\overline{Reach(Sys_{\tilde{K}}) \downarrow K'}, K') \\
=^3 \quad & Reach(Sys_{K'}) \setminus \textstyle\bigcup_{\tilde{K}\subseteq K, |\tilde{K}|=d, \tilde{K}\neq K', \tilde{K}\cap K'\neq\emptyset} Ext(Refutable(K', \tilde{K}), K')
\end{aligned}$$

Remark 6.11

In $=^1$, set complement is w.r.t. $Q_{K'}$.

In $=^2$, set complement is w.r.t. $Q_{K'\cap\tilde{K}}$.

In $=^3$, we substitute $\overline{Reach(Sys_{\tilde{K}}) \downarrow K'}$ according to Definition 6.15.

Definition 6.15

We call the set $\overline{Reach(Sys_{\tilde{K}}) \downarrow K'} = Q_{K'\cap\tilde{K}} \setminus Reach(Sys_{\tilde{K}}) \downarrow K'$ the set of states that are **refutable for K' due to $\tilde{K}$**.

$Refutable(K', \tilde{K}) := Q_{K'\cap\tilde{K}} \setminus Reach(Sys_{\tilde{K}}) \downarrow K'$.

Further let the set of substates that are refutable due to $\tilde{K}$ comprise all states that are are refutable for some K' due to $\tilde{K}$.

$Refutable(\tilde{K}) := \bigcup_{K'\subseteq K, |K'|=d} Refutable(K', \tilde{K})$.

Based on the definitions and equations presented above, we are now able to render the idea of Remark 6.10 more precisely and thereby explain, why Algorithm 3 works correctly:

According to the notions above, we want to remove from every set $\mathit{Reach}(\mathit{Sys}_{K'})$ and for all respective $\tilde{K} \subseteq K$ the extensions of $\mathit{Refutable}(K', \tilde{K})$ in K'. The naive approach to do so would be to loop over the sets K'.

However, we receive the same results by looping over all substates $q' \in \mathit{Subs}(K)$ with a maximum length d and checking whether q' is refutable due to some $\tilde{K}$ (i.e., whether $q' \in \mathit{Refutable}(\tilde{K})$). If this is the case then we mark all extensions of q' is unreachable in the respective subsystems where q' occurs.

Complexity Analysis of Algorithm 3 - *reachability Cross-Checking*:

Algorithm 3 prevents looping over all lines in our tables, but instead loops over all possible substates q_{sub} (of size 1 to d-1) and checks for every such substate, whether it occurs at least once in every subsystem to which it fits (Line 5-Line 11). The outer loop [1-21] is performed for any x in $\{1, \ldots, d-1\}$. The second loop [2-20] ranges over $\binom{n}{x} \leq n^x$ subsets.

In the third loop [3-19], we may choose a local state for each of the x components, for which we have an upper bound m^x.

The pairs of inner loops [5-11 and 13-17] look identical:

First, we extend the subsystems to d components, for which there are $\binom{n-x}{d-x}$ $\leq \binom{n}{d-x}$ possibilities.

Second, we choose local states for $d-x$ new components, for which there are $\leq m^{d-x}$ possibilities. The overall complexity of Algorithm 3 is bounded by $O(\sum_{x=1}^{d-1} \binom{n}{x} \cdot \binom{n-x}{d-x} \cdot m^x \cdot m^{d-x}) = O(d \cdot n^d \cdot m^d)$

Remark 6.12

Apart from the factor d (which can be considered a constant), our Cross-

Checking algorithm remains within the asymptotic time bounds already given by the first step of performing the reachability analyses of the subsystems. We consider this to be an ***important*** *property, as any refinement approach that attempts to increase the number d of considered components would instead result in a polynomial time bound with a higher degree.*

Remark 6.13
Note that Algorithm 3 may *be applied iteratively to the result of the previous application thus further reducing the number of states that are marked reachable until we reach a fix point. It is an open question how many iterations will be needed at most.*

Remark 6.14
Please note that the various sets Reach$'(Sys_{K'})$ are still approximations for Reach(Sys) according to Definition 6.13 (p. 131). Thus, all preconditions for Lemma 6.4 still hold when we substitute the sets $A_{K'}$ by Reach$'(Sys_{K'})$. In other words, the increased approximation quality that we obtain be application of Algorithm 3 does not affect the validity of our approach.

6.6 An advanced sufficient Condition

Deadlock-freedom is an important property in itself and in addition establishing safety properties can be reduced to establishing deadlock-freedom [GW92]. In this section, we present a more sophisticated approach of proving a system deadlock-free. In contrast to the first approach presented in Section 6.3 we no longer apply a static predicate but one that takes the parameter d into account: We will see that *small* local deadlocks D of size $|D| \leq d$ can be identified directly, while for *large* deadlocks we have to rely on other indications. In accord with our formalizations so far (and especially to be able to apply Lemma 6.4, p. 132) we do not define a single predicate

P' that refutes the existence of deadlocks but instead split up the property of deadlock-freedom in two properties each of which will be proven by an appropriate predicate P'. We start out with the definitions for (minimal) small and large deadlocks.

Definition 6.16
When we compute the subsystem reachability information as first described in Section 6.3 we choose a value for the parameter d. We call local deadlocks D with $|D| \leq d$ **small** local deadlocks and local deadlocks D with $|D| > d$ **large** local deadlocks.

Definition 6.17
Analogously to our definition of the predicate DL (cf. Definition 2.20, p. 30) we define the **predicates** $DL_{\boldsymbol{Small}}$ and $DL_{\boldsymbol{Large}}$ on states by
$DL_{Small}(q) = \mathit{true}$ iff q contains a small deadlock
$DL_{Large}(q) = \mathit{true}$ iff q contains a minimal large deadlock.

We overload our notions to refer to states as well as interaction systems by the following definitions.

Definition 6.18
$DL_{Small}(Sys) = \mathit{true}$ if $\exists q \in Reach(Sys)\ DL_{Small}(q)$
$DL_{Large}(Sys) = \mathit{true}$ if $\exists q \in Reach(Sys)\ DL_{Large}(q)$

The straightforward observation that a system that contains neither a small nor a large deadlock, does not contain any deadlock is formalized in the following proposition.

Proposition 6.1
$\neg DL_{Small}(Sys) \wedge \neg DL_{Large}(Sys) \Rightarrow$ *Sys is deadlock-free.*

In this section, we present two locally checkable predicates Ref_{Small} and Ref_{Large} that refute the existence of small resp. large deadlocks by the following implications (which are established in the consecutive sections as Lemmas 6.6, p. 145 and 6.9, p. 150):
$\forall K' \subseteq K, |K'| = d \; Ref_{Small}(q \downarrow K') \Rightarrow \neg DL_{Small}(q)$
$\forall K' \subseteq K, |K'| = d \; Ref_{Large}(q \downarrow K') \Rightarrow \neg DL_{Large}(q)$
Clearly, both implications are instances of the implication in Definition 6.12 (p. 131). I.e., Ref_{Small} and Ref_{Large} are local indicator predicates, and we can thus apply Lemma 6.4 (p. 132) to conclude
$\forall K' \subseteq K, |K'| = d \; \forall q' \in Reach(Sys_{K'}) \; Ref_{Small}(q') \Rightarrow \neg DL_{Small}(Sys)$,
respectively
$\forall K' \subseteq K, |K'| = d \; \forall q' \in Reach(Sys_{K'}) \; Ref_{Large}(q') \Rightarrow \neg DL_{Large}(Sys)$.

In the following subsections, we will define these predicates Ref_{Small} and Ref_{Large}, such that they comply with the implications given above and allow us to prove a system deadlock-free. When we traverse the reachable substates in the various sets $Reach(Sys_{K'})$ we will be able to identify small deadlocks directly, whereas the existence of large deadlocks will have to be excluded by a sufficient condition.

6.6.1 *Defining and checking a Condition for small Deadlocks*

In order to prove that a global state q does not contain a small local deadlock, we may simply check that neither of its substates of size 2 to d is a local deadlock. We formalize this in the following definition.

Definition 6.19
We define the **predicate** $\boldsymbol{Ref_{Small}}$ by
$\mathrm{Ref}_{Small}(q') = \neg DL(q')$

The predicate Ref_{Small} now satisfies the following Lemma 6.6 as we already assumed in the previous section.

Lemma 6.6
$\forall K' \subseteq K, |K'| = d \ Ref_{Small}(q \downarrow K') \Rightarrow \neg DL_{Small}(q)$

Proof:
Assume that $DL_{Small}(q)$ holds, i.e., $\exists D \subseteq K$, $|D| \leq d$, such that D is a deadlock in q. Then obviously, D is also a deadlock in any $q \downarrow K'$ with $D \subseteq K'$ and there exists at least one such K' with $|K'| = d$.

Complexity:
When we apply Lemma 6.4, it is again infeasible to check all 2^d subsets of every substate of every subsystem, because this would yield up to $2^d \cdot \binom{n}{d} \cdot m^d$ loop cycles in the first place. Hence, we avoid looping over the arrays $reach(Sys_{K'})$ and checking for every q' with $reach(Sys_{K'})[q'] =$ true and for every projection q'' of q' whether q'' is a local deadlock (which would yield $\binom{n}{d} \cdot m^d \cdot 2^d$ checks). Instead, we loop over the substates in $\{q' \in Subs(K) \mid 2 \leq |K(q')| \leq d\}$ and check whether some extension of q' is marked reachable in some $reach(Sys_{K'})$. If this is the case, we check whether q' is a deadlock. (For a detailed algorithm that checks for small deadlocks see Algorithm 6, p. 202, Appendix.)

Example 6.8
When we apply our check for small deadlocks to the system $DP(6)$ (with $d = 4$) we find out that there are no reachable small deadlocks, even without the application of Cross-Checking.

6.6.2 *Defining a Condition for minimal large Deadlocks*

Obviously, we cannot directly identify a large local deadlock of *Sys* by observing only d components. Instead, we are going to formulate a necessary condition *MinLarge* for the existence of a minimal large deadlock which we will then transform into a sufficient condition for freedom of minimal large deadlocks by negation. In order to define *MinLarge*, we are first going to consider the nature of minimal large deadlocks in order to derive some indicators for their existence.

Definition 6.20
Given a system Sys and a substate $q' \in Subs(K)$, we define the (bipartite) **participation graph** $G_{Part}(q') = (V_1 \uplus V_2, E_1 \uplus E_2)$, where
$V_1 = \{q'_i \mid i \in K(q')\}$,
$V_2 = \{\alpha \in Int \mid \bigcup_{q'_i \in V_1} ea(q'_i) \cap \alpha \neq \emptyset\}$,
$E_1 = \{(q'_i, \alpha) \in V_1 \times V_2 \mid ea(q'_i) \cap \alpha \neq \emptyset\}$, and
$E_2 = \{(\alpha, q'_j) \in V_2 \times V_1 \mid (\alpha \cap A_j) \not\subseteq ea(q'_j)\}$.

The distinction of edges in E_1 and E_2 is irrelevant for the notion of bipartiteness but is made for easier interpretation. E_1 induces the possibilities of interactions, whereas E_2 induces the obviations.

Example 6.9
Let us consider $DP(k)$ with $k \geq 3$ and a global state q, where[8]
$q \downarrow \{Phil_1, Fork_1, Phil_2, Fork_2, Phil_3\} =$
$(wantsleft_{Phil_1}, occupied_{Fork_1}, wantsboth_{Phil_2}, occupied_{Fork_2}, wantsright_{Phil_3})$.
In this case, $D = \{Phil_1, Fork_1, Phil_2, Fork_2, Phil_3\}$ is a minimal local deadlock in q. The bipartite graph $G_{Part}(q')$ for D is given in Figure 6.5, where oval nodes belong to V_1 and rectangular nodes belong to V_2.

[8]In fact, no such q will be reachable in $DP(k)$.

