**Basic Research in Computer Science**

**BRICS DS-96-2 A. Cheng: Reasoning About Concurrent Computational Systems**

# Reasoning About Concurrent Computational Systems

**Allan Cheng**

**BRICS Dissertation Series** **DS-96-2**

**ISSN 1396-7002** **August 1996**

**See back inner page for a list of recent publications in the BRICS Dissertation Series. Copies may be obtained by contacting:**

**BRICS**
**Department of Computer Science**
**University of Aarhus**
**Ny Munkegade, building 540**
**DK - 8000 Aarhus C**
**Denmark**

**Telephone: +45 8942 3360**
**Telefax: +45 8942 3255**
**Internet: BRICS@brics.dk**

**BRICS publications are in general accessible through WWW and anonymous FTP:**

`http://www.brics.dk/`
`ftp ftp.brics.dk (cd pub/BRICS)`

# Reasoning About Concurrent Computational Systems

Allan Cheng

Ph.D. Dissertation


Department of Computer Science
University of Aarhus
Denmark

# Reasoning About Concurrent Computational Systems

A Dissertation
Presented to the Faculty of Science
of the University of Aarhus
in Partial Fulfilment of the Requirements for the
Ph.D. Degree

by
Allan Cheng
January 1996

# Abstract

This thesis contains three main parts. The first part presents contributions in the field of verification of finite state concurrent systems, the second part presents contributions in the field of behavioural reasoning about concurrent systems based on the notions of behavioural preorders and behavioural equivalences, and, finally, the third part presents contributions in the field of set constraints.

Concurrent computational systems are any computational systems that consist of autonomous processes, each performing some tasks while possibly communicating with one-another. Concurrent systems are widely used in real-life; airline-booking systems, communication protocols, operating systems for computers, just to name a few. The study of concurrent systems deals with formal ways of reasoning about such systems; one aspect is the development of mathematical models of concurrency, another is formal methods for verification of systems described as such mathematical models. Part of the research community has focussed on the semantical study of concurrency; numerous models, such as Synchronisation Trees, Labelled Transition Systems, Petri nets, and Event Structures, have been proposed, investigated, and compared, all trying to capture, at some level of abstraction, the notion of concurrency or concurrent behaviour. Another part has focussed on developing verification methods for some of these models. Algorithms have been based on a firm understanding of the models and ways of formally expressing or specifying a desired "correct" behaviour.

Model-checking algorithms rely on temporal logics, such at PLTL (Propositional Linear Temporal Logic) and CTL (Computational Tree Logic), or on automata as a specification language, while other algorithms rely on establishing behavioural relations between specifications and implementations. Behavioural equivalences, such as bisimulations, have played a major role in the process-algebra community, who are well-known for their modular approach to implementation and verification, based on congruence properties of behavioural equivalences.

At the same time, computational complexity issues have played an important role; both from the practical point view—developing fast algorithms—and from the theoretical point of view—e.g., "is it feasible to choose a more expressive logic for the verification task ?".

The intense study of both temporal logics and behavioural equivalences the last 15 years is directly related to the study of concurrent systems.

In the first part, we start by studying the computational complexity of several standard verification problems for 1-safe Petri nets and some of its subclasses. We prove that reachability, liveness, and deadlock problems are all PSPACE-complete for 1-safe nets. We complete the picture by proving, among other things, that deadlock is NP-complete for free-choice nets and for 1-safe free-choice nets and that for arbitrary Petri nets, deadlock is equivalent to reachability and liveness. Our results provide the *first systematic study* of the computational complexity of these problems for 1-safe nets.

We then investigate the computational complexity of a more general verification problem, model-checking, when an instance of the problem consist of a formula and a description of a system whose state space is at most exponentially larger than the description. Based on Turing machines, we define *compact systems* as a general formalisation of such system descriptions. Examples of such compact systems are $K$-bounded Petri nets and synchronised automata. We

present polynomial space upper bounds for the model-checking problem over compact systems and the logics CTL and $\mathsf{L}(X, U, S)$. For many instances of compact systems, the above model-checking problems have PSPACE-hard lower bounds. Our *general upper bounds* provide the matching upper bounds.

We continue by considering the problem of performing model-checking relative to a *partial order semantics* of concurrent systems, in which not all possible sequences of actions are considered relevant. By taking progress fairness assumptions into account one obtains a more realistic view of the behaviour of the systems. We present P-CTL, a CTL-like logic, which is interpreted over a partial order semantics for labelled 1-safe nets. It turns out that Mazurkiewicz trace theory provides a natural partial order semantics, in which the progress fairness assumptions can be formalised. We provide the first, to the best of our knowledge, set of *sound and complete tableau rules* for a CTL-like logic interpreted under progress fairness assumptions. Furthermore, we also present a state labelling based model-checking algorithm for P-CTL, extensions of P-CTL with modal operators expressing concurrent or conflicting behaviour, and computational complexity and undecidability results.

After these investigations in the field of verification of finite state concurrent systems, we turn to behavioural equivalences over models of (concurrent) computation.

In the second part, we start by investigating Joyal, Nielsen, and Winskel's proposal of spans of open maps as an abstract category-theoretic way of adjoining a bisimulation equivalence, $\mathcal{P}$-bisimilarity, to a category of models of computation $\mathcal{M}$. We show that a representative selection of *well-known* bisimulations and behavioural equivalences such as, e.g., trace equivalence, weak bisimulation, Hennessy's testing equivalence, Milner and Sangiorgi's barbed bisimulation, and Larsen and Skou's probabilistic bisimulation, can be captured in the setting of spans of open maps. Hence, Joyal, Nielsen, and Winskel's proposed *notion of open maps seems successful*. We also examine some "true concurrency" equivalences, in the context of the theory of open maps, and discuss decidability issues.

An issue left open by Joyal, Nielsen, and Winskel's work on open maps was the *congruence properties* of behavioural equivalences. We address the following fundamental question: given a category of models of computation $\mathcal{M}$ and a category of observations $\mathcal{P}$, are there any conditions under which algebraic constructs viewed as functors preserve $\mathcal{P}$-bisimilarity? We define the notion of functors being *$\mathcal{P}$-factorisable* and show how this ensures that $\mathcal{P}$-bisimilarity is a congruence with respect to such functors. Guided by the definition of $\mathcal{P}$-factorisability we show how it is possible to *parametrise proofs* of functors being $\mathcal{P}$-factorisable with respect to the category of observations $\mathcal{P}$, i.e., with respect to a behavioural equivalence.

In the last part we then, almost, leave the field of concurrency to investigate set constraints.

Set constraints are inclusion relations between expressions denoting sets of ground terms over a ranked alphabet. They are the main ingredient in set-based program analysis. They are typically derived from the syntax of a program and solutions to them can yield useful information for, e.g., type inference, implementations, and optimisations.

We provide a *complete Gentzen-style axiomatisation* for sequents $\Phi \vdash \Psi$, where $\Phi$ and $\Psi$ are finite sets of set constraints, based on the axioms of termset algebra. Sequents of the restricted form $\Phi \vdash \bot$ correspond to positive set constraints, and those of the more general form $\Phi \vdash \Psi$ correspond to systems of mixed positive and negative set constraints. We show that the deductive system is complete for the restricted sequents $\Phi \vdash \bot$ over standard models, incomplete for general sequents $\Phi \vdash \Psi$ over standard models, but complete for general sequents over set-theoretic termset algebras.

In a brief section, we sketch how Milner's protocol can be verified for the absence of deadlocks using Kozen's `clp(sc)`, a constraint logic programming language over set constraints.

We then continue by investigating Kozen's *rational spaces*. Rational spaces are topological spaces obtained as spaces of runs of topological $\Sigma$-hypergraphs. They were introduced by Kozen who showed how the topological structure of the spaces of solutions to systems of set constraints can be given in terms of rational spaces. We give a Myhill-Nerode-like characterisation of rational points, which in turn is used to re-derive results about the rational points of finitary rational spaces. We show that the rational points in finitary rational spaces in some sense exactly capture the topological structure of the space. We define and investigate congruences on $\Sigma$-hypergraphs, and finally we determine the computational complexity of some decision problems related to rational spaces.

# Bibliographic Sketch

*Learning is a treasure that will follow its owner everywhere.*
**Chinese Proverb**

Allan was born March 5, 1969 in Silkeborg, Denmark. He spent his infancy in Switzerland on a tricycle and in Egypt trying to train straying dogs on the beaches of Alexandria. Realising that he had little luck with these friendly loafers, he decided to grow up, attending elementary schools in Congo, France, and Denmark. In his quest for learning, he entered Silkeborg Amtsgymnasium, in the summer of 1985. Three years later, he left the secure domestic surroundings, to find himself stranded in the big city of Aarhus. There, he discovered the intriguing beauty of mathematics and challenges of computer science. Despite many distractions, he finally settle down with a graduate study in computer science. He got very—perhaps too—accustomed to the departmental life, pleasant as it was. To his luck, he looked far across the pond, and spotted an opportunity to visit the Finger Lakes. He chose to reside in Ithaca, and managed to disguise himself as a visiting student at Cornell University's Department of Computer Science. The exposure to campus life at Cornell made profound changes in him. He watched, learned, and enjoyed the company of what became dear and close friends. After returning to Aarhus, he was awarded a Ph.D. degree for this thesis. His tale continues, but must be told elsewhere.

*To my Parents*

*For their love and support, which gave me all the opportunities.*

# Acknowledgements

*It was the Best of Times, It was the Worst of Times ...*
**Charles Dickens**, A Tale of Two Cities

First and foremost I thank my advisor Mogens Nielsen; his enthusiasm has been a constant source of inspiration and energy. He introduced me to the intriguing field of concurrency and has always been willing to discuss work and ideas. He has also given me the freedom to let myself be distracted by non-concurrency related topics. In Mogens I found a perfect mixture of guidance, support, and friendship, which made working with him a very enjoyable experience.

I would also like to thank the members of my committee, Kim Guldstrand Larsen, Ugo Montanari, and Erik Meineche Schmidt, for interesting comments and discussions during my defence.

I thank Javier Esparza and Jens Palsberg for making the writing of my first paper a memorable experience. I particularly enjoyed sharing office with Jens; it was so much fun and his optimism was very contagious. I also thank Madhavan Mukund and P.S. Thiagarajan, whose company has always been a pleasure.