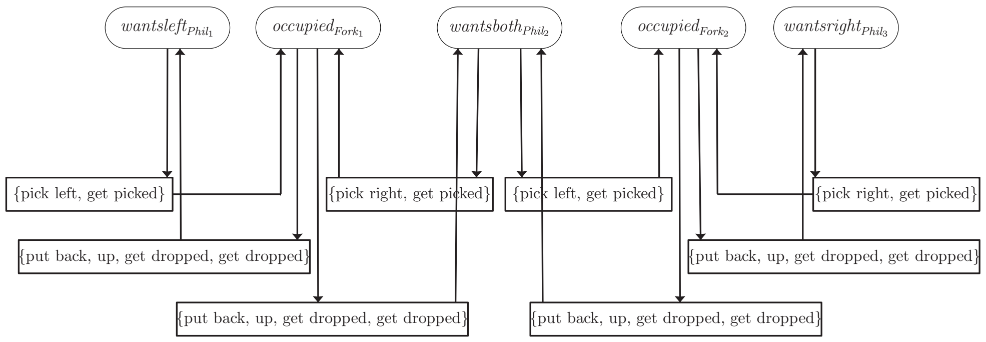


Figure 6.5: The Graph G_{Part} for the Deadlock D given in Example 6.9

Proposition 6.2

Let Sys be an interaction system, and let q be a global state such that a set $D \subseteq K$ is a deadlock in q. Then, every node $v \in V_2$ has at least one ingoing edge from a (predecessor) node $x \in V_1$ and at least one outgoing edge to a (successor) node $y \in V_1$.

Proof:

The existence of x follows from the analogous definitions of V_2 and E_1. The existence of y follows from the fact that D is a deadlock: If there was a node v without such an outgoing edge it would represent an enabled interaction.

Lemma 6.7

If D is a minimal local deadlock in q then $G_{Part}(q \downarrow D)$ is strongly connected, i.e., for every pair $u, v \in V$ there is a path from u to v in $G_{Part}(q \downarrow D)$.

Proof:

Let D be a minimal local deadlock in a state q such that Lemma 6.7 does not hold. Let $u, v \in V$ such that v is not reachable from u. W.l.o.g., we as-

sume $u, v \in V_1$, otherwise we substitute (according to Proposition 6.2) u by one of its corresponding successor nodes y and v by one of its corresponding predecessor nodes x.
Now, let U be the set of nodes that are reachable from u in $G_{Part}(q \downarrow D)$.
On the one hand, $D' := U \cap D$ is a deadlock in q. This is obvious, as for all components in D' all interactions they could participate in (i.e., their direct successor nodes in V_2) require the participation of some other component in D' whose corresponding action is not enabled.
On the other hand, due to $u \in D'$, D' contains at least one component. Due to $v \in D \setminus D'$, $D \setminus D'$ also contains at least one component.
Hence, $D' \subsetneq D$ is a deadlock in q in contradiction to our minimality assumption for D.

We build on Definition 6.9 (p. 120) to refine the idea of *blocking chains* which is used in Algorithm 2 (p. 123). We will derive a dynamic predicate from the static predicate that investigated relations between triples of components. In order to do so we assign to a global state q (resp. a substate q') a directed graph as follows.

Definition 6.21
For a system *Sys* and a global state q we define the **waiting graph** $G_{Wait}(q) = (V, E)$ by:
$V = \{q_i \mid 1 \leq i \leq n\}$ and $E = \{(q_i, q_j) \in V \times V \mid q_i$ waits for $q_j\}$.
For $K' \subseteq K$ and a corresponding substate $q' = q \downarrow K'$ we denote by $G_{Wait}(q')$ the subgraph of $G_{Wait}(q)$ generated by $V' = \{q_i \in V \mid i \in K'\}$.

Example 6.10
The waiting graph $G_{Wait}(q')$ for the local deadlock q' given in Example 6.9 is given in Figure 6.6.

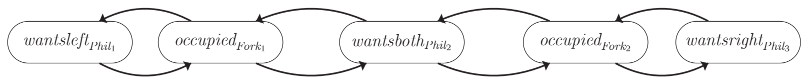


Figure 6.6: $G_{Wait}(q')$ for the deadlock $K(q')$ given in Example 6.9

Remark 6.15

According to Definitions 6.8 and 6.9 one can view $G_{Wait}(q')$ as an abstraction of $G_{Part}(q')$. We obtain $G_{Wait}(q')$ from $G_{Part}(q')$ by introducing an edge $(v_1, v_1') \in V_1 \times V_1$ iff $\exists v_2 \in V_2\ (v_1, v_2) \in E \wedge (v_2, v_1') \in E$. Then we remove the nodes in V_2 and the original edges of $G_{Part}(q')$.

By Remark 6.15 and Lemma 6.7 we may deduce the strong connectedness of $G_{Wait}(q')$.

Corollary 6.3

Let D be a minimal local deadlock in q. Then, $G_{Wait}(q \downarrow D)$ is strongly connected.

Lemma 6.8

*Given a strongly connected graph $G = (V, E)$, there is, for any $k \leq |V|$ a subset $V' \subseteq V$, $|V'| = k$ with an induced subgraph $G \downarrow V' = (V', E')$ such that there is a mapping **order**: $V' \to \{1, \ldots, k\}$ that satisfies for all $u, v \in V'$ the implication*

$order(u) < order(v) \Rightarrow (u, v) \in E'^*$,

where E'^* denotes the transitive closure of E'.

Definition 6.22

In Definition 6.9 (p. 120), we defined the waiting relation between local states of (different) components. Here, we formalize the very same notion.

Let ***wait*** $:= \{(q_i, q_j) \in \bigcup_{i \in K} Q_i \times \bigcup_{j \in K} Q_j \mid q_i$ waits for $q_j\}$.

Let $wait^*$ denote the transitive closure of *wait*.

Corollary 6.3 and Lemma 6.8 imply that for any large deadlock q' there is a subset $K' \subset K(q'), |K'| = d$, for which we will be able to detect (in at least one subsystem) a pattern as described in the following theorem.

Theorem 6.1

Given a reachable global state q and a minimal large deadlock $D \subseteq K$, then there is a subset $K' \subseteq D$ (with $|K'| = d$) such that MinLarge($q \downarrow K'$), where MinLarge(q') := $\exists\, order : K' \rightarrow \{1, \dots, d\}$:

$$order(i) = x \wedge order(j) = x + 1 \Rightarrow (q_i, q_j) \in wait^*$$

As already mentioned at the beginning of this section, we may now use the necessary condition *MinLarge* to define – by negation – a sufficient condition for freedom of minimal large deadlocks.

Definition 6.23
We define the **predicate *Ref_{Large}*** by
$Ref_{Large}(q') = \neg MinLarge(q')$

From the observations presented above, we deduce the following lemma.

Lemma 6.9

Ref_{Large} is a local indicator predicate for freedom of minimal large deadlocks:
$\forall K' \subseteq K, |K'| = d\ Ref_{Large}(q \downarrow K') \Rightarrow \neg DL_{Large}(q)$.

Definition 6.24
We combine the predicates Ref_{Small} and Ref_{Large} to a new predicate P' by
$P'(q') = \text{Ref}_{Small}(q') \wedge \text{Ref}_{Large}(q')$.

Corollary 6.4

By Lemmas 6.6 and 6.9 and Proposition 6.1, we may conclude that P' as defined in Definition 6.24 is a local indicator predicate for deadlock-freedom.

Example 6.11

When we apply P' (for $DP(6)$ with $d = 4$) to the reachable state spaces $Reach(Sys_{K'})$ that we computed in the first place, we will detect 1,584 (of 185.883 reachable) substates for which P' is not valid.

One of these states (that violates Ref_{Large}) is $q' = (vacant_{Fork_1}, think_{Phil_2}, occupied_{Fork_6}, down_{Sem_1})$, where we detect the waiting order $(Fork_6, Fork_1, Phil_2, Sem_1)$. The interpretation of this output is that we consider it possible that there might for example be a fifth (presently unobserved) component j for which Sem_1 waits and which itself waits for $Fork_6$. This cyclic pattern of waiting might be a minimal large deadlock.

Applying P' (for $DP(6)$ with $d = 4$) to the modified reachable state spaces $Reach'(Sys_{K'})$ that we obtain after applying reachability Cross-Checking, the number of substates for which P' is not valid decreases to 432 (of 81,534 reachable states).

These numbers induce that among the substates whose reachability was refuted via Cross-Checking there are indeed critical ones. Even more, the percentage of reachable substates that are critical has decreased. This is due to a tendency in our approach to leave uncritical substates marked reachable.

6.6.3 *Complexity of checking our Condition for large Deadlocks*

In order to check the predicate *MinLarge* on a substate q' we need to

a) Compute the waiting graph for q',

b) Apply the transitive closure to E,

c) Check whether the components $K(q')$ can be ordered as described above.

Under the assumption that we computed a $(n \cdot m \times n \cdot m)$-matrix W with[9] $W(q_i, q_j) = 1$ if q_i waits for q_j and $W(q_i, q_j) = 0$ otherwise as a preprocessing, all of these steps can be computed in $O(d^3)$ as follows:

- For a substate q', we simply create a $d \times d$-matrix $W(q')$ that contains the wait relation for q' and fill it by copying the relevant information from our matrix W. This can be done in $O(d^2)$.
- $W^*(q')$ can be computed (using Warshall's algorithm) in $O(d^3)$.
- We apply Algorithm 4 which also runs in $O(d^3)$.
  Algorithm 4 tries to find a local state q_i (respectively a component i) in $G_{Wait}(q')$ from which all other states can be reached. Then it tries to find a state from which all remaining (not yet ordered) states are reachable and so on. Whenever such a state cannot be found we abort. We return the order when all components are ordered.

Algorithm 4 - Correctness:
It is obvious that if the Procedure Order is not aborted then a returned order suffices our requirements. We show that if there is a linear order as described in Theorem 6.1, then Procedure Order will find one.
Note that if there is a linear order of the components in K' as described in Theorem 6.1 then this also holds for every subset of K' (w.r.t. the graph $G_{Wait}(q')$). This means in every step of Procedure Order, we can choose the next component for the linear order, and it is always guaranteed that the linear order so far can be extended (by a linear order of the remaining components) to a correct linear order for all d components.
Algorithm 4 - Complexity:
The while-loop is performed up to d times. In the while-loop, we determine

[9]The matrix W includes for every substate q' the information about $G_{Wait}(q')$.

Algorithm 4 Procedure Order($G_{Wait}(q')$)

```
1: queue $order$ = new queue();
2: $remain = K(q')$;
3: while $order$.length $\neq$ d do
4:     Find some $i \in remain$ from which all $j \in remain$ are reachable.
5:     if such a component $i$ can be found then
6:         $order$.enqueue(i);
7:         $remain = remain \setminus \{i\}$;
8:     else
9:         abort;
10:    end if
11: end while
12: return order;
```

the next component in our linear order by examining for each of the d components, if the remaining (not yet ordered) components are reachable from it. This check simply accounts to looking up (up to) d entries in the matrix $W^*(q')$.

Thus, the Procedure Order can be performed in an overall time in $O(d^3)$.

6.7 Cross-Checking for Uncriticalness

In the previous section, we defined a locally checkable predicate P' that – when valid on all reachable substates in the various sets $Reach'(Sys_{K'})$ – indicates that Sys is deadlock-free. By applying P', we may consider a state "critical" (i.e., to be a potential deadlock or a part of one) for two reasons: Either we detect a small deadlock in it, or we deem it might be part of a minimal large deadlock[10].

For the example $DP(6)$, this more sophisticated approach only considers $< 1\%$ of the reachable states (with $d = 4$) as potentially deadlock-containing. Also, it harmonizes well with our Cross-Checking approach: After Cross-Checking the relative amount of potentially deadlock-containing states even decreases to 0.5% (cf. Figure B.1, p. 200 Appendix).

In order to decrease the number of critical states even further, we are interested in additional checks that rule out the possibility that a substate q' may be part of a large deadlock. In this section, we present a variant of the Cross-Checking algorithm that checks whether we can guarantee (even in our restricted subsystem view) that an interaction will be enabled in a substate that was hitherto considered to (potentially) be part of a large deadlock.

Definition 6.25
Let Sys be an interaction system and let $Sys_{K'}$ with $K' \subseteq K$ be an induced subsystem. Then we say that $q' \in Q_{K'}$ **enables an original**[11] **interaction** α, denoted by $q' \xrightarrow{\alpha}_{Int}$ if $\exists q'' \in Q_{K'}\ \alpha \in Int\ q' \xrightarrow{\alpha} q''$.

It is a straightforward observation that whenever some q' enables an original interaction, we can mark q' *uncritical* w.r.t. to large deadlocks. (I.e., there is no deadlock D with $K(q') \subseteq D$ in any extension of q'.)