At the department, I would also thank my friends, colleagues, and officemates, Kian Abolfazlian, Lars Arge, Torben Braüner, Dany Breslauer, Gerth S. Brodal, Ole Caprani, Søren Christensen, Bettina Blaaberg Clausen, Devdatt Dubhashi, Uffe Engberg, Gudmund Frandsen, Thore Husfeldt, Claus Torp Jensen, Nils Klarlund, Kjeld H. Mortensen, Morten K. Nielsen, Vladimiro Sassone, Erik Meineche Schmidt, Michael Schwartzbach, Morten Hoffmann Sørensen, Kim Sunesen, Igor Walukiewicz, Glynn Winskel, and Peter Ørbæk, with whom I have shared many hours of interesting discussions and fun. I would also like to thank our helpful secretaries Susanne Brøndberg, Janne Damgaard, Marianne Iversen, Karen Møller, and Charlotte Nielsen.

I am very grateful to Dexter Kozen for giving me the opportunity to visit him at Cornell University; I value my stay immensely. I enjoyed working in his company and got the chance to experience his sage and, yet, youthful nature; it impressed me.

I also enjoyed working, joking and laughing (a lot), and travelling with Ashvin Dsouza, for which I thank him. I thank my friends and housemates Deborah Swartz, Jessica Greenstein, Kjartan Stefánsson, Shanthi Subramanian, Funda Ergun, and especially Katie Guo and Andy Chen for making me feel "at home".

Special thanks goes to Heike Hüttner for her encouragements and vitality.

I thank and admire my dear brothers Ivan and Erik who kept me writing this dissertation when my thoughts were elsewhere.

I also express my gratitude for the financial support from BRICS, the Danish Research Academy, and the Danish Research Councils.

I am however, by far, most grateful to my parents for giving me the opportunities, their support, and their love. To them I dedicate this thesis.

The process of creating this dissertation has indeed been full of joy and frustration.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Structure

This thesis contains three main parts, each presenting results of my research during my graduate study. The chapters are presented in somewhat chronological order, reflecting how my interests broadened over time. Part I and II present work on reasoning about concurrent systems, which was mostly done during my stay in Aarhus. Part III presents my latest work, done while visiting Cornell University.

## 1.2 Contents of Part I

Part I presents contributions in the field of verification of finite state concurrent systems.

### 1.2.1 On Model-Checking

Formal verification of concurrent systems is based on the choice of a "suitable" formal semantical description of the systems and techniques for proving that these descriptions have certain properties. By "suitable" we mean that at some desired level of abstraction the formal semantical description captures (in fact defines) what we understand by the behaviour of the concurrent systems.

A predominant verification technique is known as *model-checking*. The approach is as follows. The systems one considers either explicitly or implicitly specify state spaces which are (labelled) graphs. Viewing these graphs as so-called Kripke structures (or just structures), one can interpret temporal logic formulas over them. Hence, *temporal logics* can be viewed as *specification languages* [Pnu77]. For a nice introduction, see [Eme90]. The problem of verifying if a system $s$ satisfies a property encoded in a formula $\varphi$ then reduces to the task of checking if the formula is satisfied in the system's state space regarded as a Kripke structure. The encoded properties often express that a system has some desirable behaviour, i.e., it behaves "correctly" in some respect. Model-checking algorithms perform this task automatically for finite state concurrent systems. The

approaches are rather different and based on, e.g, tableaux methods [Lar88, SW89], state labelling methods and graph algorithms [SC85, CES86, Mil89, CPS89], automata theory [VW86, Kup95], partial order semantics [Val90, WG93], BDDs [BCM+92], and "partial model-checking" (gradual elimination of a system while simultaneously transforming the specification formula) [And95]. Infinite state systems can also be model-checked, see [Bra91, BS92]. However, the procedure is no longer fully automatic.

### 1.2.2 Petri Nets

Petri nets are discrete models of concurrent systems and were introduced by C.A. Petri [Pet62]. They lend themselves easily to the modelling of concurrent systems and are widely used, partly due to their ability to model "true concurrency", and partly due to their nice graphical representation, which invites one to play the "token game".

A Petri net consists of a static part—a finite directed bipartite graph whose two disjoint node sets, $P$ and $T$, are called the *places* and the *transitions*—commonly referred to as the topology of the Petri net, and a dynamic part—a function $M : P \longrightarrow \mathbb{N}$ mapping each place to a natural number, called a *marking*—which represents the current *distributed state* of the Petri net. Below is an example of a so-called 1-safe net. It has four places, $p_1, \ldots, p_4$, illustrated as circles, and three transitions, $t_1, \ldots, t_3$, illustrated as boxes. The initial marking is illustrated by the black "tokens", indicating that only $p_1$ and $p_2$ are "marked"; they have one token each.

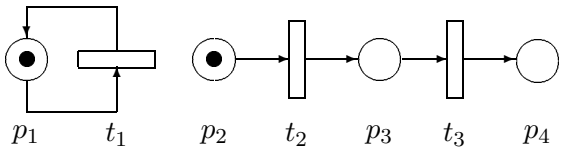



Figure 1.1: Example Petri net.

The marking of a Petri net can be changed when a transition "*fires*". For a transition to be able to fire at a given marking, all its input places—which are determined by the directed edges going into the transition—must be "marked". The effect of the transition firing is that a new marking is reached, by removing from the current marking one token from each input place and adding one token to each output place. In the above 1-safe net $t_1$ and $t_2$ may fire at the illustrated marking. If $t_1$ fires, the marking remains unchanged, while if $t_2$ fires, $p_2$ becomes unmarked and $p_3$ gets one token. A consecutive sequence of firing transitions is referred to as a firing sequence and may be viewed as a computation of the Petri net. The notation $M \xrightarrow{\sigma} M'$ denotes that from marking $M$ it is possible to sequentially fire the transitions in $\sigma$ and reach the marking $M'$. So, a computation of the Petri net changes the current marking according to its topology. The behaviour of a net is therefore entirely captured by its *reachability graph*, which is the graph whose nodes are the markings of the net and whose labelled edges represent firing of transitions leading from one marking to another. In general, such a structure is often referred to as the *state space* of a system. The state space of the example net is illustrated below.

The initial marking is $\odot$, and we have just used $\cdot$ to illustrate the other two reachable markings.

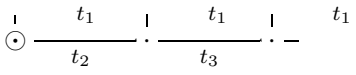



Figure 1.2: State space of the example net.

Finiteness of the state space can be guaranteed by, e.g., structural properties such as conservativeness (every transition "consumes" as many token as it "produces", i.e., the number of tokens remains constant) or by choosing a subclass such as $K$-bounded nets (at most $K$ tokens are allowed on a place).

Petri nets are one of the oldest and most studied formalisms for the investigation of concurrency. Reasoning about the behaviour of a Petri net is typically done by establishing that its state space has certain properties [Rei85]. In the case where we are given a finite state space of a net, there exist algorithms solving—naively or optimally—most of the problems the Petri net community has considered.[1] Even if we are given an arbitrary Petri net, whose state space might be infinite, it is still possible to decide interesting properties about it. E.g., by computing the coverability tree one can decide if the Petri net is bounded (its state space is finite) [KM69]. Also, for a given Petri net $N$ and a marking $M$, it is decidable if $M$ is reachable from the initial marking [May81, Kos82, May84].[2]

The computational complexity of many Petri net problems have been investigated and are well-understood. In their classical paper [JLL77] Jones, Landweber, and Lien studied the complexity of several fundamental problems for Place/Transition nets (called in [JLL77] just Petri nets). Some years later, Howell, Rosier, and others studied the complexity of numerous problems for conflict-free nets, a subclass of Place/Transition nets [HR88, HR89]. For an survey, see [Jan86, EN94].

### 1-safe Nets

In the 1980's, *process algebras* were introduced as an alternative approach to the study of concurrency; they are more compositional and of higher level. The relationship between Petri Nets and process algebras has been thoroughly studied; in particular, many different Petri net semantics of process algebras have been described, see for instance [BDH92, DNM88, Gol88, Old91]. Also, a lot of effort has been devoted to giving nets an algebraic structure by embedding them in the framework of category theory, see among others [Win87, MM90]. Although part of this work has been done for Place/Transition nets [Gol88, MM90], it has been observed that the nets in which a place can contain at most

[1] As an example of open problems, consider the "true concurrency" behavioural equivalences over 1-safe nets in Chap. 5.

[2] This result is probably the most celebrated in the history of Petri nets.

one token, henceforth called *1-safe nets*, have many interesting properties. Places of 1-safe nets no longer model counters but logical conditions; a token in a place means that the corresponding condition holds. This makes 1-safe nets rather different from Place/Transition nets, even though both have similar representations; for instance, finite Place/Transition nets can have infinite state spaces, but finite 1-safe nets cannot.

The advantages of 1-safe nets are numerous, and they have become a significant model. Several semantics can be smoothly defined for 1-safe nets [BF88, NRT90], but are however difficult to extend to Place/Transition nets. Nielsen, Rozenberg and Thiagarajan [Thi87, NRT90] have shown that a model of 1-safe nets, called Elementary Net Systems, has strong categorical connections with many other models of concurrency, such as event structures (another good reference is [WN95]). Finally, 1-safe nets are closer to classical language theory, and can be interpreted as a synchronisation of finite automata.

These properties have motivated the design of verification methods particularly suited for 1-safe nets. Several different proposals have recently been presented in the literature [Val90, God90, McM92, Esp93, WG93]. In order to evaluate them, and as a guide for future research, it is necessary to know the complexity of verification problems for 1-safe nets.

In Chap. 2, we present complexity results concerning the computational complexity of determining whether or not 1-safe nets have certain properties. To be more specific, we investigate the *reachability*, *liveness*, and *deadlock* properties. In fact, we classify these properties with respect to several classes of Place/Transition nets, one of them being 1-safe nets. Since the problems are given in terms of a 1-safe net, obvious decision procedures for these properties can be obtained by first computing the state space of the net, and subsequently applying the algorithms which require a state space as input. However, from a computational complexity point of view, this approach is not optimal, since it implies an exponential blow up. One main conclusion of our work is that for most natural properties of 1-safe nets, the computational complexity of establishing whether or not a net has the desired property is PSPACE-complete.

Our results have enabled us to complete Table 1.1, providing the first systematic study for 1-safe nets.

The contents of Chap. 2 is based on joint work with Javier Esparza and Jens Palsberg, and has been published in the proceedings of FST&TCS 13 [CEP93] and in the journal Theoretical Computer Science [CEP95].