Example 6.12
Applying P' (for $DP(6)$ with $d = 4$) to the state spaces $Reach'(Sys_{K'})$ that we obtain after applying Cross-Checking, the number of substates for which P' (including the check whether a substate allows for the performance of an

[10]It is natural that the latter case contains a higher degree of "uncertainty" and our case studies show that indeed the predicate Ref_{Large} is violated more often than Ref_{Small}.

[11]Please note that the definition denotes some $\alpha \in Int$ (**not** $Int_{K(q')}$). So by original interaction we refer to an interaction that has not been affected by projection to K'.

original interaction) is not valid decreases to 408 (of 81,534 reachable states). One of the 432 remaining states (for which P' is not valid) from Example 6.11 (p. 151) is $q' = (\mathit{wantsleft}_{\mathit{Phil}_1},\ \mathit{vacant}_{\mathit{Fork}_1},\ \mathit{think}_{\mathit{Phil}_2}, \mathit{down}_{\mathit{Sem}_1})$, where the detected waiting order is $(\mathit{Fork}_1,\ \mathit{Phil}_2,\ \mathit{Sem}_1,\ \mathit{Phil}_1)$. However, in this state the original interaction $(\mathit{pickleft}_{\mathit{Phil}_1},\ \mathit{occupy}_{\mathit{Fork}_1})$ may be performed. Thus, q' cannot be a part of a deadlock (esp. not a minimal large one) and does not appear among the critical states that remain when we check this additional condition.

It would now be straightforward to check this additional condition for every substate that is marked reachable in the various subsystems. Instead, considering again the lack of information of subsystems and the way we used Cross-Checking to enhance the quality of the state space approximations we make the following more ubiquitous observation:

Remark 6.16
Let $K' \subseteq K$ and $q' \in Q_{K'}$. Even if q' is not able to perform an original interaction we might still be able to refute that q' is a part of some minimal large deadlock by taking into account the local states of q' and their correlation with the local states of components that are presently not observed (i.e., that are not in K'). Namely, if there is a substate q'' of q' and a subset $K'' \subseteq K$ with $K(q'') \subseteq K''$ such that all reachable extensions of q'' in $\mathit{Sys}_{K''}$ allow for the performance of an original interaction in which some component of K' participates, then there is no reachable global state q such that K' is part of a deadlock in q.

Example 6.13
As stated in Example 6.11 (p. 151), when we apply P' (for $DP(6)$ with $d = 4$) to the reachable state spaces $\mathit{Reach}'(\mathit{Sys}_{K'})$ that we obtain after applying Cross-Checking, the number of substates for which P' is not valid is 432. One of these states is $q' = (\mathit{vacant}_{\mathit{Fork}_1}, \mathit{think}_{\mathit{Phil}_2}, \mathit{occupied}_{\mathit{Fork}_6}, \mathit{down}_{\mathit{Sem}_1})$,

where our *Procedure Order* detected the waiting order ($Fork_6$, $Fork_1$, $Phil_2$, Sem_1). Now let us consider the substate $q'' = (vacant_{Fork_1}, think_{Phil_2}, down_{Sem_1})$ of q' and the subsystem $Sys_{K''}$ with $K'' = \{Phil_1, Fork_1, Phil_2, Sem_1\}$ in which q'' occurs (namely the subsystem from Example 6.12).
The only extensions of the substate q'' that are in $Reach'(Sys_{K''})$ are the states $\hat{q}''_1 = (wantsboth_{Phil_1}, vacant_{Fork_1}, think_{Phil_2}, down_{Sem_1})$ and $\hat{q}''_2 = (wantsleft_{Phil_1}, vacant_{Fork_1}, think_{Phil_2}, down_{Sem_1})$. Note that **both** of them enable the original interaction $\alpha = \{pickleft_1, occupy_1\} \in Int$, in which $Fork_1$ participates.
So for every globally reachable state q with $q \downarrow \{Fork1_, Phil_2, Sem_1\}$ some component in $K' = \{Fork_1, Phil_2, Sem_1\}$ is able to participate in an enabled interaction. Thus, the extension $q' = (vacant_{Fork_1}, think_{Phil_2}, occupied_{Fork_6}, down_{Sem_1})$ of q'' cannot be a part of a deadlock.

Our aim is to incorporate the idea of Remark 6.16 into our locally checkable predicate Ref_{Large} in such a way that it is still an indicator predicate (cf. Lemma 6.9, p. 150). To do so we first give an auxiliary definition.

Definition 6.26
For a state q' in a subsystem $Sys_{K'}$ and a subset $K'' \subseteq K'$ we say K'' **may participate in some original interaction** (denoted by the boolean value part_pos($Sys_{K'}[q'], K''$)) iff

- $q' \notin Reach(Sys_{K'})$ (i.e., q' is not reachable in $Sys_{K'}$) **or**
- $q' \xrightarrow{\alpha}_{Int} \wedge \bigcup_{i \in K''} i(\alpha) \neq \emptyset$

Now, we can formalize a locally checkable *Uncritical* predicate for freedom of minimal large deadlocks of a state q as follows.

Definition 6.27
$Uncritical(q') :=$
$\exists K'' \subseteq K(q')\ \exists K' \supseteq K''\ |K'| = d\ \forall q'' \in Ext(q', K')\ part\_pos(Sys_{K'}[q''], K'')$

Definition 6.28
We redefine the **predicate $\boldsymbol{Ref_{Large}}$** that was first defined in Definition 6.23 (p. 150) by
$Ref_{Large}(q') = \neg MinLarge(q') \vee Uncritical(q')$

It is obvious that this redefinition maintains the validity of Lemma 6.9 (p. 150), so we can refute the criticalness of a state by checking the predicate *Uncritical*. In order to check the predicate *Uncritical* for all states in our sets $Reach'(Sys'_K)$, we apply again the general approach that was used in Algorithm 3, i.e., we loop over all substates q' of $size \leq d$ and check for all subsystems $Sys_{K'}$ with $q' \in Subs(K')$ whether there is at least one K' such that all extensions in $Reach'(Sys'_K)$ enable some interaction in which a component in $K(q')$ participates. If this is the case, we mark all extensions of q' of length d *uncritical* in all subsystems $Sys_{K'}$ with $q' \in Subs(K')$.

Example 6.14
When we apply P' (for $DP(6)$ with $d = 4$) to the reachable state spaces $Reach'(Sys_{K'})$ that we obtained after applying Cross-Checking, the number of substates for which P' is not valid was 432 (of 81,534 substates in the various $Reach'(Sys_{K'})$).
If we apply P' to only those states that remain unmarked by our uncriticalness Cross-Checking, the number of critical substates we find is reduced to 24 (of 81,534) (= 0.03%, cf. Figure B.1, p. 200 Appendix). When we resolve these 24 crticial substates up to symmetry the number of critical substates decreases to 2.

Remark 6.17
We have not yet mentioned in which way we want to incorporate Algorithm 5 into our overall approach. However, the redefinition of our predicate Ref_{Large} to exclude the existence of large deadlocks (cf. Definition 6.28, p. 157) where

Algorithm 5 Uncriticalness Cross-Checking(Sys, d, $reach$)

```
1:  for $x := 1$ to $(d-1)$ do
2:      for all subsets $K'' = \{i_1, \ldots, i_x\}$ of $K$ do
3:          for all $q'' = (q_{i_1}, \ldots, q_{i_x}) \in Q_{K''}$ do
4:              uncritical := false;
5:              for all subsystems $Sys_{K'}$ with $K'' \subseteq K'$ (and $|K'| = d$) do
6:                  refute := true;
7:                  for all $q' \in Ext(q'', K')$ do
8:                      refute := refute AND part_pos($Sys_{K'}[q'], K''$);
9:                  end for
10:                 uncritical := uncritical OR refute;
11:             end for
12:             if (uncritical) then
13:                 for all subsystems $Sys_{K'}$ with $K'' \subseteq K'$ (and $|K'| = d$) do
14:                     for all $q' \in Ext(q'', K')$ do
15:                         critical($Sys_{K'}$)[$q'$] := false;
16:                     end for
17:                 end for
18:             end if
19:         end for
20:     end for
21: end for
```

we simply compose MinLarge and Uncritical with the $\vee$*-operator accounts for the fact that the order in which we apply the algorithms is not important. In order to clear things up, let us assume that we apply our check for uncriticalness directly after the check for small deadlocks. Also, it is unnecessary to introduce a flag critical whose existence is assumed in Algorithm 5, but* *instead simply set the reachability flag of a substate q' that we want to mark uncritical to false in our reachability arrays. We will refer to the thus computed arrays, respectively sets by $Reach''(Sys_{K'})$. By doing so, we prevent uncritical states from being processed in the concluding check for large deadlocks.*

6.8 Restriction to connected Subsystems

Note that for the purpose of deadlock detection we may restrict our attention to systems that are connected in a topological sense (as for an unconnected system it is convenient to prove its connected parts deadlock-free). However, if the original interaction system is connected, we may restrict all computations so far to connected subsystems. In order to formulate this proposition more exactly, we are going to introduce some notions first.

Definition 6.29
For an interaction system *Sys* we define the **interaction graph** $G_{Int} = (V, E)$ by $V := K$ and $E := \{\{i, j\} \mid \exists \alpha \in Int\, (\alpha \cap A_i \neq \emptyset \wedge \alpha \cap A_j \neq \emptyset)\}$
We call an interaction system **connected** if its interaction graph is connected.

Example 6.15
Figure 6.7 sketches the interaction graph for a large instance of Tanenbaum's dining philosophers, where the squares in the outer ring represent philosophers alternating with forks and the squares in the inner ring represent the semaphores.

The approach we presented so far can be performed in five phases:

1. Perform reachability analyses for all subsystems
2. Perform Algorithm 3 (p. 139) reachability Cross-Checking

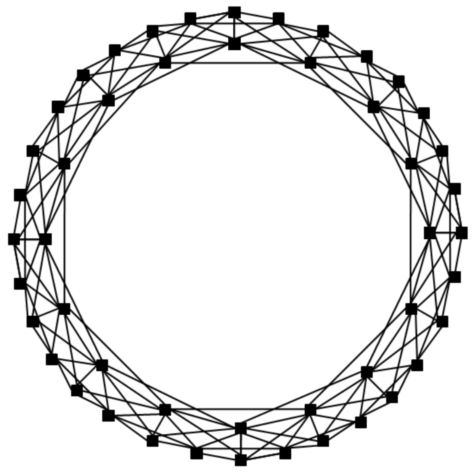


Figure 6.7: The interaction graph for Tanenbaum's Dining Philosophers

3. Check Ref_{Small} on all substates $q' \in \bigcup_{K' \subseteq K, |K'|=d} Reach'(Sys_{K'})$
4. Perform Algorithm 5 (p. 158) uncriticalness Cross-Checking
5. Check Ref_{Large} on all Substates $q' \in \bigcup_{K' \subseteq K, |K'|=d} Reach''(Sys_{K'})$

We modify these five phases now by restricting all considerations to connected subsystems, where our restricted phases compute the following data:

1. Perform reachability analyses for all **connected** subsystems
2. Perform Algorithm 3 (p. 139) reachability Cross-Checking where we apply the following modifications:

   Line 5: for all **connected** subsystems $Sys_{K'} \ldots$

   Line 13: for all **connected** subsystems $Sys_{K'} \ldots$
3. Check Ref_{Small} on all substates $q' \in \bigcup_{\textbf{connected } K' \subseteq K, |K'|=d} Reach'(Sys_{K'})$
4. Perform Algorithm 5 (p. 158) uncriticalness Cross-Checking where we apply the following modifications:

   Line 5: for all **connected** subsystems $Sys_{K'} \ldots$

   Line 13: for all **connected** subsystems $Sys_{K'} \ldots$
5. Check Ref_{Large} on all substates $q' \in \bigcup_{\textbf{connected } K' \subseteq K, |K'|=d} Reach''(Sys_{K'})$

Given a system *Sys* our *general approach* (i.e., without restriction to connected subsystems, *GA* in short) is able to verify it deadlock-free iff neither of the steps 3 and 5 detects a violation of our local indicator predicates. We write *GA(Sys) = true* if this is the case and *GA(Sys) = false* otherwise. Analogously, we define the same notion for the *restricted approach* (*RA* in short) and refer to it by *RA(Sys)*. We want to show that the restriction to connected subsystems as described above causes neither a loss of correctness nor of information. We formulate this proposition in Theorem 6.2 and prove it via Lemmas 6.10 and 6.11.

Theorem 6.2

The restriction to connected subsystems as described above causes neither a loss of correctness nor of information.

Proof:

The proof of Theorem 6.2 is given by Lemmas 6.10 and 6.11.

Lemma 6.10

$RA(Sys) \Rightarrow$ *Sys is deadlock-free*

Proof:

The restriction to connected subsystems in steps 1, 3 and 5 compared to the one in steps 2 and 4 is of a different quality: Considering less subsystems in steps 1,3 and 5 means (at least syntactically) weakening the condition for deadlock-freedom while considering less subsystems in steps 2 and 4 means (at least syntactically) invigorating the condition because we abstain from decreasing the number of substates on which we check our local indicator predicates.

So basically to prove that correctness is preserved, we have to prove that the predicates $\mathit{Ref}_{\mathit{Small}}$ and $\mathit{Ref}_{\mathit{Large}}$ imply deadlock-freedom of a system, even if we check them on restricted subsystems only. For this purpose, we apply the following Propositions 6.3 and 6.4 to turn the implications given in Lemmas 6.6 (p. 145) and 6.9 (p. 150) into our desired stronger results.

Proposition 6.3
$\forall K' \subseteq K, |K'| = d, K'$ connected $\mathit{Ref}_{\mathit{Small}}(q \downarrow K') \Rightarrow$
$\forall K' \subseteq K, |K'| = d\ \mathit{Ref}_{\mathit{Small}}(q \downarrow K')$

Proof:
Assume that there exists a global state q and an unconnected $K' \subseteq K$ with $\neg \mathit{Ref}_{\mathit{Small}}(q \downarrow K')$. Then, there must be a connected subset $\tilde{K} \subseteq K'$ that includes a small deadlock[12]. Successively extending $\tilde{K}$ to size d (such components must exist because *Sys* is connected) yields a connected set $\hat{K}$ with $\neg \mathit{Ref}_{\mathit{Small}}(q \downarrow \hat{K})$.

Proposition 6.4
$\forall K' \subseteq K, |K'| = d, K'$ connected $\mathit{Ref}_{\mathit{Large}}(q \downarrow K') \Rightarrow$
$\forall K' \subseteq K, |K'| = d\ \mathit{Ref}_{\mathit{Large}}(q \downarrow K')$

Proof:
We defined $\mathit{Ref}_{\mathit{Large}}$ as the negation of the predicate *MinLarge*. So for any K' for which $\mathit{Ref}_{\mathit{Large}}(q \downarrow K')$ does not hold $\mathit{MinLarge}(q \downarrow K')$ does hold. However, $\mathit{MinLarge}(q \downarrow K')$ implies the existence of a sequence of waiting relations that form an order. This implies that any K' with $\neg \mathit{Ref}_{\mathit{Large}}(q \downarrow K')$ has to be connected.

[12]There is a minimal deadlock $D \subseteq K'$ and a minimal deadlock cannot contain components that do not (not even over third party components) communicate.

Lemma 6.11

$GA(Sys) \Rightarrow RA(Sys)$

Proof:

We want to prove that the restriction to connected subsystems does not affect the chances of our approach to prove a system deadlock-free. It is obvious that checking *less* substates for small, respectively large deadlocks (by checking the respective predicates) cannot lead to a situation where the restriction to connected subsystems affects our chances for success in a negative way. There are however two aspects of the restriction to connected subsystems that could – at first glance – very well have a negative impact on our results:

A) Firstly, when marking substates unreachable in the reachability Cross-Checking algorithm, we take *less* substates into account to refute the reachability of others.

B) Secondly, when marking substates uncritical in the uncriticalness Cross-Checking algorithm, we take *less* substates into account to mark others uncritical.

We will show that in neither of the two cases, the restriction to connected subsystems affects the number of states whose reachability resp. criticalness is refuted. This is formalized in the following Lemmas 6.12 (concerning A) and 6.13 (concerning B).

Definition 6.30

In the Algorithms 3 and 5, the reachability, respectively the criticalness of states is refuted in Lines 13 to Lines 17.

This happens iff a state q'' (defined in Line 3) has been processed in a subsystem $Sys_{K'}$ (defined in Line 5) in such a way that all extensions of q' in K' were unreachable, respectively allowed for a participation in an original interaction. In this case, we call q'' a **witness** in K'.

Definition 6.31

Let $Sys = (K, \{A_i\}_{i \in K}, \mathit{Int}, \{T_i\}_{i \in K})$ be a connected interaction system and $K' \subseteq K$. K' induces a subgraph $G_{\mathit{Int}}(K')$ of the interaction graph G_{Int} of Sys where $G_{\mathit{Int}}(K')$ is not necessarily connected. We define the **connected decomposition** of K' by $CD(K') = \{K'_1, \ldots, K'_k\}$, with

- $\biguplus_{1 \leq l \leq k} K'_l = K'$
- i and j belong to the same K'_l iff i is reachable from j in $G_{\mathit{Int}}(K')$.

Proposition 6.5

Let $Sys = (K, \{A_i\}_{i \in K}, \mathit{Int}, \{T_i\}_{i \in K})$ be a connected interaction system and $K' \subseteq K$ such that $Sys_{K'}$ is unconnected. Then (due to the fact that the connected parts of K' act totally independently from each other) the reachable state space of $Sys_{K'}$ equals the Cartesian product of the reachable state spaces of the subsystems induced by the connected parts of K':
$\mathit{Reach}(Sys_{K'}) = \prod_{K'' \in CD(K')} \mathit{Reach}(Sys_{K''})$.

Definition 6.32

Let $Sys = (K, \{A_i\}_{i \in K}, \mathit{Int}, \{T_i\}_{i \in K})$ be a connected interaction system and $K' \subseteq K$ such that $Sys_{K'}$ is unconnected with $CD(K') = \{K'_1, \ldots, K'_k\}$. Further, let $q' \in \mathit{Subs}(K')$. Then we denote by **covering** of q' in K' the set $\mathit{Cov}(q', K') = \bigcup_{K'_i \in CD(K') \text{ with } K'_i \cap K(q') \neq \emptyset} K'_i$.

Lemma 6.12

Let $K' \subseteq K$ and $q' \in \mathit{Subs}(K')$. If q' is a witness in K', i.e., the extensions of q' are unreachable in K' and $Sys_{K'}$ is unconnected, then there exists a connected subsystem K'' in which q' (or a substate q'' of q') is also a witness. More formally put:

$$\begin{aligned}
&\exists K' \subseteq K, |K'| = d, K' \text{ unconnected } \exists q' \in \mathit{Subs}(K') : \\
&\mathit{Ext}(q', K') \cap \mathit{Reach}(\mathit{Sys}_{K'}) = \emptyset \\
\Rightarrow \quad &\exists K'' \subseteq K, |K''| = d, K'' \text{ connected } \exists q'' \in \mathit{Subs}(K(q')) : \\
&\mathit{Ext}(q'', K'') \cap \mathit{Reach}(\mathit{Sys}_{K''}) = \emptyset
\end{aligned}$$

Proof:

Firstly, we prove that the reason why no extension of q' is reachable in $\mathit{Sys}_{K'}$ lies in those components of K' that are connected with $K(q')$.

More formally: $\forall \hat{q}' \in \mathit{Ext}(q', \mathit{Cov}(q', K'))\ \hat{q}' \notin \mathit{Reach}(\mathit{Sys}_{\mathit{Cov}(q',K')})$.

We know that $\mathit{Ext}(q', K') \cap \mathit{Reach}(\mathit{Sys}_{K'}) = \emptyset$. On the other hand, we can apply the extension operator succesively, i.e., $\mathit{Ext}(q', K') = \mathit{Ext}(\mathit{Ext}(q' \downarrow \mathit{Cov}(q', K')), K' \backslash \mathit{Cov}(q', K'))$ and by Proposition 6.5 we know $\mathit{Reach}(\mathit{Sys}_{K'})$ $= \mathit{Reach}(\mathit{Sys}_{\mathit{Cov}(q',K')}) \times \mathit{Reach}(\mathit{Sys}_{K' \backslash \mathit{Cov}(q',K')})$.

Substituting these expressions yields:

$$\begin{aligned}
&\mathit{Ext}(\mathit{Ext}(q' \downarrow \mathit{Cov}(q', K')), K' \setminus \mathit{Cov}(q', K')) \\
\cap \quad &\mathit{Reach}(\mathit{Sys}_{\mathit{Cov}(q',K')}) \times \mathit{Reach}(\mathit{Sys}_{K' \backslash \mathit{Cov}(q',K')}) \\
= \quad &\emptyset
\end{aligned}$$

As the global starting state q^0 of Sys is reachable in the global system, we know that its projection is reachable in every subsystem, in particular $q^0 \downarrow (K' \setminus \mathit{Cov}(q', K')) \in \mathit{Reach}(\mathit{Sys}_{K' \backslash \mathit{Cov}(q',K')})$.

However, assuming the existence of a $\hat{q}' \in \mathit{Ext}(q', \mathit{Cov}(q', K'))$ that is reachable in $\mathit{Sys}_{\mathit{Cov}(q',K')}$ yields that both sets of which we build the intersection here would contain the composition of $\hat{q}'$ with $q^0 \downarrow (K' \setminus \mathit{Cov}(q', K'))$. This yields a contradiction to their intersection being empty.

So we conclude $\forall \hat{q}' \in \mathit{Ext}(q', \mathit{Cov}(q', K'))\ \hat{q}' \notin \mathit{Reach}(\mathit{Sys}_{\mathit{Cov}(q',K')})$.

If $\mathit{Cov}(q', K')$ is connected then we may simply extend this set of components to a connected set K'' of size d and with $q'' = q'$, we surely have $\mathit{Ext}(q'', K'') \cap \mathit{Reach}(\mathit{Sys}_{K''}) = \emptyset$. If however $\mathit{Cov}(q', K')$ is unconnected

then let $CD(Cov(q', K')) = \{K'_1, \dots, K'_k\}$.
Assume that $\forall 1 \le i \le k\ \exists \hat{q}_i{}' \in Ext(q' \downarrow K'_i, K'_i)$ with $\hat{q}_i{}' \in Reach(Sys_{K'_i})$.
If this was the case we could compose these (disjoint) extensions to a state $\hat{q}_i$ and invoke Proposition 6.5 to conclude that $\hat{q}_i \in Reach(Sys_{Cov(q',K')})$ which is a contradiction because this $\hat{q}_i$ would be an extension of q' in $Cov(q', K')$.
Thus, we have proven that $\exists i \in \{1, \dots, k\}\ \forall \hat{q}' \in Ext(q \downarrow K'_i, K'_i)\ \hat{q}' \notin Reach(Sys_{K'_i})$. Finally, we define $q'' = q' \downarrow K'_i$ and extend K'_i to a connected set K'' of size d and we surely have $Ext(q'', K'') \cap Reach(Sys_{K''}) = \emptyset$.

Lemma 6.13

Let $K' \subseteq K$ and $q' \in Subs(K')$. If q' is a witness in K', i.e., the extensions of q' can all participate in original interactions in K', and $Sys_{K'}$ is unconnected, then there exists a connected subsystem K'' in which q' (or a substate q'' of q') is also a witness.
More formally put:

$$\begin{aligned}
& \exists K' \subseteq K, |K'| = d, K' \text{ unconnected } \exists q' \in Subs(K') : \\
& \forall \hat{q}' \in Ext(q', K')\ part\_pos(Sys_{K'}[\hat{q}'], K(q')) \\
\Rightarrow\ & \exists K'' \subseteq K, |K''| = d, K'' \text{ connected } \exists q'' \in Subs(K(q')) : \\
& \forall \hat{q}'' \in Ext(q'', K'')\ part\_pos(Sys_{K''}[\hat{q}''], K(q'))
\end{aligned}$$

Proof:

Firstly, we prove that the reason why an original interaction in which q' participates is always possible in K' lies in those components of K' that are connected with q'.
More formally put:
$\forall \hat{q}' \in Ext(q', Cov(q', K'))\ part\_pos(Sys_{Cov(q',K')}[\hat{q}'], K(q'))$.
Assume the existence of a state $\hat{q}' \in Ext(q', Cov(q', K'))$ for which
$\neg part\_pos(Sys_{Cov(q',K')}[\hat{q}'], K(q'))$.
Then $\hat{q}' \in Reach(Sys_{Cov(q',K')})$ must hold, but $\hat{q}'$ does not allow for an original

interaction. As a consequence, the composition of $\hat{q}$ with $q^0 \downarrow K' \backslash Cov(q', K')$ would be reachable in $Sys_{K'}$ and would obviously not allow for an original interaction in which q' participates. Such a state would however contradict our assumptions.