### 1.2.3 Compact Systems

We continue by investigating the more general problem of model-checking branching time and linear time temporal logics over a class of finite state concurrent systems.

| **Petri net class** | **Reachability** | **Liveness** | **Deadlock** |
|---|---|---|---|
| Arbitrary | decidable<br>EXPSPACE-hard | decidable<br>EXPSPACE-hard | *decidable*<br>*EXPSPACE-hard* |
| 1-safe | *PSPACE-complete* | *PSPACE-complete* | *PSPACE-complete* |
| Acyclic | *NP-complete* | *linear time* | *linear time* |
| 1-safe acyclic | NP-complete | constant time | constant time |
| Conflict-free | NP-complete | polynomial time | polynomial time |
| 1-safe conflict-free | polynomial time | polynomial time | polynomial time |
| Free-choice | *decidable*<br>*EXPSPACE-hard* | NP-complete | *NP-complete* |
| 1-safe free-choice | *PSPACE-complete* | polynomial time | *NP-complete* |

Table 1.1: Summary of complexity results for Petri nets.

**Temporal logics and verification.**

When talking about a logic, we are implicitly talking about several components; the logical formulas (usually given by some grammar), a model (a set of possible "worlds" or states, which assign values to basic formulas, such as atomic propositions), and a way to interpret a given formula in a given world (the interpretation tells us whether or not the formula is to be considered "true" or "false" in the given world).

For temporal logics we also have several components: the grammar of the formulas include temporal operators, e.g., the future operator $\diamond$, and the model we consider are Kripke structures. A Kripke structure is a graph whose nodes may be considered as worlds in the above sense. An edge from a world $w_1$ to a world $w_2$ can be thought of as meaning that $w_2$ is a possible (immediate) successor of or future world for $w_1$.

In a classical paper [Pnu77], Pnueli argued that temporal logic could be useful for the specification and verification of non-terminating programs, such as reactive systems, concurrent programs, operating systems, etc. Temporal logic formulas give us the capability to express temporal properties such as "formula $F$ holds sometime in the future of the current world", commonly written as the formula $\diamond F$. In our case, the notion of time is discrete and represented as edges in a Kripke structure. Also, there are two predominant views of how time elapses. Linear time temporal logics consider time as "linear", i.e., at any point in time there is only one possible future; a linear time temporal logic formula is interpreted over a sequence of worlds (a computation path through a Kripke structure). Branching time temporal logics, on the other hand, see time as "branching", i.e., at any point in time there may be several possible futures; a branching time temporal logic formula is interpreted at a world/state of a Kripke structure and the branching structure beginning at that world.

Examples of well-known temporal logics are $\mathsf{L}(X, U, S)$ (linear time) and CTL (branching time). Temporal logics can be classified in many ways. Here we have mainly talked about the view of (discrete) time [Lam80, EH86]. For a more elaborate classification, see [Eme90].

Recall that the finite state systems one wishes to reason about either explicitly or implicitly specify state spaces, typically as (labelled) graphs. Viewing these graphs as Kripke structures, one can use temporal logic formulas to express properties of the state spaces. The problem of deciding if a system $s$ satisfies a property expresses by (encoded in) a formula $\varphi$—the model-checking problem for $s$ and $\varphi$—then reduces to the problem of checking if the formula is satisfied in the systems state space regarded as a Kripke structure. Hence, temporal logics can be seen as specification languages.

The computational complexity of the model-checking problem for both linear and branching time propositional temporal logics has been investigated by many researches. Among the well-known are Sistla and Clarke [SC85] and by Clarke, Emerson, and Sistla [CES86]. Both papers consider (propositional) Kripke structures as models for the logics and the complexity results are stated in terms of the sizes of the Kripke structures and the length of the formulas. The paper [CES86] shows that the model-checking problem for the computational tree logic CTL can be solved in polynomial time, while [SC85] shows, among other things, that the model-checking problem is NP-complete for the linear time temporal logic $\mathsf{L}(F)$ and PSPACE-complete for $\mathsf{L}(X, U, S)$. Recent work shows that complexity bounds from [CES86] can be preserved even if one considers CTL$^2$, a strictly more expressive branching time temporal logic than CTL [BG94].

There exist other well-known classes of systems over which such logics can interpreted. $K$-bounded Petri nets and synchronised automata are examples of such classes of systems. Common to these systems is that they can be viewed as compact representations of Kripke structures; they can specify exponentially large Kripke structures. Verification techniques for these and related systems have been presented in, e.g., [VW86, Lar88, SW89, Val90, WG93, ES92, Esp93, BCM$^+$92, And95]. The work in [Lar88, SW89] focusses on algorithms (tableau systems) for solving the model-checking problem, while the work in [Val90, WG93, ES92, Esp93, BCM$^+$92, And95] is mainly motivated by the "state space explosion" problem and how to overcome it taking time and, especially, space consumption into account. Notions such as "stubborn sets", "persistent sets", "net unfoldings", and "Binary Decision Diagrams" have been proposed to obtain efficient model-checkers in practice.

**Compact systems and model-checking.**

It turns out that the model-checking problems over models like, e.g., synchronised automata and 1-safe Petri nets are very similar. In Chap. 3, we show that such systems can be seen as instances of a general notion we introduce as *compact systems*. Our definition of compact systems is based on Turing machines. Intuitively, compact systems are descriptions of systems whose state spaces are (at most) exponentially larger than the descriptions themselves. The idea is that a class of compact systems is determined by a nondeterministic polynonimal space bounded Turing machine, which interprets its input-strings (input-systems) as system descriptions. The Turing machine has a special "signal" state, which it enters whenever it has computed a state of the input-system. From the Turing machine's computations on the input-system it is then possible to de-

rive a Kripke structure. E.g., the class of $K$-bounded Petri nets can be determined by a Turing machine which given an input $s$, checks if $s$ encodes a $K$-bounded Petri net (in some predetermined format), if so, stores the current marking of the net and tries nondeterministically to simulate a firing transition from the current marking. Whenever a new marking is computed it enters its signal state.

Determining the computational complexity of model-checking is done relative the size of the problem instance. Not surprisingly, the computational complexity of, e.g., model-checking CTL over 1-safe nets is lower—in terms of the size of the problem instance—when the problem instance consists of a formula and the reachability graph of a 1-safe net than when it consists of a formula and the usual description (places, transition, etc.) of a 1-safe net—the reason being that a 1-safe net can encode an exponentially large reachability graph.

Since the finite state concurrent systems being model-checked in practice can very often be viewed as compact systems, it is necessary and, probably more relevant, to evaluate the computational complexity in terms of the size of these "compact" descriptions.

In Chap. 3, we provide *general upper bounds*, which are valid for any class of compact systems. More specifically, we show that the model-checking problems for the logics $\mathsf{L}(X, U, S)$ and CTL and any class of compact systems lie in PSPACE, hereby contributing to the general picture of the computational complexity of model-checking. Our results are summarised in Table 1.2 and have appeared as a technical report [Che95a].[3]

| **Logic** | **Problem Instance** | **Complexity** |
|---|---|---|
| CTL | R-structure (Kripke) and a formula | P |
| $\mathsf{L}(F)$ | R-structure and a formula | NP-complete |
| $\mathsf{L}(X, U, S)$ | R-structure and a formula | PSPACE-complete |
| CTL<br>$\mathsf{L}(F)$<br>$\mathsf{L}(X, U, S)$ | Compact system and a formula | *PSPACE* |

Table 1.2: Complexity in terms of Kripke structures and compact systems.

One use of our results is easily obtainable PSPACE upper bounds. Moreover, for many of the classes of models investigated in the literature the model-checking problems for the temporal logics we consider are PSPACE-hard. By showing how the systems being model-checked can be viewed as compact systems [4] our results provide the matching

[3] The author has become aware that his results on linear time temporal logic indirectly appear in [VW86]. Also, similar results for CTL and CLT$^*$ have recently been obtained using alternating tree automata [BVW94, Kup95]. We have chosen to present our results, partly because we will use them in Chap. 4 and partly because our proofs are quite different.

[4] I.e., show that it is possible to transform the description of the system—in the problem instance used

upper bounds—the intended use of our results.[5] We consider $K$-bounded Petri nets as an example of how our results can be applied.

### 1.2.4 Fair Progress

We continue by investigating the notion of fair progress in the setting of labelled 1-safe nets.

Consider the process agent $\mathit{fix}\,(X = a.X)|(\tau.b.\mathbf{0})$. Its transition graph is given in Figure 1.3. The initial state is $i$, and $s_1$ and $s_2$ are the only other reachable states.

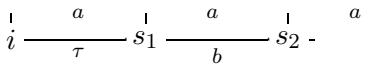



Figure 1.3: Transition system for the process agent $\mathit{fix}\,(X = a.X)|(\tau.b.\mathbf{0})$.

Consider the process agent $(\mathit{fix}(X = a.X + \tau.(\mathit{fix}(Y = a.Y + b.(\mathit{fix}(Z = a.Z))))))$. It's transition graph can also be depicted as Figure 1.3. However, looking at the agents one would expect that there should be a difference with respect to the *degree of concurrency* the two agents exhibited. This example shows that from a model theoretic point of view, concurrency is modelled as *nondeterministic interleaving*. This makes it difficult, and sometimes impossible, to express or reason about certain natural properties of concurrent systems.

#### Partial order semantics and P-CTL.

Let us consider yet another example, this time a labelled 1-safe net.

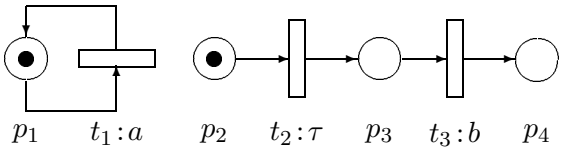



Figure 1.4: Labelled version of the net in Figure 1.1.

The process agent $\mathit{fix}\,(X = a.X)|(\tau.b.\mathbf{0})$ can be represented by this net. The concurrency is obviously visible in this graphical representation and hinted to by the definition of

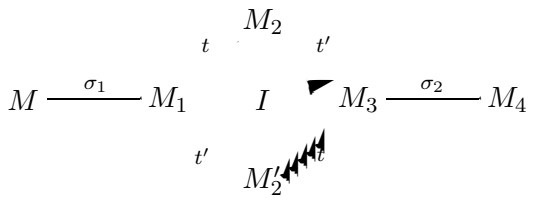
to obtain the lower bound—into the representation as a compact system, using at most a polynomial amount of space. This is a very mild condition.