If $Cov(q', K')$ is connected then we may simply extend this set of components to a connected set K'' of size d and with $q'' = q'$, we may conclude (as additionally observed components cannot interfere with the executability of original interactions) $\forall \hat{q}' \in Ext(q'', K'')\ part\_pos(Sys_{K''}[\hat{q}'], K(q'))$. If however, $Cov(q', K')$ is unconnected then let $CD(Cov(q', K')) = \{K'_1, \ldots, K'_k\}$. Assume that for each $1 \leq i \leq k\ \exists \hat{q}' \in Ext(q', K'_i)$ s.t. $\hat{q}' \in Reach(Sys_{K'_i}) \wedge \hat{q}' \not\rightarrow_{Int}$.

Then their composition would also be reachable in $Sys_{Cov(q',K')}$ and it would not allow for an original interaction in which q' participates in contradiction to $\forall \hat{q}' \in Ext(q', Cov(q', K'))\ part\_pos(Sys_{Cov(q',K')}[\hat{q}'], K(q'))$.

Hence, $\exists i \in \{1, \ldots, k\}\ \forall\ \hat{q}' \in Ext(q', K'_i)$:

$\hat{q}' \in Reach(Sys_{K'_i}) \Rightarrow \hat{q}' \xrightarrow{\alpha}_{Int}$, respectively

$\forall\ \hat{q}' \in Ext(q', K'_i)\ part\_pos(Sys_{K'_i}[\hat{q}'], K(q'))$.

Finally, we define $q'' = q' \downarrow K'_i$ and extend K'_i to a connected set K'' of size d and (as additionally observed components do not interfere with the executability of original interactions) may conclude:

$\forall\ \hat{q}'' \in Ext(q'', K'')\ part\_pos(Sys_{K''}[\hat{q}''], K(q''))$.

Remark 6.18

The restriction to connected subsystems will result in a major speed-up for our approach for systems with "regional connectivity" or in other words, for systems where the interaction graph $G_{Int} = (V, E)$ is sparse, i.e., $|E| \ll |V|^2$.

Example 6.16

For Tanenbaum's dining philosophers as modeled here, the maximum degree of a node in the interaction graph is 9 (independently of p, cf. Figure 6.7, p. 160). So an arbitrary $K' \subseteq K$ such that $Sys_{K'}$ is connected can be selected as follows: Firstly, select an arbitrary component (one out of n). Secondly, select a neighbor (up to 9 choices) of some already selected component (up to $d-1$ choices). Repeat the second step $d-1$ times. This consideration allows us to derive that the maximum number of *connected* subsystems is bounded by $n \cdot (9(d-1))^{d-1} = O(n)$ for a fixed choice of d.

Chapter 7

Conclusion & Related Work

7.1 Formal Verification

Since the first emergence of attempts to not only check systems for errors, but to rather formally prove desired properties of a system, one may identify five central approaches that try to achieve this goal in different ways:

i) Deductive Program Verification

ii) Abstract Interpretation

iii) Sufficient Conditions

iv) Model-Checking

v) Equivalences

i) Deductive program verification goes back to [Flo67] and [Hoa69] and combines program code and logic to formally verify programs. These first elementary approaches have been extended by higher programming constructs like arrays or procedure calls [GL80], parallelism [GL81, Owi75] and abstract data types [Owi79]. Prominent examples for tools for deductive verification are, e.g., StEp [MBB$^+$95] and TLV [SP96]. Apart from these, also classic theorem provers like Isabelle [NPW02], PVS [OS08], ACL2 [KMM00] and

Coq [PM93] can be applied for the purpose of verification. Furthermore, deductive methods allow to deduce (by a logic) properties of a system based on a specification of a system. As the application of deduction can be a lengthy process, one will in many cases not verify the final program code but rather the underlying algorithm or a simplified abstraction of the program.

A focus on component-based systems in the sense of this work is rarely found in the area of deductive program verification. An approach that deals in this context with the model of interaction systems is D-Finder [BBNS09], whose theoretical background was presented in [BBSN08]. D-Finder exploits compositionality of interaction systems and builds on so-called interaction-invariants that characterize the coordination between components to iteratively prove more and more strict invariance-properties. After each iteration, it checks whether a desired property (which is also formulated as an invariant) can be derived. In [BBNS09], deadlock is treated as a possible property. The depth of the iteration is a parameter and bears analogy to the parameter in our approach. In both cases raising the value of the parameter allows for a more precise analysis. Furthermore, the approach of D-Finder suggests to interpret the check that is applied in every iteration as a sufficient condition.

ii) Abstract interpretation comprises all approaches that abstract from certain aspects of a system (respectively program) to ease the proof of properties. A classic example is the abstraction from data values by intervals. If, given a reactive system S, the (Kripke-)structure $K(S)$, on which the system S is usually mapped, is too large an abstraction is created. This is not done by first mapping to $K(S)$ and then abstracting from it, but by applying a different (more abstract) semantics to the system directly. The aim of these approaches is a reduction of the state space by identifying sets of states based on the program. The formalization of this idea goes back to [CC77]. The concrete (original) respectively the abstract (smaller) state space are both described by a partial order $(C, \sqsubseteq)$ respectively $(A, \preceq)$. Between these partial orders a relation is established by an abstraction function $\alpha : C \to A$ and a concretization function $\gamma : A \to C$. Usually, one demands that this pair of mappings builds a Galois-connection [CC77, CC92, San77] from $(C, \sqsubseteq)$ to $(A, \preceq)$. Weaker conditions than a Galois-connection have also been defined, where the abstraction- respectively concretization function are replaced by appropriate relations [BBLS93, CGL92, CGL94, LGS$^+$95].

An important requirement concerning these relations is that they have to make sure, that the validity of a proof of a property in the abstract system is maintained when going to the concrete system. Once a system property is proved on the abstract system, it thus also holds for the concrete one. If the property can not be proven correct in the abstract system, one can try to achieve this aim by refinement. This framework yields a trade-off, as an abstraction function has to be fine enough to allow for the proof of the desired property on the one hand but coarse enough to allow for checking in the time available.

Abstract interpretation has been applied (mainly in the context of embedded systems) with great success. Finding an abstraction that is appropriate for proving a property is a non-trivial step and requires a good understanding of the "domain of application". Like deductive program verification, this area hitherto lacks a special focus on component-based systems.

iii) Sufficient conditions comprise a variety of approaches. We subsume under this term different techniques that try to establish properties of systems under certain preconditions. These preconditions may occur in form of structural restrictions like, e.g., the class of component-based systems with tree-like communication structure which is among others given for (nets of) master-client systems. The question of deadlock-freedom for systems with a tree-like communication structure has been investigated for different models of computation [BR91, MM08a, MM09, Lam09, AB03]. Brookes and Roscoe showed [BR91] that it is sufficient for unidirectional tree-like systems to prove deadlock-freedom for subsystems consisting of two neighbor components. If this is done for all pairs of neighbor components one may conclude deadlock-freedom for the complete system. Sufficient conditions have in common that they concentrate on subclasses of systems and that they usually allow for larger input parameters (i.e., mainly larger systems) at the expense of the variety of treatable systems. They have been developed independently across all models of computation. Some sufficient conditions (like the Cross-Checking approach presented in this work) that treat deadlock-freedom are, e.g., [BCD02] which uses weak bisimulation on an abstract description language whose semantics is related to the process algebras CSP and CCS, or [IU01] which is based directly on CCS and exploits a partial equivalence relation. Further examples for sufficient conditions are [BR91, AG97, AC05]. Our Cross-Checking approach belongs to the class of sufficient conditions.

iv) Since its appearance in the 80s, when Emerson and Clarke [EC82] and independently of them Queille and Sifakis [QS82] presented a new approach for the verification of computer systems, model-checking makes a standalone area of computer science. Specifically adapted model-checking is also applied to component-based systems [Arb04]. Model-checking checks for a system, which is usually given by an automaton M, a property represented by a temporal-logic formula, e.g., an LTL-formula ϕ. For this purpose, the nega-

tion of ϕ is translated into an automaton B and the automaton $M \cap B$ for the intersection is established. If in this automaton no final state is reachable from the starting state, then the system satisfies ϕ. The automaton construction also suffers from state space explosion, i.e., if the system consists of k parallel processes (or components) the state space of M may be exponentially large in k.
Model-checking can be divided into four subcategories, according to the nature of the different approaches that try to avoid state space explosion: *Symbolic model-checking*, *Bounded model-checking*, *Half-order reduction* and *Abstraction*.

v) In Chapter 2 we defined various equivalences that describe similarities between concurrent systems (respectively their corresponding behaviors). A straightforward idea is to compute a simulation for a given pair of systems. Examples of how such approaches can be implemented are the algorithm of Kanellakis and Smolka [KS90] for which an improved parallel implementation was published in [JKKO98], or the bisimulation quotient computation by Paige and Tarjan [PT87]. A discussion of approaches and results of equivalence computation was presented by Cleaveland and Sokolsky [RO01]. As we already pointed out in the introduction of this theses, equivalencs are the most original and general way to formalize the similarity of systems or of a system implementation and its specification. As a consequence, one can say that equivalence computation is the most straightforward way to reason about verficiation, which means that this approach is most vulnerable to state space explosion which manifests itself – for interaction systems – in the PSPACE-hardness results that we presented in Section 6.

7.2 Related Work

Depending on the model, the system, the desired property, and other aspects one can name advantages and disadvantages for each of the various approaches, and there exist various benchmarks that favor one or the other technique like Moshi Vardi pointed out in his talk on the *Reachability Problems Conference* [Var09] in 2009. This fact makes a general quantitative comparison impossible and the last conclusion of wisdom seems to be a distant prospect and motivates new approaches to prove properties for concurrent systems, especially in a component-based scenario.

Comparing the approach presented in this work to other ideas, it can be said that in symbolic model-checking there exist a couple of approaches which – similarly to Cross-Checking – try to avoid state space explosion by decomposing a system to subsystems.

Cho et al. [CHM$^+$] gave different algorithms for the (forwards-)reachability analysis of the state space that yields an over-approximation. The basic idea they use is to decompose the state variables in pairwise disjoint subsets and then perform reachability analyses on these subsets. The corresponding subsets of states can then be viewed as “subautomata”. The original problem is thus reduced to the ordinary reachability analysis on smaller automata. Cho et al. treated different variants of the approximation of the transition relation for (forwards-)reachability: MBM (machine by machine) and FBF (frame by frame). The main difference between these approaches is the way in which they model the communication between the respective subautomata. FBF allows communication between the subautomata in every time frame of an lfp-routine (i.e., reachability analysis). MBM on the other hand allows communication only after the reachability analyses have been completed. Furthermore, two variants of the FBF-approach have been proposed, RFBF (reached frame by frame) and TFBF (to frame by frame), which differ with respect to the constraint sets that are imposed on the subautomata during the reachability analyses.

In [CHM$^+$96] Cho et al. presented heuristics to compute favorable partitions for the set of state variables. One possibility is to exploit potentially accessible knowledge about subsystems from which the original system was composed. These subsystems could be viewed as components in our setting.

Moon et al. [MJH$^+$98] applied algorithms for the approximative computation of the reachable state space to support model-checking. Cabodi et al. [CCQ94] combined approximative forwards-reachability with exact backwards-reachability. Lee et al. [LPJ$^+$96] proposed so-called "tearing"-schemata for approximative backwards-analysis and extended it to the idea of "variable tearing" and "blockwise tearing" to approximate the successor function of a system and then refine it in a stepwise manner until a given ACTL or ECTL [McM92] formula can be proved or refuted. They also partitioned the set of state variables into pairwise disjoint subsets, formed blockwise subrelations for the various subsets, and finally connected them until the resulting successor function was exact enough to prove or disprove a property.

The basic idea of the investigation of subsystems in symbolic model-checking is that their representation as BDDs is more compact. From these BDDs an approximation of the original systems is then obtained by conjunction and this over-approximation is then used to check conditions that imply properties of the original system. For this purpose, an over-approximation of the global state space is computed by iteratively abstracting, concretizing and applying the global successor function. The computation of the global successor function is an important technical aspect and an efficient approach for this step is presented, e.g., in [GDHH98].