[5] Although we prove the PSPACE upper bounds by giving algorithms, the algorithms are not intended to be implemented. It is however interesting to notice that several algorithms which are implemented have worst case exponential running times and exponential space consumptions, in terms of $s$ and $\varphi$ [Lar88, SW89, WG93]. In [Kup95], alternating tree automata provide a PSPACE procedure for CTL$^*$ model-checking of concurrent programs (synchronisation of automata).

the effect of firing a transition. We would expect that in any computation of the net, transitions from the left and the right "subnets" would occur if both the subnets were assumed to *progress in a fair way*.

Guided by this observation, let us define a so-called independence relation $I \subseteq T \times T$ over the transitions of the net. The intention is that this relation should capture whether or not two transitions may occur concurrently. We tentatively derive it from the explicit representation of possible concurrency between transitions of the nets—disjointness of neighbourhoods. Hence, $t_1$ would be independent of $t_2$ and $t_3$, written $(t_1, t_2) \in I$ and $(t_1, t_3) \in I$, while $t_2$ and $t_3$ would be dependent, written $(t_2, t_3), \notin I$.

If we take a closer look at the structure of the net's state space, we discover that if we have $M \xrightarrow{\sigma_1} M_1 \xrightarrow{t} M_2 \xrightarrow{t'} M_3 \xrightarrow{\sigma_2} M_4$, where $(t, t') \in I$ and $\sigma_1, \sigma_2$ are themselves firing sequences, then there must necessarily exist a marking $M_2'$ such that $M \xrightarrow{\sigma_1} M_1 \xrightarrow{t'} M_2' \xrightarrow{t} M_3 \xrightarrow{\sigma_2} M_4$. This is best illustrated by the "independence diamond" below.




Exploiting $I$ we may consider the two sequences $\sigma_1 t t' \sigma_2$ and $\sigma_1 t' t \sigma_2$ equivalent, since they only differ with respect to permutation of adjacent independent transitions. Hence, the relation $I$ induces an equivalence relation on firing sequences [6], whose equivalence classes are commonly referred to as (Mazurkiewicz) *traces* [Mar77]. Moreover, a natural partial order can be defined over the traces [Maz86]. This partial order represents a more concrete view of the computations of concurrent systems in which events are ordered partially, rather than linearly, reflecting both their causal dependencies and independencies. Such a partially ordered structure is often referred to as a *partial order semantics* of a concurrent system.

Below in Figure 1.5, we have "unfolded" the state space of the net from Figure 1.4. The chosen labelling indicate how the unfolded structure—which, by the way, is isomorphic to the aforementioned partial order structure over the traces of the net—is obtained. Intuitively, the only progress fair computations are the firing sequences that eventually reach the "lower path"—$t_2$ and $t_3$, which are independent of $t_1$, must eventually occur. In Chap. 4, we show how a generalisation of Mazurkiewicz's traces to infinite sequences captures this observation formally.

The foundation of traces was presented by Mazurkiewicz in [Mar77, Maz86]. His idea was to equip an alphabet $\Sigma$ with a *symmetric irreflexive independence relation*, which induced an equivalence relation over the monoid $\Sigma^*$ of finite string over $\Sigma$. In [Maz86], Mazurkiewicz applied his theory to Petri net as sketched in the above example, where the set of transitions corresponded to an alphabet and $I$ corresponded to an independence relation over the alphabet.

[6] as well as the monoid $T^*$

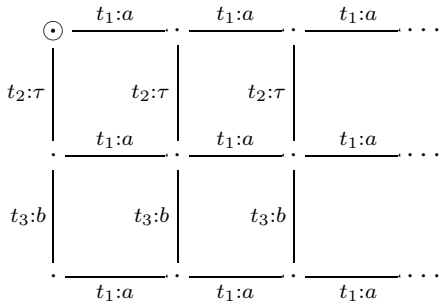



Figure 1.5: Unfolded state space.

Petri's introduction of nets focussed on the explicit representation of concurrency in the topology of a net and in the firing rule. Mazurkiewicz's work [Maz86] focussed on the model theoretic view that concurrency should be represented *explicitly* by imposing more structure on the underlying semantical model, the state space. The study of partial order semantics/"true concurrency" has developed numerous new models, e.g., concurrent and asynchronous transition system [Shi85, Bed88, Sta89, Old91] and event structures [Win80, NPW81, Win86]. For an overview of the relation between many of the existing models, see [WN95]. Common to these models is that they represent concurrency explicitly by either an independence relation (asynchronous transition system) or a conflict relation (event structures). Also, Mukund and Nielsen [MN92] have shown how it is possible to obtain elementary labelled asynchronous transition systems from process agents, like the above, by introducing locations in the structural operational semantics rules for CCS.

In the context of model-checking, partial order semantics have several advantages. The so-called "state space explosion" problem has motivated researches to use partial order semantics. It has been observed that an exhaustive state space exploration can often be avoided; e.g., if a sequence (element) of a trace leads to a deadlocked state, then all sequences in that trace must necessarily lead to that deadlock. Hence, it is sufficient only to explore one sequence in that trace. This can lead to significantly improved running times and space consumptions as observed, among others, by Valmari [Val88, Val90] and by Godefroid and Wolper [God90, GW91, WG93, GW94]. Another motivation to investigate partial order semantics has been the possibility to interpret temporal logics over traces taking causality and concurrency into account, see, e.g., [PP90, Pen93, Thi94, APP95, PK95].

**Model-checking P-CTL.**

In Chap. 4, we investigate the notion of *fair progress* for labelled 1-safe nets, motivated by the above example. Our main objective is to explore the use of the extra structure of independence in the context of specification logics. Based on an independence relation

on transitions (given by disjointness of neighbourhoods) and a generalisation of traces, which takes infinite firing sequences into account, we define a partial order semantics for the labelled 1-safe nets. This semantics captures—in a formal sense—the notion of fair progress among independent event; we can then formally define which firing sequences are progress fair. We then introduce and study a CTL-like branching time temporal logic, P-CTL, which contains one important feature: the model-theoretic incorporation of progress. P-CTL-formulas are interpreted relative to the progress fair computations, rather than all computations, as is the case for the standard interpretation of CTL.

As an example, the formula $\mathsf{Ev}(<b>tt)$—to be read as"eventually a $b$-labelled transition/action is enabled"—is true of the process agent example under the assumption of progress (our interpretation), but not without (standard CTL interpretation). Our interpretation is conservative in the sense that P-CTL interpreted over standard labelled transition systems coincides with the standard CTL interpretation. In process algebraic terms, our notion of fair progress—progress of independent events—intuitively corresponds to a progress fair "parallel operator".

When handling progress fairness in the setting of partial order semantics, we are able to avoid the obstacle of encoding certain fairness assumptions in the logic or into the model-checking algorithm [MP92, CES86], and treat progress fairness assumptions uniformly by using Mazurkiewicz trace-theory.

In the standard setting of Kripke structures, model-checking of CTL-like logics has been described in [CES86] using a state based algorithm and in [Lar88, SW89] using tableaux rules.

We give both a tableau based method and a state labelled base method for model-checking P-CTL. These methods are both based on state space exploration. However, they differ in the way the exploration is performed. Tableau based methods are usually referred to as "local model-checking"; the way one establishes that a state satisfies a given formula is from the given state to explore the state space according the tableau rules. These rules typically infer the properties of a state in terms of the properties of its neighbouring states. State labelling methods, on the other hand, explore the entire state space, labelling the states in a bottom-up fashion with the subformulas (of a given formula) they satisfy.

Our methods are conservative extensions of the existing standard methods in the sense that our methods are equivalent if the systems we consider may not exhibit concurrent behaviour. Based on the results from the previous two chapters, we determine the computational complexity of model-checking our new logic. Our results show that there is now significant penalty, when going from CTL to P-CTL.

Although we choose a partial order semantics for the nets, the syntax of our logic does not allow us to express concurrent behaviour explicitly. We therefore also investigate extensions of our logic with modal operators expressing concurrent or conflicting behaviour. It turns out that variations of the satisfiability problem, i.e., the problem of deciding whether or not there exists a model in which a given formula is satisfiable, becomes undecidable.

The contents of Chap. 4 been published in the proceedings of AMAST'95 [Che95b] as an extended abstract of the technical report [Che95c].

## 1.3 Contents of Part II

Part II presents contributions in the field of behavioural reasoning about concurrent systems based on the notions of behavioural preorders and behavioural equivalences.

### 1.3.1 Behavioural Equivalences

Assume we are given some relation $R$ between models of the concurrent systems we wish to reason about. $R$ could for example be a relation between (rooted) labelled transition systems. Let us further assume that $s_1$ and $s_2$ are two system descriptions and that $\langle s_1 \rangle$ and $\langle s_2 \rangle$ are labelled transition systems given by some semantical mapping $\langle\, \rangle$. $R$ then induces a relation on the system descriptions defined by $s_1 \prec_R s_2$ if and only if $\langle s_1 \rangle R \langle s_2 \rangle$. Assume it is known that $\langle s_1 \rangle$ possesses some (behavioural) property P and that the relation $R$ "preserves" P in the sense that if $t_1$ and $t_2$ are two transition systems such that $t_1 \, R \, t_2$ and $t_1$ possesses the property P, then $t_2$ must also possess the property P. By establishing $s_1 \prec_R s_2$ one then implicitly establishes that $\langle s_2 \rangle$ possesses the property P.

A well-know example of this setting is Milner's CCS (Calculus of Communication Systems) [Mil80, Mil89]. In this case, the concurrent systems are described as process agents, terms of a process algebra. A semantical mapping assigning a transition graph to each process agent is induced by a set of inference rules, commonly referred to as CCS's structural operational semantics (SOS), in which the behaviour of a composite process term is given by the behaviour of its components.[7] The nodes of the graphs are process terms and the labelled edges are generated by the inference rules. The transition graph of a term $t$ can be viewed as a labelled transition system rooted at $t$. Milner investigates several behavioural relations of which *strong bisimulation*, denoted $\sim$, and *weak bisimulation*, denoted $\approx$, are the most well-known; both are equivalence relations. We only sketch the intuition behind strong bisimulation. Two nodes $n_1$ and $n_2$ of any transition graphs are strongly bisimilar if they can continuously simulate each others transitions in the following sense: (1) $n_2$ can simulate $n_1$, i.e., if $n_1$ has a labelled transition leading to a state $n_1'$, then $n_2$ must have a similarly labelled transition leading to a state $n_2'$, such that $n_1'$ and $n_2'$ are also strongly bisimilar, and (2) $n_1$ can simulate $n_2$. Weak bisimulation is—as the name suggests—a coarser, i.e., less distinguishing, relation, because the individual transitions are no longer required to be simulated by corresponding individual transitions, but may, e.g., be simulated by several additional so-called "invisible transitions".