In contrast to the approaches [RS95] and [RMSS98], we are (like [Gov00]) interested in the computation of an over-approximation. In contrast to the approaches in [CCQ94, CHM$^+$, CHM$^+$96, LPJ$^+$96, MJH$^+$98], we allow (like [Gov00]) overlapping projections and even push the idea further by investigating all (relevant) overlapping projections of a fixed size d.

To the best of our knowledge, the work presented in [GDHH98, GD98, GDB99, Gov00] is the one closest related to our Cross-Checking approach. In both cases non-disjoint subsystems (i.e., overlapping projections) are established, whose state space is bounded by a polynomial of degree d, where d is the maximum (and as far as Cross-Checking is concerned: the exact) number of components in a subsystem. In both cases, this yields an additional degree of freedom in subsystems, namely the non-observed components.

The first major difference of the Cross-Checking approach is that we do not try to compute the global transition relation. Instead, we compute at first the (smaller) transition systems of the subsystems and accept the imprecision due to the additional degrees of freedom. Only after having computed the reachable state spaces of the subsystems do we enhance the quality of our approximation by Cross-Checking.

The second major difference is that we do not (in contrast to all approaches mentioned above) try to compute a representation of the global state space (neither explicitly nor through BDDs), to prove a desired property on this representation. Instead, we apply a sufficient condition that – if valid on all reachable states of the subsystems – implies the desired property for the reachable global state space.

An additional distinguishing feature of our approach is the fact that it is fully automated, including the determination of the relevant subsystems, whereas the basic principle of partitioning the set of state variables into projections, e.g., in [GDHH98, GD98, GDB99, Gov00], relies on the user.

Approaches that are methodically further away from our Cross-Checking approach, but nevertheless should be mentioned here because they deal with the model of interaction systems which we use to present our ideas, can be found in [GS03, Sif05, GGM+07b, GGM+07a, BBS06, GQ07].

7.3 Concluding Remarks

We presented a method to obtain an enhanced over-approximation of the reachable global state space of a component-based system with n components in polynomial time. The method consists of choosing a parameter d, investigating subsystems consisting of d components in a first step (in $O(n^d \cdot m^d)$) and then improving this approximation by Cross-Checking (in $O(d \cdot n^d \cdot m^d)$). The computation of the first step can be improved in various respects. Firstly, for the purpose of proving deadlock-freedom we do not have to consider all $\binom{n}{d}$ subsystems but only the ones that are connected. Secondly, the computation of the various sets $Reach(Sys_{K'})$ can be performed in parallel. Moreover, we may combine these techniques with other methods as, e.g., using BDDs for the computation of the $Reach(Sys_{K'})$. Our approximation can be used to investigate global properties by considering subsystems and checking conditions on them which requires only polynomial time costs. We showed how this can be achieved for the property of local deadlock-freedom. The presented techniques have been implemented in our tool "PrInSESSA" [MS08] where in addition we also apply a variant of Cross-Checking for the detection of minimal large deadlocks. This allows us for our example $DP(6)$ with $n = 18$ and $d = 4$ to reduce the number of critical states to 24 (which accounts to 2 critical states up to symmetry). Also, for $n = 18$ and $d = 5$ our sufficient condition for deadlock-freedom is valid (i.e., there are no critical states at all) and we conjecture (based on the regular structure of the example's interaction graph) that $d = 5$ is sufficient to prove $DP(p)$ deadlock-free, independently of p.

Due to the fact that our tool PrInSESSA is still in a prototype state, empirical comparison with other approaches to prove properties by reachable state space approximation is not yet feasible. The empiric results provided in this thesis are rather meant to point out the potential of our approach.

Finally, we end this conclusion by enlisting and discussing the major features, respectively selling points of our approach.

- **Run-time is always bounded by a polynomial with degree d.** The parameter d for the subsystem size also serves as a setscrew to adjust accuracy vs. runtime. This is a concept that occurs similarly, e.g., in bounded model-checking, where the longer paths increase the time bounds of the algorithm as well as its chance of success. This makes the approach adjustable to any setting depending on the system size and the available time and computation power.

- **No restriction of system communication structure.** We do not yet exploit any system structure (e.g., symmetry). The choice of the symmetric examples used in this work might well divert the reader from the fact that we do not exploit (or in other words: do not rely on) symmetry. The symmetry that occurs in our examples simply serves the purpose of easy scalability. Our approach does work well with non-symmetric examples. Furthermore, symmetry could be exploited in future versions where it might be sufficient to investigate only one representative of an equivalence class of subsystems (and some of its neighbors to allow for Cross-Checking).

- **Applicability to other models.** We chose the model of interaction systems as a means of demonstration. However, the approach can easily be conferred upon other models, especially such models that feature multi-party synchronizations.

- **Combination with other approaches**. Hybrid approaches that try to unite the advantages of multiple techniques are very common in the setting of formal methods. Especially the Cross-Checking technique seems suited to be outsourced to enhance the approximation quality of other approaches.

  Also, if our approach is performed and is not able to prove a system to be deadlock-free on its own it will output critical states. It is self-evident that a combination with other approaches to refute the criticalness of the remaining states is promising. Finally, the BDD-representation of symbolic model-checking that is used to save both space and running time can be also applied in our setting and is currently used for a more sophisticated implementation.

- **Implementations are promising.** Our first prototype implementation PrInSESSA [MS08] already proved the potential of our approach. It showed the general possibility to prove large systems deadlock-free within a polynomial time bound and provided all empirical data that is used in this work. A recent, more sophisticated implementation based on BDDs allows us to handle 500 philosophers in $\sim$10min.

- **Easy distributability.** When reasoning about concurrent systems, it is an obvious idea to perform any kind of formal analysis in a distributed manner. In our case there are two steps that can be identified in this context: Firstly, we analyze $\binom{n}{d}$ subsystems for reachability. Secondly, we perform the Cross-Checking algorithm to compare them among each other. Both steps can easily be performed in parallel: Given k processing units $u_1, \ldots, u_k$ we could decompose the set of subsystems into k subsets $S_1, \ldots, S_k$, such that u_i performs the reachability analyses for S_i. As to Cross-Checking, we could at first glance use $d-1$ processing units $u_1, \ldots, u_{d-1}$, where u_i investigates substates of size $i+1$. Additionally, if more than $d-1$ processing units are available, one could further split up the substates of a common size, e.g., based on their lexicographic order.

- **White-box representation of components not necessary.** One aspect under which the different approaches described in Section 7.1 can well be compared is the degree of information that is needed to apply them. While abstraction usually requires a white-box representation of the system, half-order reduction on the other hand only requires (apart from the information which actions are independent from each other) a possibility to generate the global transition system in a stepwise manner (e.g., by requests to gray boxes). The same holds for our Cross-Checking technique. Additionally, we do not need information about interdependency of actions.

- **Potential for further development.** We presented a sufficient condition for deadlock-freedom of interaction systems. Generalizing our ideas, we suggested a framework where system properties that are defined by predicates on the reachable global state space are instead proved on subsystems. In such a framework, the ideas presented in this work can only be a starting point. Both, the approximation quality and the sufficient conditions seem to bear potential for improvement. Especially our local indicator predicates are still very strict: Blocking chains are only one possible characteristic of a deadlock and it seems worthwhile to try and build other abstractions from the participation graph to deduce necessary conditions for the existence of a large deadlock. Such additional characteristics could simply be checked additionally (cf. Remark 6.17) before we mark a state critical.

Finally, the presented framework is suited to modularly integrate other predicates, e.g., for liveness, availability, and other important properties of component-based systems.

Bibliography

[AB03] Alessandro Aldini and Marco Bernardo. A General Approach to Deadlock Freedom Verification for Software Architectures. In *Proceedings of FME'03*, LNCS 2805, pages 658–677, 2003.

[AC05] Paul Attie and Hana Chockler. Efficiently Verifiable Conditions for Deadlock-Freedom of Large Concurrent Programs. In *Proceedings of VMCAI'05*, LNCS 3385, pages 465–481, 2005.

[AG97] Robert Allen and David Garlan. A Formal Basis for Architectural Connection. *ACM Trans. Softw. Eng. Methodol.*, 6(3):213–249, 1997.

[AH01] Luca de Alfaro and Tom Henzinger. Interface Automata. In *Proceedings of ESEC/SIGSOFT FSE'01*, pages 109–120, 2001.

[Arb98] Farhad Arbab. What Do You Mean, Coordination? In *Bulletin of the Dutch Association for Theoretical Computer Science (NVTI)*, pages 11–22, 1998.

[Arb04] Farhad Arbab. Reo: a Channel-Based Coordination Model for Component Composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.

[Arn94] André Arnold. *Finite Transition Systems: Semantics of Communicating Systems.* Prentice Hall International (UK) Ltd., Hertfordshire, UK, 1994. Translator-John Plaice.

[BBLS93] Saddek Bensalem, Ahmed Bouajjani, Claire Loiseaux, and Joseph Sifakis. Property Preserving Simulations. In *CAV '92: Proceedings of the Fourth International Workshop on Computer Aided Verification*, pages 260–273, London, UK, 1993. Springer-Verlag.

[BBNS09] Saddek Bensalem, Marius Bozga, Thanh-Hung Nguyen, and Joseph Sifakis. D-Finder: A Tool for Compositional Deadlock Detection and Verification. In *CAV '09: Proceedings of the 21st International Conference on Computer Aided Verification*, pages 614–619, Berlin, Heidelberg, 2009. Springer-Verlag.

[BBS06] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *Proceedings of SEFM'06*, pages 3–12. IEEE Computer Society, 2006.

[BBSN08] Saddek Bensalem, Marius Bozga, Joseph Sifakis, and Thanh-Hung Nguyen. Compositional Verification for Component-Based Systems and Application. In *ATVA '08: Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis*, pages 64–79, Berlin, Heidelberg, 2008. Springer-Verlag.

[BCD02] Marco Bernardo, Paolo Ciancarini, and Lorenzo Donatiello. Architecting Families of Software Systems with Process Algebras.

ACM Trans. on Software Engineering and Methodology, 11:386–426, 2002.

[BGLZ05a] Mario Bravetti, Roberto Gorrieri, Roberto Lucchi, and Gianluigi Zavattaro. On the Expressiveness of Probabilistic and Prioritized Data-retrieval in Linda. *Electr. Notes Theor. Comput. Sci.*, 128(5):39–53, 2005.

[BGLZ05b] Mario Bravetti, Roberto Gorrieri, Roberto Lucchi, and Gianluigi Zavattaro. Quantitative Information in the Tuple Space Coordination Model. *Theor. Comput. Sci.*, 346(1):28–57, 2005.

[BGM00] Frank S. de Boer, Maurizio Gabbrielli, and Maria Chiara Meo. A Timed Linda Language. In *COORDINATION '00: Proceedings of the 4th International Conference on Coordination Languages and Models*, pages 299–304, London, UK, 2000. Springer-Verlag.

[BGZ00] Nadia Busi, Roberto Gorrieri, and Gianluigi Zavattaro. On the Expressiveness of Linda Coordination Primitives. *Inf. Comput.*, 156(1-2):90–121, 2000.

[BR91] S. Brookes and A. Roscoe. Deadlock Analysis in Networks of Communicating Processes. *Distributed Computing*, 4(4):209–230, 1991.

[CC77] Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on*

Principles of programming languages, pages 238–252, New York, NY, USA, 1977. ACM.

[CC92] Patrick Cousot and Rahida Cousot. Abstract Interpretation and Application to Logic Programs. *J. Log. Program.*, 13(2-3):103–179, 1992.

[CCK+05] Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Using Probabilistic I/O Automata to Analyze an Oblivious Transfer Protocol. Technical Report MIT-LCS-TR-1001, MIT CSAIL, Cambridge, MA, August 2005.

[CCQ94] Gianpiero Cabodi, P. Camurati, and Stefano Quer. Symbolic Exploration of Large Circuits With Enhanced Forward/Backward Traversals. In *EURO-DAC '94: Proceedings of the conference on European design automation*, pages 22–27, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[CEP93] Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity Results for 1-safe Nets. In *FST TCS'93*, LNCS 761, pages 326–337, 1993.

[CGL92] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model Checking and Abstraction. *Symposium on Principles of Programming Languages and Systems (POPL)*, pages 342–354, 1992.

[CGL94] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model Checking and Abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5):1512–1542, 1994.

[CHM$^+$] Hyunwoo Cho, Gary D. Hachtel, Enrico Macii, Bernard Plessier, and Fabio Somenzi. Algorithms for Approximate FSM traversal. In *DAC '93: Proceedings of the 30th International Design Automation Conference*.