[7] The inference rules are defined in the structure of the process terms, and allows one to derived, step by step, the operational behaviour of a term.

**Behavioural equivalences and verification.**

In [Mil89], Milner shows that it is possible to reason about the behaviour of process agents using behavioural equivalences. In fact, his approach corresponds to the one sketched in the above paragraph: assume a given process agent $s$ is viewed as an implementation of some system. Reasoning about $s$'s behaviour can then be done by first choosing another process agent $s_P$ which possesses a desired property $P$. This process agent can be viewed as a specification of (part of) the desired behaviour of $s$. Since $s_P$ might only express some, but not all, desired behaviours of $s$, one typically modifies $s$ slightly by "ignoring" certain actions. E.g., one might only be interested in whether or not a subset of $s$'s possible actions occur in some fixed sequence or pattern. Hence, one has to abstract away from $s$'s other actions—typically by making them "invisible"—before one tries to establish that $s_P \approx s$.[8]

It turns out that there exist serval temporal/modal logics which *characterise* these bisimulations. E.g., two process agents are strongly bisimilar if and only if they—or rather the corresponding nodes in their transition graphs—satisfy the same set of Hennessy-Milner logic formulas [Mil89]. If we assume that the behavioural equivalence $R$ is characterised by the logic $L$, then the task of establishing whether or not $s_P$ is behaviourly related to the modified version of $s$ corresponds to checking whether or not their state spaces satisfy the same set of formulas. A related issue is discussed in [Pnu85], where the notion of a logic $\mathcal{L}$ being *expressive* for a process language $\mathsf{Proc}$ (equiped with a notion of behavioural equivalence) is defined. Intuitively, for such a logic, there exists a characterisitc formula $\mathcal{L}(p)$ for every process $p$, which essentially characterises the process' behavioural equivalence class.

### 1.3.2 Open Maps and Abstract Behavioural Equivalences

Since bisimulation's exposure in [Mil80, Par81], the research community has invented and investigated an impressive number of bisimulation-like behavioural equivalences, e.g., [GG89, vG90, LS91, MS92], confirming that the notion of bisimulation is an important general concept. However, much of this work is very *concrete* in the sense that, typically, a specific class of models is chosen with respect to which a bisimulation-like relation is investigated.

The amount of research done in the area of process algebras and fields related to the semantics of concurrent systems has lead some researches to look for a unifying theory, e.g., Control Structures [MMP95], Linear Logic and the Geometry of Interaction [Gir87a, Gir87b, AJ92], and Joyal, Nielsen, and Winskel's Presheaves [JNW93]. As a response to some of the numerous models for concurrency proposed in the literature, Winskel and Nielsen have used category theory as an attempt to understand the relationship between models like event structures, Petri nets, trace languages, and asynchronous transition

[8] Using $\sim$ for $R$ is typically too strong a requirement. The reason is that $s_P$ is usually relatively simple compared to $s$. Hence, when $s$ is modified before trying to establish its behavioural relationship to $s_P$, several actions from $s$ will be made "invisible" and actions from $s_P$ may only be meaningfully simulated by several—possibly invisible—actions from the modified version of $s$.

systems [WN95]. They show, among other things, how most common process algebraic operators can be presented as category-theoretic concepts, e.g., universal constructions such as product and co-product, and present the relationship between different categories of models such as Petri nets, event structures, and transition systems with independence, by exhibiting adjunctions between these categories.

However, a similar category-theoretic way of adjoining abstract behavioural equivalences to a category of models had been missing until Joyal, Nielsen, and Winskel proposed the notion of *spans of open maps* [JNW93] as an *abstract* definition of the notion of bisimulation. Joyal, Nielsen, and Winskel showed how the theory of open maps can capture Milner's strong bisimulation and how it can be applied to identify a new bisimulation, strong history-preserving bisimulation, on models with independence like event structures and Petri nets.

Let us sketch their idea. A category of models of computations $\mathcal{M}$ is chosen, then a subcategory of observations $\mathcal{P}$ is chosen relative to which open maps are defined. Two models are $\mathcal{P}$-bisimilar if there exists a span of open maps between them.

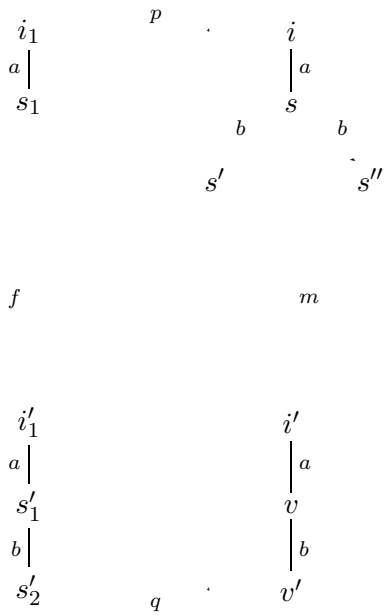
If $X$ and $Y$ are two models in the category $\mathcal{M}$, and $m : X \longrightarrow Y$ is a morphism between them, then we can think of $m$ as a map that specifies how to observe the behaviour of $X$ in $Y$. For $m$ to be open, we intuitively require that $Y$'s behaviour can be simulated by $X$'s. The idea is perhaps best explained by a small example.

Consider $\mathcal{M}$ as the category of labelled transition systems. The objects are of the form $(S, i, Act, \longrightarrow)$, where S is the set of states, $i \in S$ is the initial state, and $\longrightarrow \subseteq S \times Act \times S$ is the transition relation. A morphism $f : T_1 \longrightarrow T_2$ is a map $\sigma_f$ between the set of states of the objects, such that $T_1$'s initial state is mapped to $T_2$'s, i.e., $\sigma_f(i_1) = i_2$, and transitions from $T_1$ are preserved, i.e., if $s \xrightarrow{a} s'$ is a transition in $T_1$, then $\sigma_f(s) \xrightarrow{a} \sigma_f(s')$ must exist a transition in $T_2$. Notice how $T_1$ in a very direct way is simulated by $T_2$: if state $s$ in $T_1$ is mapped to state $v$ in $T_2$, and $s \xrightarrow{a} s'$, then we know that there must be an $a$-labelled transition from $v$ leading to a state $v'$, such that $\sigma_f(s') = v'$. The reader might already be able to see some resemblance to the definition of Milner's strong bisimulation. Intuitively, we need to show that if $v = \sigma_f(s)$ and $v \xrightarrow{a} v''$, then there must exist an $a$-labelled transition from $s$ leading to a state $s''$, such that $\sigma_f(s'') = v''$.

Now suppose we choose $\mathcal{P}$ as the subcategory in $\mathcal{M}$ induced by the objects of the form

$$i \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n \ ,$$

where all states are distinct. Let us refer to such objects as words. Consider the diagram below. To the left, we have two words (observations), and to the right we have two labelled transition systems (models). The morphisms $f$, $p$, $q$, and $m$ are uniquely determined in this case, hence we refrain from giving their formal definition.




We have used solid arrows to indicate labelled transitions within the objects, and dashed arrows to indicate morphisms between objects. The diagram has the property that it is commuting, i.e., if we compose the morphisms $p$ and $m$, denoted $m \circ p$, and compose the morphisms $f$ and $q$, denoted $q \circ f$, then $m \circ p = q \circ f$. Let us denote the objects in the diagram as (morphisms are now solid arrows)

$$\begin{array}{ccc} O_1 & \xrightarrow{p} & T_1 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle m} \\ O_2 & \xrightarrow[q]{} & T_2 \end{array} \tag{1.1}$$

Let us for the moment ignore the two $b$-labelled transitions from $s$ in $T_1$. Since $m$ maps $s$ to $v$, we know that transitions from $s$ can be matched by transitions from $v$. How could we conclude the converse? Assume we require that for commuting diagrams as the above, there has to exist a morphism $h : O_2 \longrightarrow T_1$ such that the diagram

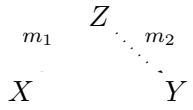
$$\begin{array}{ccc} O_1 & \xrightarrow{p} & T_1 \\ {\scriptstyle f}\downarrow & \swarrow{\scriptstyle h} & \downarrow{\scriptstyle m} \\ O_2 & \xrightarrow[q]{} & T_2 \end{array} \tag{1.2}$$

commutes, i.e., $p = h \circ f$ and $q = m \circ h$. The equation $p = h \circ f$ ensures that $h$ maps $s'_1$ to $s$. The existence of $h$ guarantees that there must exist a $b$-labelled transition $s \xrightarrow{b} r$ in $T_1$, for some $r$, since $h$ maps $s'_1$ to $s$ and $s'_1$ has an out-going $b$-labelled transition. The equation $q = m \circ h$ then guarantees that this transition in $T_1$ is mapped to the transition

$v \xrightarrow{b} v'$ in $T_2$, i.e., the transition $\sigma_m(s) \xrightarrow{b} v'$ in $T_2$ can be simulated by a transition $s \xrightarrow{b} r$ in $T_1$, such that $\sigma_m(r) = v'$. (Notice that $q$ maps $s'_1$ to $v$, and that the $b$-labelled transition $s'_1 \xrightarrow{b} s'_2$ in $O_2$ is mapped, via $q$, to the transition $v \xrightarrow{b} v'$ in $T_2$.)

The morphism $m$ is said to be $\mathcal{P}$-open if when quantifying over all possible commuting diagrams of the form (1.1), one can always find a mediating morphism such that the corresponding diagram of the form (1.2) commutes.

In general, two models $X$ and $Y$ in $\mathcal{M}$ are said to be $\mathcal{P}$-bisimilar if there exists a span of open maps between them, as illustrated below.




In our example, $\mathcal{P}$-bisimilarity corresponds to Milner's strong bisimulation [JNW93].

In Chap. 5, we show, as a measure of the *applicability* of open maps as an abstract definition of bisimulation, that it is possible to capture not only Milner's strong bisimulation but a representative selection of *well-known* bisimulations. We are also able to capture general, non-bisimulation-like behavioural equivalences such as (pomset) trace equivalence. Furthermore, the theory of open maps focusses on different parameters such as $\mathcal{M}$ and $\mathcal{P}$. We discuss how this view point helps identifying new or different aspects and problems related to models for concurrency.