[CHM93] Søren Christensen, Yoram Hirshfeld, and Faron Moller. Bisimulation Equivalence is Decidable for Basic Parallel Processes. In *CONCUR '93: Proceedings of the 4th International Conference on Concurrency Theory*, pages 143–157, London, UK, 1993. Springer-Verlag.

[CHM$^+$96] Hyunwoo Cho, Gary D. Hachtel, Enrico Macii, Massimo Poncino, and Fabio Somenzi. Automatic State Space Decomposition for Approximate FSM Traversal Based on Circuit Analysis. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 15(12):1451–1464, 1996.

[CJY95] P. Ciancarini, K. K. Jensen, and D. Yankelevich. On the Operational Semantics of a Coordination Language. In P. Ciancarini, O. Nierstrasz, and A. Yonezawa, editors, *Object-Based Models and Languages for Concurrent Systems: Proc. of the ECOOP'94 Workshop on Modles and Languages for Coordination of Parallelism and Distribution*, pages 77–106. Springer, Berlin, Heidelberg, 1995.

[EC82] E. Allen Emerson and Edmund M. Clarke. Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons. *Science of Computer Programming*, 2(3):241–266, December 1982.

[Flo67] R. W. Floyd. Assigning Meaning to Programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science: Proc. American Mathematics Soc. Symposia*, volume 19, pages 19–31, Providence RI, 1967. American Mathematical Society.

[GD98] Shankar G. Govindaraju and David L. Dill. Verification by Approximate Forward and Backward Reachability. In *ICCAD '98: Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design*, pages 366–370, New York, NY, USA, 1998. ACM.

[GDB99] Shankar G. Govindaraju, David L. Dill, and Jules P. Bergmann. Improved Approximate Reachability Using Auxiliary State Variables. In *DAC '99: Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, pages 312–316, New York, NY, USA, 1999. ACM.

[GDHH98] Shankar G. Govindaraju, David L. Dill, Alan J. Hu, and Mark A. Horowitz. Approximate Reachability with BDDs Using Overlapping Projections. In *DAC '98: Proceedings of the 35th Annual Design Automation Conference*, pages 451–456, New York, NY, USA, 1998. ACM.

[GGM$^+$07a] Gregor Goessler, Susanne Graf, Mila Majster-Cederbaum, Moritz Martens, and Joseph Sifakis. An Approach to Modelling and Verification of Component Based Systems. In *Proceedings of SOFSEM'07*, LNCS 4362, 2007.

[GGM$^+$07b] Gregor Goessler, Susanne Graf, Mila Majster-Cederbaum, Moritz Martens, and Joseph Sifakis. Ensuring Properties of

Interaction Systems by Construction. In *Program Analysis and Compilation, Theory and Practice*, LNCS 4444, pages 201–224, 2007.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[GL80] David Gries and Gary Levin. Assignment and Procedure Call Proof Rules. *ACM Trans. Program. Lang. Syst.*, 2(4):564–579, 1980.

[GL81] David Gries and Gary Levin. A Proof Technique for Communicating Sequential Processes. *Acta Informatica*, 15(3):281–302, 1981.

[Gov00] Gaurishankar Govindaraju. *Approximate Symbolic Model Checking Using Overlapping Projections*. PhD thesis, Stanford University, Stanford, CA, USA, 2000. Adviser-Dill, David L.

[GQ07] S. Graf and S. Quinton. Contracts for BIP: Hierarchical Interaction Models for Compositional Verification. In *Proceedings of FORTE'07*, LNCS 4574, pages 1–18, 2007.

[GS03] Gregor Goessler and Joseph Sifakis. Component-based Construction of Deadlock-free Systems. In *Proceedings of FSTTCS'03*, LNCS 2914, pages 420–433, 2003.

[GS05] Gregor Goessler and Joseph Sifakis. Composition for Component-based Modeling. *Sci. Comput. Program.*, 55(1-3):161–183, 2005.

[GW92] Patrice Godefroid and Pierre Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In *CAV '91: Proceedings of the 3rd International Workshop on Computer Aided Verification*, pages 332–342, London, UK, 1992. Springer-Verlag.

[Hir94] Yoram Hirshfeld. Petri Nets and the Equivalence Problem. In *CSL '93: Selected Papers from the 7th Workshop on Computer Science Logic*, pages 165–174, London, UK, 1994. Springer-Verlag.

[Hoa69] C. A. R. Hoare. An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 12(10):576–580, October 1969.

[IU01] Paola Inverardi and Sebastian Uchitel. Proving Deadlock Freedom in Component Based Programming. In *Proceedings of ETAPS'01*, LNCS 2029, pages 60–75, 2001.

[Jan94] Petr Jancar. Decidability Questions for Bisimilarity of Petri Nets and Some Related Problems. In *Proceedings of STACS94, Springer-Verlag, LNCS*, pages 581–592. Springer-Verlag, 1994.

[JKKO98] Cheoljoo Jeong, Youngchan Kim, Heungnam Kim, and Youngbae Oh. A faster parallel implementation of the kanellakis-smolka algorithm for bisimilarity checking. In *In Proceedings of the International Computer Symposium*, 1998.

[KBLD08] Jakub Kurzak, Alfredo Buttari, Piotr Luszczek, and Jack Dongarra. The PlayStation 3 for High-Performance Scientific Computing. *Computing in Science and Engg.*, 10(3):84–87, 2008.

[KLSV06] Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. *The Theory of Timed I/O Automata*. Morgan & Claypool Publishers, 2006.

[KMM00] Matt Kaufmann, Panagiotis Mandios, and J. Strother Moore. ACL2 essentials. *Computer-Aided Reasoning: ACL2 Case Studies*, pages 27–37, 2000.

[KS90] Paris C. Kanellakis and Scott A. Smolka. Ccs expressions finite state processes, and three problems of equivalence. *Inf. Comput.*, 86(1):43–68, 1990.

[Lam09] Christian Lambertz. Exploiting Architectural Constraints and Branching Bisimulation Equivalences in Component-Based Systems. In *Proceedings of FM'09, Doctoral Symposium*, 2009.

[LGS$^+$95] Claire Loiseaux, Susanne Graf, Joseph Sifakis, Ahmed Bouajjani, and Saddek Bensalem. Property Preserving Abstractions for the Verification of Concurrent Systems. In *Formal Methods in System Design*, volume 6, pages 11–44, 1995.

[LPJ$^+$96] Woohyuk Lee, Abelardo Pardo, Jae-Young Jang, Gary Hachtel, and Fabio Somenzi. Tearing Based Automatic Abstraction for CTL Model Checking. In *ICCAD '96: Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, pages 76–81, Washington, DC, USA, 1996. IEEE Computer Society.

[MBB$^+$95] Zohar Manna, Nikolaj Bjrner, Anca Browne, Edward Chang, Michael Coln, Luca Alfaro, Harish Devarajan, Arjun Kapur, Jaejin Lee, Henny Sipma, and Toms Uribe. STeP: The Stanford

Temporal Prover. In *TAPSOFT '95: Theory and Practice of Software Development*, pages 793–794, Berlin, Heidelberg, 1995. Springer-Verlag.

[McM92] Kenneth Lauchlin McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1992.

[Mil89] Robin Milner. *Communication and Concurrency.* Prentice/Hall, 1989.

[Min67] Marvin L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[Min07] Christoph Minnameier. Local and Global Deadlock-Detection in Component Based Systems are NP-hard. *Information Processing Letters*, 103(3):105–111, 2007.

[MJH$^+$98] In-Ho Moon, Jae-Young Jang, Gary D. Hachtel, Fabio Somenzi, Jun Yuan, and Carl Pixley. Approximate Reachability Don't Cares for CTL Model Checking. In *ICCAD '98: Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design*, pages 351–358, New York, NY, USA, 1998. ACM.

[MM06] Mila E. Majster-Cederbaum and Christoph Minnameier. Termination and Divergence Are Undecidable Under a Maximum Progress Multi-step Semantics for LinCa. In *Proceedings of ICTAC'06*, LNCS 4281, pages 65–79, 2006.

[MM08a] Mila Majster-Cederbaum and Moritz Martens. Compositional Analysis of Deadlock-Freedom for Tree-like Component Architectures. In *EMSOFT '08: Proceedings of the 8th ACM International Conference on Embedded Software*, pages 199–206, New York, NY, USA, 2008. ACM.

[MM08b] Mila Majster-Cederbaum and Christoph Minnameier. Deriving Complexity Results for Interaction Systems from 1-safe Petri Nets. In *Proceedings of SOFSEM'08*, LNCS 4910, pages 352–363, 2008.

[MM08c] Mila Majster-Cederbaum and Christoph Minnameier. Everything Is PSPACE-Complete in Interaction Systems. In *Proceedings of the 5th International Colloquium on Theoretical Aspects of Computing*, pages 216–227, Berlin, Heidelberg, 2008. Springer-Verlag.

[MM09] Mila Majster-Cederbaum and Moritz Martens. Using Architectural Constraints for Deadlock-Freedom of Component Systems with Multiway Cooperation. In *TASE '09: 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering*, 2009.

[MMM06] Mila Majster-Cederbaum, Moritz Martens, and Christoph Minnameier. Deciding Liveness in Interaction Systems is NP-hard. Technical report, University of Mannheim, 2006.

[MMM07a] Mila Majster-Cederbaum, Moritz Martens, and Christoph Minnameier. A Polynomial-Time-Checkable Sufficient Condition for Deadlock-freeness of Component Based Systems. In *Proceedings of the 33rd International Conference on Current Trends in*

Theory and Practice of Computer Science, SOFSEM07, LNCS 4362, 2007.

[MMM07b] Mila Majster-Cederbaum, Moritz Martens, and Christoph Minnameier. Liveness in Interaction Systems. In *Proceedings of FACS'07*, ENTCS, 2007.

[Moo65] G. E. Moore. Cramming More Components Onto Integrated Circuits. *Electronics*, 38, 1965.

[MS08] Christoph Minnameier and Rouven Schaube. PrInSESSA - Proving Properties of Interaction Systems by Enhanced State Space Approximation, 2008.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[OS08] Sam Owre and Natarajan Shankar. A Brief Overview of PVS. In *TPHOLs '08: Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics*, pages 22–27, Berlin, Heidelberg, 2008. Springer-Verlag.

[Owi75] Susan Speer Owicki. *Axiomatic Proof Techniques for Parallel Programs.* PhD thesis, Cornell University, Ithaca, NY, USA, 1975.

[Owi79] Susan S. Owicki. Specifications and Proofs for Abstract Data Types in Concurrent Programs. In *Program Construction, International Summer School*, pages 174–197, London, UK, 1979. Springer-Verlag.

[PM93] Christine Paulin-Mohring. Inductive Definitions in the System Coq - Rules and Properties. In *TLCA '93: Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 328–345, London, UK, 1993. Springer-Verlag.

[PT87] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and Verification of Concurrent Systems in CESAR. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, pages 337–351, London, UK, 1982. Springer-Verlag.

[Reu90] Christophe Reutenauer. *The Mathematics of Petri Nets.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[RMSS98] Kavita Ravi, Kenneth L. McMillan, Thomas R. Shiple, and Fabio Somenzi. Approximation and Decomposition of Binary Decision Diagrams. In *DAC '98: Proceedings of the 35th Annual Design Automation Conference*, pages 445–450, New York, NY, USA, 1998. ACM.

[RO01] Cleaveland R. and Sokolsky O. Equivalence and preorder checking for finite-state systems. *Handbook of Process Algebra*, pages 391–424, 2001.

[RS95] Kavita Ravi and Fabio Somenzi. High-Density Reachability Analysis. In *ICCAD '95: Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design*, pages 154–158, Washington, DC, USA, 1995. IEEE Computer Society.

[San77] Luis E. Sanchis. Data Types as Lattices: Retractions, Closures and Projections. *ITA*, 11(4):329–344, 1977.

[Sav70] Walter Savitch. Relationships between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[Sif05] Joseph Sifakis. A Framework for Component-based Construction. In *Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods*, pages 293–300. IEEE Computer Society, 2005.

[SP96] Elad Shahar and Prof Amir Pnueli. The TLV System and its Applications, 1996.

[SS63] J. C. Shepherdson and H. E. Sturgis. Computability of Recursive Functions. *J. ACM*, 10(2):217–255, 1963.

[Tan08] Andrew Tanenbaum. *Modern Operating Systems*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 2008. Third Edition.