Our results indicate that Joyal, Nielsen, and Winskel's proposed theory of open maps is reasonably general.[9]

Large parts of the contents of Chap. 5 are based on joint work Mogens Nielsen and has been published in the proceedings of FST&TCS 15 [NC95], as well as appeared as a technical report [CN95].

### 1.3.3 Congruence Properties of Behavioural Equivalences.

Another aspect of behavioural equivalences which has received attention is the issue of substitutivity, or *congruence properties*. The notion of substitutivity is perhaps best explained by an example. Let us consider Milner's strong bisimulation. Recall that process agents are terms in a process algebra. This algebra has operators such as $|$, parallel composition, and $+$, nondeterministic choice. Let $t$ denote any process agent and assume we know that agents $t_1$ and $t_2$ are strongly bisimilar. One can prove that $t \,|\, t_1$ and $t \,|\, t_2$ must then also be strongly bisimilar. In fact, we can show that if $t_1 \sim t_2$ and $t'_1 \sim t'_2$, then $t_1 \,|\, t'_1 \;\sim\; t_2 \,|\, t'_2$. This shows that strong bisimulation is a congruence [10] with respect to—or preserved by, to put it otherwise—CCS's parallel composition. It

[9] There are equivalences we haven't been able to capture, such as branching bisimulation [vG90] and history preserving bisimulation [GG89]. In Sect. 5.4 we discuss history preserving bisimulation.

[10] In general, an equivalence relation $R$ is a congruence with respect to an $n$-ary operator $Op(\,.\,,\ldots,\,.\,)$ of the algebra, if it is the case that whenever $a_1\, R\, a'_1$, …, $a_n\, R\, a'_n$, then $Op(a_1,\ldots,a_n)\, R\, Op(a'_1,\ldots,a'_n)$.

turns out, as shown by Milner in [Mil89], that strong bisimulation is a congruence with respect to all the operators of CCS.

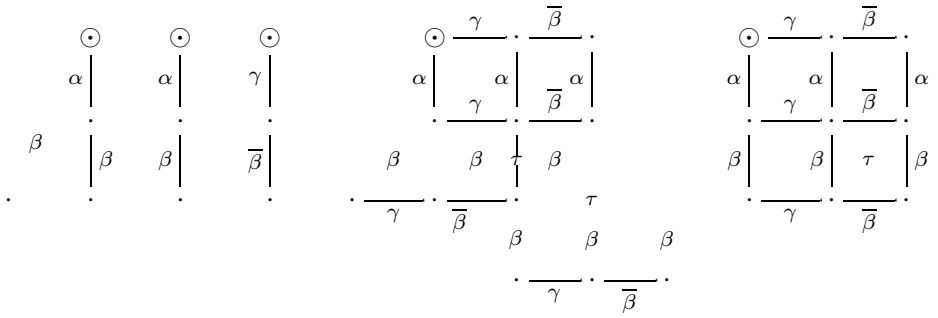
The relevance and importance of congruence properties in the context of verification based on behavioural equivalences, as sketched above, is exemplified by observations as following. If a behavioural equivalence is a congruence with respect to some set of operators, then two behaviourally equivalent systems build up using these operators will remain to be so under substitutions of subcomponents (or subsystems), if the subcomponents which are substituted for each-other are behaviourally equivalent. For example, if a systems has been verified along the lines sketched above, one can *freely re-implement or replace a subcomponent* $t_1$ as, say, $t_2$, as long as they are behaviourally equivalent.

Returning to Winskel and Nielsen's presentation of process algebraic operators in terms of category-theoretic concepts such as products and co-products [WN95], a natural question to ask—and which we address in Chap. 6—is whether or not it is also possible to capture the following important aspect of process algebraic operators and bisimulation equivalences: when is $\mathcal{P}$-bisimilarity a congruence with respect to some of these operators?

### $\mathcal{P}$-factorisability.

Based on the view that endofunctors on $\mathcal{M}$ may be seen as abstract operators we define a natural and general notion of a functor being *$\mathcal{P}$-factorisable*. We then show that a $\mathcal{P}$-factorisable functor must preserve $\mathcal{P}$-bisimilarity. We observe an apparent similarity with the idea behind Milner's proofs that CCS operators preserve strong bisimulation. Below we give a example, illustrating the intuition behind $\mathcal{P}$-factorisability.

Consider $\mathcal{M}$ and $\mathcal{P}$ from the example in the previous section and transition systems below, which we denote—left to right—$T_1, \ldots, T_5$.



$T_1$ is strongly bisimilar ($\mathcal{P}$-bisimilar)—in the sense of Milner [Mil89]—to $T_2$. In fact, there is an obvious open map $k$ from $T_1$ to $T_2$. Considering $T_3$ to be fixed, we can define a functor $\_|T_3 : \mathcal{M} \longrightarrow \mathcal{M}$, where $|$ acts as a CCS-like parallel composition. $T_4 = T_1|T_3$ and $T_5 = T_2|T_3$ serve as an informal illustration of $\_|T_3$, when applied to $T_1$ and $T_2$, respectively.

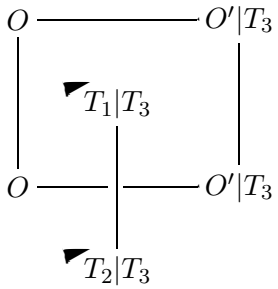
Recall that $\mathcal{P}$-bisimilarity is based on open maps, which again are based on observations from $\mathcal{P}$. E.g., we can observe $O$, the behaviour $\odot \xrightarrow{\alpha} \cdot \xrightarrow{\gamma} \cdot$, in $T_4$ and—via

$k|T_3 : T_4 \longrightarrow T_5$—in $T_5$. Some of these transitions in $T_4$ are due to transitions "from" $T_1$, here only the $\alpha$-transition.

In much the same way as Milner [Mil89] shows that $P \sim P'$ implies $P \,|\, Q \sim P' \,|\, Q$, we would like to conclude that if $k : T_1 \longrightarrow T_2$ is open, then so is $T_1|T_3 \xrightarrow{k|T_3} T_2|T_3$.[11]

Using $k$, we conclude that the $\alpha$-transition in $O$ must also be observable in $T_2$. In fact, we have a commuting diagram as in (6.1) with $X = T_4$, $Y = T_5$, $O_1 = O_2 = O$, $m = k|T_3$, and $f = 1_O$ (the identity morphism), and by the above we have extracted a second commuting diagram of the form (6.1) with $X = T_1$, $Y = T_2$, $O_1 = O_2 = O' = \odot \xrightarrow{\alpha} \cdot$, and $m = k$.

In fact, the way we have "factored" $O$ in to $O'$ is consistent with $\_|T_3$ in the following sense: there exists a commuting diagram of the form




In Chap. 6, we formalise this as $\mathcal{P}$-factorisability, and, as a consequence, we will be able to conclude that $k|T_3$ is an open map.

Winskel and Cattani are developing presheaves over categories of observations as models for concurrency [CW96]. For presheaves there are general results on open maps, including the axioms for open maps of Joyal and Moerdijk [JM94], which make light work of showing the bisimulation of presheaves is a congruence for CCS-like languages. A condition superficially like $\mathcal{P}$-factorisability is important in transferring such congruence properties from presheaves to other models like transition systems and event structures.

**Meta-theorems.**

Common to much work on behavioural equivalences being congruences is that one chooses a specific (a) process term language, (b) class of models, and (c) behavioural equivalence. One then shows that specific operators—such as "parallel composition" and "nondeterministic choice"—preserve the proposed behavioural equivalence. Well-known examples are [Hen88, Mil89]. The behaviour of their process algebras is given by a structural operational semantics (SOS) [Plo81], in which the behaviour of a composite process term is given by the behaviour of its components.

[11] In fact, just as Milner uses a bisimulation $P \sim P'$ to exhibit a bisimulation $P \,|\, Q \sim P' \,|\, Q$, we will "factor" the observation $\odot \xrightarrow{\alpha} \cdot \xrightarrow{\gamma} \cdot$ into transitions from $T_3$ and from $T_1$ and $T_2$, respectively. This will guide us to the mediating morphism required in (6.2).

In general, the term languages resemble each other, usually CCS-like, and hence the results differ from each other primarily with respect to the proposed equivalences. Based on this observation, one might look for general results.

One approach could be not to look at specific operators, but try to reason about a general set of operators. In [BIM88], Bloom, Istrail, and Meyer study a meta-theory for process algebras which are defined by SOS rule systems. They identify a rule format which ensures that any process language in so-called GSOS format has strong bisimulation as a congruence. It is worth noticing that they fix the notion of behavioural equivalence, strong bisimulation, and obtain general results by allowing the operators in the language to vary.

Based on the notion of $\mathcal{P}$-factorisability, we choose an approach "orthogonal" to that of [BIM88]. The presentation of $\mathcal{P}$-factorisability focusses, especially, on certain closure properties of the category $\mathcal{P}$. Based on this observation, we show how one can *parametrise* the proofs of functors being $\mathcal{P}$-factorisable with respect to the choice of the observation category $\mathcal{P}$, i.e., the choice of a behavioural equivalence. Intuitively, we fix the operators, but allow the behavioural equivalence to vary. Then we identify conditions on $\mathcal{P}$ which ensure that the varying equivalences are congruences with respect to the operators. Hence, our results can be seen as "orthogonal" to that of Bloom, Istrail, and Meyer, in that we can parametrise with respect to the behavioural equivalences, as opposed to operators, [BIM88].

Inspired by our work on weak bisimulation in Chap. 5 we propose a category which could be subject to investigations similar to the above. This category generalises the category of labelled transition systems by splitting the labelling set into two disjoint set—visible and invisible labels.

Large parts of the contents of Chap. 6 are based on joint work Mogens Nielsen and will appear in the proceedings of CAAP'96, and have appeared as a technical report [CN96].

## 1.4 Contents of Part III

Part III presents contributions in the field of set constraints.

### 1.4.1 Set Constraints

Set constraints are inclusions of the form $X \subseteq Y$ between set expressions ($X$ and $Y$) denoting sets of (ground) terms of a free algebra over a finitely ranked alphabet. E.g., 0 denote the empty set, 1 the set of all terms, $c(X, Y)$ denote the set of terms of the form $c(t_X, t_Y)$, where $t_X$ and $t_Y$ are terms in the sets denoted be $X$ and $Y$, respectively.

Set constraints have been used in *program analysis* for functional programming languages, imperative programming languages, and logic programming languages They are typically derived from the syntax of a program and solutions to them can yield useful information for, e.g., type inference, implementations, and optimisations. Consequently, much work has been devoted to (implementing) program analysis based on set

constraints [Rey69, JM79, Mis84, MR85, YO88, HJ90b, AM91a, AM91b, HJ92, Hei92, AW93, AWL94, Hei94, AL94]. Also, set constraints have recently been used to define a constraint logic programming language over sets of ground terms that generalises ordinary logic programming over an Herbrand domain [Koz94].

From the theoretical side, there has also been much work in understanding the complexity of the satisfiability problem for different classes of set constraints [HJ90a, GTT93a, AW92, GTT93b, Ste94, AKVW93, AKW95, BGW93, CP94a, CP94b].

### 1.4.2 A Gentzen-style Axiomatisation for Set Constraints

An axiomatisation of the main properties of set constraints was proposed in [Koz93]. General models of these axioms are called *termset algebras*. In [Koz93], a representation theorem was proved showing that every termset algebra is isomorphic to a set-theoretic termset algebra. These models include the standard models in which set expressions are interpreted as sets of ground terms, as well as nonstandard models in which set expressions are interpreted as sets of states of *term automata* [KPS92].

In Chap. 7 we continue the theoretical investigation of set constraints. We propose a *Gentzen-style axiomatisation* involving sequents of the form $\Phi \vdash \Psi$, where $\Phi$ and $\Psi$ are finite sets of set constraints. The intended interpretation of the sequent $\Phi \vdash \Psi$ is that if all the constraints in $\Phi$ hold of some model, then at least one of the constraints $\Psi$ holds in that model.

This axiomatisation can be thought of as a deductive system for refuting unsatisfiable systems of *mixed* positive and negative constraints. Deriving the sequent $\Phi \vdash \Psi$ is tantamount to refuting the mixed system $\Phi \cup \{s \neq t \mid s = t \in \Psi\}$. Systems of the restricted form $\Phi \vdash \bot$ correspond to systems of positive set constraints alone.

For this deductive system, we prove

(i) completeness over standard models for satisfiability of positive set constraints alone (if $\Phi$ is unsatisfiable, then $\Phi$ is refutable, *i.e.*, $\Phi \vdash \bot$ is derivable);

(ii) incompleteness over standard models for satisfiability of mixed positive and negative constraints (*i.e.*, not all valid sequents $\Phi \vdash \Psi$ are derivable);

(iii) completeness over nonstandard models (all set-theoretic termset algebras) for satisfiability of mixed positive and negative constraints (*i.e.*, all valid sequents $\Phi \vdash \Psi$ are derivable) .

Our axiomatisation is based on the close connection between set constraints and hypergraphs [AKW95]. Hypergraphs are a generalisation of finite automata, where the usual binary edges are replaced by (hyperedge) relations of arbitrary arity. For systems consisting of positive constraints only, it is possible to construct a hypergraph, whose set of runs (mappings of ground terms to states of the hypergraph, consistent with the hyperedge relation) corresponds to the constraint system's set of solutions. When negative constraints are involved, the situation becomes technically more challenging [CP94b, GTT93b, AKW95, Ste94].

Although several interesting results involving the decidability and complexity of set constraints extended with negative constraints ($X \not\subseteq Y$) have appeared [CP94b, GTT93b, AKW95, Ste94], the distinction between the two cases is still far from clear from a deductive standpoint.

Our results shed light on the distinction between exclusively positive and mixed positive and negative constraints.

We also give an example of how verification of finite state concurrent systems can be performed using set constraints. More specifically, we show how Milner's protocol can be verified for the absence of deadlocks by encoding it in `clp(sc)`, a logic programming language over set constraints introduced by Kozen [Koz94].

The contents of Chap. 7 on the axiomatisation is joint work with Dexter Kozen and has been published in the proceedings of ICALP '96 [CK96] as weel as a technical report [CK95].

### 1.4.3 Rational Spaces

Set constraints exhibit a rich mathematical structure and are closely related to type theory, automata theory, first-order monadic logic, Boolean algebras with operators, and modal logic [JT51, JT52, GTT93a, BGW93, GTT93b, AKW95, KPS93, KPS94, CP94a, Koz93, Koz95]

It has been noticed that many results in the literature on set constraints have a topological flavour. Recently in [Koz95], Kozen defines *rational spaces* as a family of topological spaces with a *regular structure*, develops the basic theory, and shows how many results in the literature could be re-derived by general topological principles. By endowing a hypergraph with a topology on its set of states, $D$, and requiring that certain sets of hyperedges are closed in the derived product topology, the set of runs over the hypergraph can be given a topology, yielding a rational space. A category of rational spaces is obtained by defining morphisms as rational maps; these are continuous maps preserving the rational structure. Certain singleton rational subspaces are defined as rational points and shown to play an important role.

Kozen also presents a connection to set constraints by giving a complete characterisation of the sets of solutions to systems of set constraints in terms of rational spaces. He gives a one-to-one correspondence, up to logical equivalence on one side and so-called rational equivalence preserving $X$ on the other, between (finite) systems of set constraints over variables $X$ and certain (finitary) subspaces of a certain rational space. The given correspondence preserves the partial order of logical entailment between systems of set constraints over $X$ and so-called $X$-preserving rational embeddings (injective rational maps) between the corresponding subspaces.

These results strongly suggest that further study of rational spaces is needed. In Chapter 8, we continue the preliminary investigations of rational spaces.

We give a Myhill-Nerode-like characterisation of rational points and, based on this characterisation, we give a simple and direct proof that the rational points of a finitary rational space are dense [Koz95]. As noted in [Koz95], we conclude that any non-empty

finitary rational space has a rational point. This translates to finite systems of (positive) set constraints as follows: if a systems of (positive) set constraints has a solution, then it has a regular solution. We show that the rational points in finitary rational spaces in some sense exactly capture the topological structure of the space. We also investigate congruences in $\Sigma$-hypergraphs and their interplay with the Myhill-Nerode characterisation. Congruences in rational spaces are strongly related to the notion of bisimulation [Mil89] in models of concurrency and a similar notion has appeared in [Koz94] in the context of efficient constraint solving. Finally, we determine the computational complexity of some decision problems related to rational embeddings.

The contents of Chap. 8 is joint work with Dexter Kozen.

# Part I

# On Model-Checking

Chapter 2

# Complextity Results for 1-safe Nets

**Contents**

| **Petri net class** | **Reachability** | **Liveness** | **Deadlock** |
|---|---|---|---|
| Arbitrary | decidable<br>EXPSPACE-hard | decidable<br>EXPSPACE-hard | decidable<br>EXPSPACE-hard |
| 1-safe | PSPACE-complete | PSPACE-complete | PSPACE-complete |
| Acyclic | NP-complete | linear time | linear time |
| 1-safe acyclic | NP-complete | constant time | constant time |
| Conflict-free | NP-complete | polynomial time | polynomial time |
| 1-safe conflict-free | polynomial time | polynomial time | polynomial time |
| Free-choice | decidable<br>EXPSPACE-hard | NP-complete | NP-complete |
| 1-safe free-choice | PSPACE-complete | polynomial time | NP-complete |

Table 2.1: Summary of complexity results for Petri nets.

## 2.1 Introduction

In the following sections, we study the maybe three most important verification problems for Petri nets: *reachability*, *liveness*, and existence of *deadlocks*. We determine their complexity for 1-safe nets, and for three important subclasses: *acyclic*, *conflict-free* and *free-choice* nets. In all cases, we compare the results with the complexity of the corresponding problems for Place/Transition nets.

The presentation is a mixture of survey and new results. Our new results have enabled us to complete Table 2.1. Throughout, we attribute previously known results to their authors.

Two interesting subclasses of Petri nets are not covered by Table 2.1, namely S- and T-systems [BT87]. For those, reachability, liveness, and deadlock are known to be polynomial in the Place/Transition case [BT87, CHEP71, GL73], hence also in the 1-safe case. Related work concerning not the complexity of particular verification problems but the complexity of deciding different equivalence notions can be found in [JM93].

Our results are organised as follows. Section 2.2 contains basic definitions. In Section 2.3 we show that the deadlock problem is recursively equivalent to the liveness and reachability problems. Section 2.4 shows that the three problems are PSPACE-complete in the 1-safe case. In section 2.5, the different classes of Petri nets mentioned above are considered. Finally, in Section 2.6 other problems are studied.

*Remark.* We have defined 1-safe nets as a subclass of Place/Transition nets. Other versions of 1-safe nets can be found in the literature, namely the Condition/Event systems [Rei85] and the Elementary Net Systems [Thi87]. This multiplicity of definitions is maybe annoying but harmless: the differences among them are small, and of rather technical nature (see [BC92] for a discussion). In particular, our results are independent of the definition used.

## 2.2 Definitions

We recall in this section some basic concepts about Place/Transition nets and 1-safe nets, and define the reachability, liveness and deadlock problems.

### 2.2.1 Place/Transition Nets

**Definition 1** A *Place/Transition net*, or a *net*, is a four-tuple $N = (P, T, F, M_0)$ such that

1. $P$ and $T$ are disjoint sets; their elements are called *places* and *transitions*, respectively.

2. $F \subseteq (P \times T) \cup (T \times P)$; $F$ is called the *flow relation*.

3. $M_0 \colon P \to \mathbb{N}$; $M_0$ is called the *initial marking* of $N$; in general, a mapping $M \colon P \to \mathbb{N}$ is called a *marking* of $N$.

□

Given $a \in P \cup T$, the *preset* of $a$, denoted by ${}^\bullet a$, is defined as $\{a' \mid a'Fa\}$; the *postset* of $a$, denoted by $a^\bullet$, is defined as $\{a' \mid aFa'\}$.

Sometimes, we denote that a transition $t$ has preset $I$ and postset $O$ in the following way:

$$t : I \ \rightarrow \ O$$

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

We will let $+$ denote union of multisets.