[Var09] Moshe Vardi. Model-Checking as a Reachability Problem, Invited Talk. In *RP '09: Proceedings of the 3rd International Workshop on Reachability Problems*, page 35, Berlin, Heidelberg, 2009. Springer-Verlag.

[YSR09] S. Han Yunghsiang, Omiwade Soji, and Zheng Rong. Survivable Distributed Storage with Progressive Decoding. Technical report uh-cs-09-09, University of Houston, 2009.

Appendix A

A.1 A simple Example for enc_{MTS}

In Figure A.1, we give an example for the transition systems $MTS[P]$ and $MTS\text{-}mp[enc_{MTS}(P)]$. Please note that (violating formal correctness) we let the tuples c, d, e occur in the labels of the displayed transition systems. These tuples should actually never occur in labels but it seems convenient to include them here for ease of understanding. Labels that consist exclusively of such tuples should actually be replaced by the label τ.

Let $P := out(a) \mid out(b)$.

Then $\text{enc}_{MTS}(\text{P}) =$

$$\begin{aligned} & in(c).out(a) \mid in(c).out(b) \\ & \mid\, !\, in(d).[rd(e).out(c) \mid out(d)] \\ & \mid\, !\, in(d).out(e).wait.in(e).wait.out(d) \\ & \mid out(d) \end{aligned}$$

The labeled nodes in Figure A.1 correspond to the following states, where the *weak simulation* relation S used in the proof of Theorem 3.4 in Section 3.2 would in this case be
$S = \{(1, 1^*), (1, 1'), (2, 2'), (3, 3'), (4, 4'), (5, 5'), (6, 6')\})$.

1: $< out(a) \mid out(b), \emptyset >$

1*: $< enc_{MTS}(out(a) \mid out(b)), \emptyset >$

$1'$: $< \widetilde{enc}_{MTS}(out(a) \mid out(b)), \{d\} >$

2: $< out(b), \{a\} >$

$2'$: $< \widetilde{enc}_{MTS}(out(b)), \{a, d\} >$

3: $< out(a), \{b\} >$

$3'$: $< \widetilde{enc}_{MTS}(out(a)), \{b, d\} >$

4: $< 0, \{a, b\} >$

$4'$: $< \widetilde{enc}_{MTS}(0), \{a, b, d\} >$

5: $< 0, \{a, b\} >$

$5'$: $< \widetilde{enc}_{MTS}(0), \{a, b, d\} >$ (not in the picture for better readability)

6: $< 0, \{a, b\} >$

$6'$: $< \widetilde{enc}_{MTS}(0), \{a, b, d\} >$ (not in the picture for better readability)

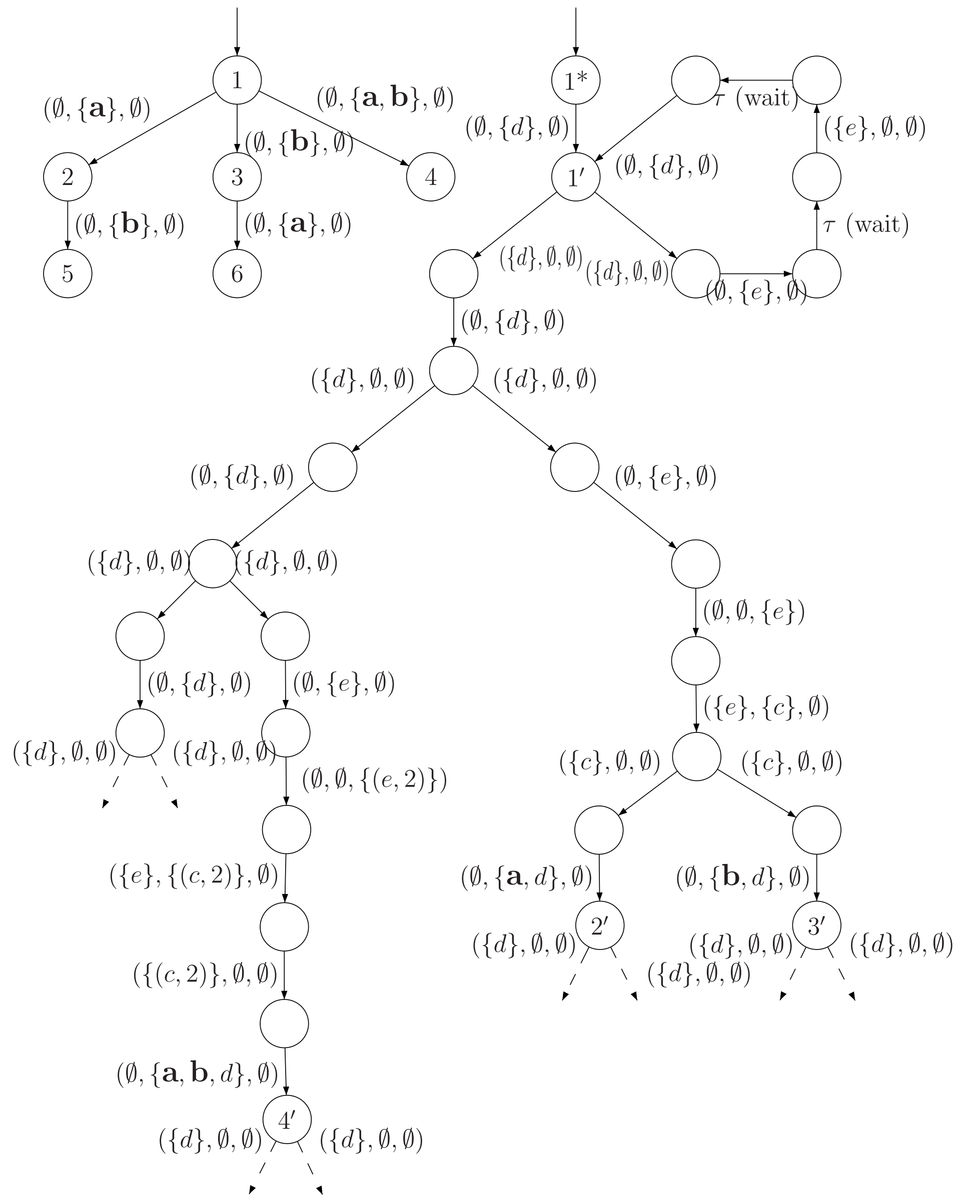

1
2
3
4
5
6
$(\emptyset, \{\mathbf{a}\}, \emptyset)$
$(\emptyset, \{\mathbf{b}\}, \emptyset)$
$(\emptyset, \{\mathbf{a}, \mathbf{b}\}, \emptyset)$
$(\emptyset, \{\mathbf{b}\}, \emptyset)$
$(\emptyset, \{\mathbf{a}\}, \emptyset)$
1*
$1'$
$(\emptyset, \{d\}, \emptyset)$
$(\emptyset, \{d\}, \emptyset)$
τ (wait)
$(\{e\}, \emptyset, \emptyset)$
τ (wait)
$(\{d\}, \emptyset, \emptyset)$
$(\{d\}, \emptyset, \emptyset)$
$(\emptyset, \{e\}, \emptyset)$
$(\emptyset, \{d\}, \emptyset)$
$(\{d\}, \emptyset, \emptyset)$
$(\{d\}, \emptyset, \emptyset)$
$(\emptyset, \{d\}, \emptyset)$
$(\emptyset, \{e\}, \emptyset)$
$(\{d\}, \emptyset, \emptyset)$
$(\{d\}, \emptyset, \emptyset)$
$(\emptyset, \emptyset, \{e\})$
$(\emptyset, \{d\}, \emptyset)$
$(\emptyset, \{e\}, \emptyset)$
$(\{e\}, \{c\}, \emptyset)$
$(\{d\}, \emptyset, \emptyset)$
$(\{d\}, \emptyset, \emptyset)$
$(\emptyset, \emptyset, \{(e, 2)\})$
$(\{c\}, \emptyset, \emptyset)$
$(\{c\}, \emptyset, \emptyset)$
$(\{e\}, \{(c, 2)\}, \emptyset)$
$(\emptyset, \{\mathbf{a}, d\}, \emptyset)$
$(\emptyset, \{\mathbf{b}, d\}, \emptyset)$
$2'$
$3'$
$(\{d\}, \emptyset, \emptyset)$
$(\{d\}, \emptyset, \emptyset)$
$(\{d\}, \emptyset, \emptyset)$
$(\{d\}, \emptyset, \emptyset)$
$(\{(c, 2)\}, \emptyset, \emptyset)$
$(\emptyset, \{\mathbf{a}, \mathbf{b}, d\}, \emptyset)$
$4'$
$(\{d\}, \emptyset, \emptyset)$
$(\{d\}, \emptyset, \emptyset)$

Figure A.1: $MTS[out(a) \mid out(b)]$ and $MTS\text{-}mp[enc_{MTS}(out(a) \mid out(b))]$

Appendix B

B.1 Sample Data derived from our Tool PrInSESSA

Reach-CC	Uncrit-CC	# of reach. substates	# of crit. substates	Percentage
		185,883	1,584	0.85%
x		81,534	432	0.5%
	x	185,883	342	0.18%
x	x	81,534	24	0.03%

Figure B.1: Critical substates in the system DP(6) for $d = 4$ depending on which variants of Cross-Checking we apply.

B.2 Checking for small Deadlocks

Here, we are going into more detail concerning how checking a single substate for deadlocks is done efficiently.

Let $D = \{i_1, \ldots, i_k\}$ be a (not necessarily minimal) deadlock of size $|D| = k \leq d$ in some global state q. According to our deadlock definition, for all interactions α in which at least one component $i \in D$ participates there is at least one component $j \in D$ that participates in α and does not enable its corresponding action in q_j.

In order to prove that there is no deadlock of size $|D| = k \leq d$ in any reachable global state q, we loop over all substates of size $< d$ (cf. Algorithm 6) and for those that are marked reachable in the reachability array of the corresponding subsystem, we check (cf. Algorithm 7) whether there is at least one interaction in which one of the components participates and enables its corresponding action and for which neither of the others participates and denies its corresponding action. If such an interaction can be found for $(q_{i_1}, \ldots, q_{i_x})$, then $(q_{i_1}, \ldots, q_{i_x})$ cannot be a deadlock. If such an interaction is found for every substate $(q_{i_1}, \ldots, q_{i_x})$ then there is no small deadlock in *Sys*. The number of possible substates of size $< d$ is in $O(n^d \cdot m^d)$, thus in particular the number of substates we loop over is in $O(n^d \cdot m^d)$. As we have to check all interactions and the actions therein (cf. Algorithm 7) for every reachable substate, the complexity for our search for small deadlocks is bounded by $O(n^d \cdot m^d \cdot \sum_{\alpha \in Int} |\alpha|)$.

Algorithm 6 Check for small Deadlocks

```
1:  **for** $x := 2$ to $d$ **do**
2:      **for** all subsets $K'' = \{i_1, \ldots, i_x\}$ of $K$ **do**
3:          **for** all $q'' = (q_{i_1}, \ldots, q_{i_x}) \in Q_{K''}$ **do**
4:              reachable := true;
5:              **for** all subsystems $Sys_{K'}$ with $K'' \subseteq K'$ (and $|K'| = d$) **do**
6:                  **for** all $q' \in Ext(q'', K')$ **do**
7:                      **if** $reach(Sys_{K'})[q']$ **then**
8:                          Check for Deadlock(q')
9:                      **end if**
10:                 **end for**
11:             **end for**
12:         **end for**
13:     **end for**
14: **end for**
```

Algorithm 7 Check for Deadlock($q' = (q_{i_1}, \ldots, q_{i_x})$)

```
1:  for all $\alpha = \{a_{j_1}, \ldots, a_{j_k}\} \in Int$ do
2:      Participation := False;
3:      Denial := False;
4:      for $l = 1$ to $k$ do
5:          if $j_l \in \{i_1, \ldots, i_x\}$ then
6:              if $a_{j_l} \in ea(q_{j_l})$ then
7:                  Participation = True;
8:              else
9:                  Denial = True;
10:             end if
11:         end if
12:     end for
13:     if (Participation AND not(Denial)) then
14:         Break;          ▷ Substate cleared. Proceed with the next substate.
15:     end if
16: end for
17: if (not(Participation) OR Denial) then
18:     Output "Local deadlock in $(q_{i_1}, \ldots, q_{i_x})$";
19: end if
```

Index