Let $N = (P, T, F, M_0)$ be a net. A transition $t \in T$ is *enabled* at a marking $M$ of $N$ if $M(p) > 0$ for every place $p$ in the preset of $t$. Given a transition $t$, we define a relation $\xrightarrow{t}$ between markings as follows: $M \xrightarrow{t} M'$ if $t$ is enabled at $M$ and $M'(s) = M(s) + F(t,s) - F(s,t)$, where $F(x,y)$ is 1 if $(x,y) \in F$ and 0 otherwise. The transition $t$ is said to *occur* (or *fire*) at $M$. If $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} M_n$ for some markings $M_0, M_1, \ldots M_n$, then the sequence $\sigma = t_1 \ldots t_n$ is called an *occurrence sequence*. $M_n$ is the marking *reached* by $\sigma$, and this is denoted $M_0 \xrightarrow{\sigma} M_n$. A marking $M$ is *reachable* if it is the marking reached by some occurrence sequence. Given a marking $M$ of $N$, the set of reachable markings of the net $(P, T, F, M)$ (i.e., the net obtained replacing the initial marking $M_0$ by $M$) is denoted by $[M\rangle$.

Notice that the empty sequence is an occurrence sequence and that it reaches the initial marking $M_0$.

A net $N$ is *unary* if at every reachable marking at most one transition is enabled. $N$ is *1-conservative* if for every transition $t$, $|{}^\bullet t| = |t^\bullet|$.

### 2.2.2 1-safe Nets

**Definition 2** A marking $M$ of a net $N$ is *1-safe* if for every place $p$ of the net $M(p) \leq 1$. We identify a 1-safe marking $M$ with the set of places $p$ such that $M(p) = 1$. A net $N$ is *1-safe* if all its reachable markings are 1-safe. □

Next, we define our three main problems.

### 2.2.3 Reachability, Liveness, and Deadlock Problems

**Definition 3** The *reachability problem* for a net $N$ is the problem of deciding for a given marking $M$ of $N$ if it is reachable.

A net $N$ is live if for every transition $t$ of $N$ and every reachable marking $M$, some marking of $[M\rangle$ enables $t$. The *liveness problem* for a net is the problem of deciding if it is live.

A marking of a net is a *deadlock* if it enables no transitions. The *deadlock problem* for a net is the problem of deciding if any of its reachable markings is a deadlock. □

## 2.3 Complexity of Place/Transition Nets

For Place/Transition nets, it is known that the liveness and reachability problems are recursively equivalent [Hac74], and that they are both decidable and EXPSPACE-hard [Lip76]. In this section we complete the picture by showing that the deadlock problem is recursively equivalent to them, and thus decidable and EXPSPACE-hard.

**Theorem 4** *Reachability is polynomial-time reducible to deadlock.*

**Proof.** Given $N = (P, T, F, M_0)$, and a marking $M$ of $N$, we construct a net $N' = (P', T', F', M_0')$, as follows. Let $V$ be the set of places marked in $M$. The places and transitions of $N'$ are:

$$P' = P \cup \{p_t \mid t \in T\} \cup \{b_q, c_q \mid q \in V\}$$
$$T' = \{t_c \mid t \in T\} \cup \{t_p \mid p \in P\} \cup \{terminate\} \cup \{sub_q, loop_q \mid q \in V\}$$

The flow relation of $N'$ is given by:

| | | | |
|---|---|---|---|
| For each $t \in T$: | $t_c$ | : | ${}^\bullet t + p_t \rightarrow t^\bullet + p_t$ |
| For each $p \in P$: | $t_p$ | : | $p \rightarrow p$ |
| | $terminate$ | : | $\sum_{t \in T} p_t \rightarrow \sum_{q \in V} b_q$ |
| For each $q \in V$: | $loop_q$ | : | $c_q \rightarrow c_q$ |
| For each $q \in V$: | $sub_q$ | : | $c_q + q + b_q \rightarrow b_q$ |

Finally,

$$M_0' = M_0 + \sum_{q \in V} \alpha_q c_q + \sum_{t \in T} p_t$$

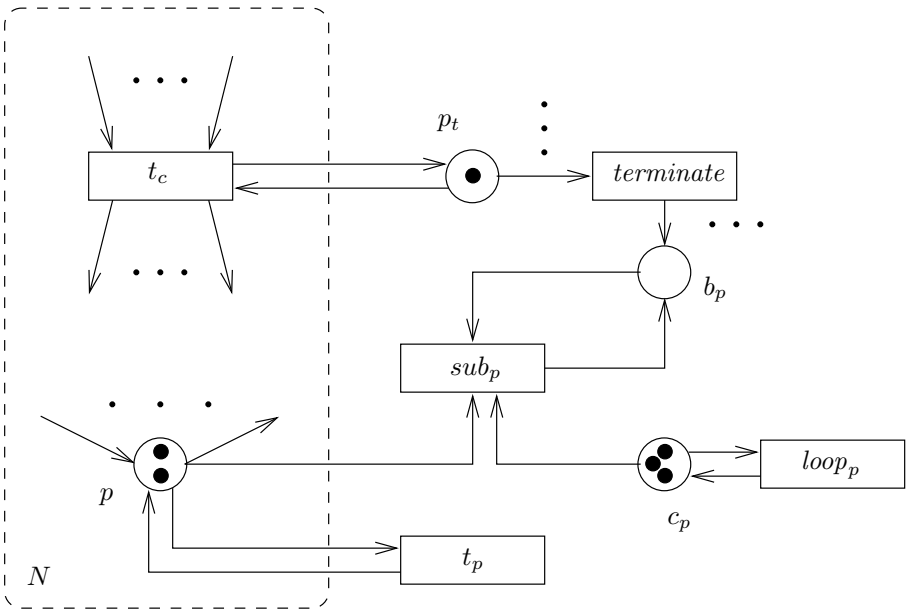



Figure 2.1: Reducing reachability to deadlock.

where

$$M = \sum_{q \in V} \alpha_q q, \quad \alpha_q > 0$$

The construction of $N'$ is illustrated in Figure 2.1.

Claim: $M$ is reachable in $N$ if and only if $N'$ has a deadlock. To see this, first notice that *terminate* can occur at most once, that this disables all the $t_c$ transitions, and that as long as it has not occurred, no marking can be dead: *terminate* can occur.

Suppose now that $M$ is reachable in $N$. Having reached $M$ in $N'$ firing only $t_c$ transitions, fire the *terminate* transition and use the $sub_q$ transitions to remove, for each $q \in V$, $\alpha_q$ tokens from $q$. This yields a dead marking.

Suppose then that $M$ is *not* reachable in $N$. Before *terminate* has fired, there is no deadlock. When *terminate* has fired, no transition in $N$ can fire. There are two cases. Suppose first that $M$ is the empty marking. Since $M$ is not reachable in $N$, there are still tokens in $N$. Thus, at least one $t_p$ transition will remain enabled. Suppose then that $M$ is a non-empty marking. If there are no tokens in $N$, then at least one $loop_q$ transition will remain enabled. If there are still tokens in $N$, then at least one $t_p$ transition will remain enabled. ■

**Theorem 5** *Deadlock is polynomial-time reducible to liveness.*

**Proof.** Given $N = (P, T, F, M_0)$, we construct a net $N' = (P', T', F', M_0')$, as follows. The places and transitions of $N'$ are:

$$P' = P \cup \{ok\}$$
$$T' = \{t_c, t' \mid t \in T\} \cup \{live\}$$

The flow relation of $N'$ is given by:

$$\begin{array}{lrcl} \text{For each } t \in T\text{:} & t_c & : & {}^\bullet t \ \rightarrow \ t^\bullet \\ \text{For each } t \in T\text{:} & t' & : & {}^\bullet t \rightarrow ok \\ & live & : & ok \ \rightarrow \ P' \end{array}$$

Finally, $M'_0 = M_0$.

**Claim 6** *$N$ has no reachable dead marking if and only if $N'$ is live.*

To see this, suppose first that $N$ can reach a dead marking $M_d$. Clearly, also $N'$ can reach $M_d$ without firing any $t'$ transitions, and since the $t'$ transitions in $N'$ have the same presets as the transitions in $N$, $M_d$ is dead in $M'$. Thus, $N'$ is not live.

Suppose then that $N$ has no reachable dead marking. Then the initial marking is not dead, so fire one of the $t'$ transitions. This places a token on the *ok* place, and there the token remains. Thus from now on, the *live* transition is enabled, and because the *live* transition places tokens on *all* places in $N'$, $N'$ is live. ■

**Corollary 7** *The deadlock, liveness and reachability problems are recursively equivalent. Thus, the deadlock problem is decidable and EXPSPACE-hard.*

**Proof.** For the equivalence of the problems, combine Theorem 4 and 5 with Hack's reduction from liveness to reachability [Hac74]. For the complexity of the deadlock problem, use the equivalence with reachability and obtain the decidability from Mayr [May84] and the EXPSPACE-hardness from Lipton [Lip76]. ■

The technique of the proofs is similar to those of, e.g., Chapter 5 in [Pet81].

The same result holds for Place/Transition nets with arc weights. To see this, just observe that our constructions can still be applied and that Hack considers nets with arc weights [Hac74].

## 2.4 Complexity of 1-safe Nets

In this section we prove that the reachability, liveness, and deadlock problems for 1-safe nets are PSPACE-complete.

Given a Place/Transition net, it is PSPACE-complete to decide if the net is 1-safe [JLL77, Corollary 3.4]. However, it is many times possible to guarantee 1-safeness be construction. Consider for instance the important case where the nets are constructed as a synchronisation of finites automata, or the class of Well-Terminating Nets [Jat93].

First we consider the liveness problem.

**Theorem 8** *The liveness problem for 1-safe nets is PSPACE-complete.*

**Proof.** To prove that the liveness problem is in PSPACE, we can use essentially the technique of Jones, Landweber, and Lien [JLL77, Theorem 3.9]. They proved that the liveness problem for 1-conservative (not necessarily 1-safe) nets is in PSPACE.

To prove completeness, we show that the problem (DETERMINISTIC) LINEAR BOUNDED AUTOMATON ACCEPTANCE (which is PSPACE-complete [GJ79]) is polynomial-time reducible to the liveness problem. A linear bounded automaton is a Turing machine which only visits the cells of the tape containing the input. The input is bounded by a left and a right marker, say # and \$, and the head can visit no cell to the left of # and no cell to the right of \$ (see [HU79] for a formal definition). The problem is defined as follows:

Given: a deterministic linearly bounded automaton $\mathcal{M}_0$ and an input $x$ for $\mathcal{M}_0$,

To decide: if $\mathcal{M}_0$ accepts $x$.

First, we construct in polynomial time a deterministic linearly bounded automaton $\mathcal{M}$, satisfying the following two properties:

(1) $\mathcal{M}$ accepts $x$ iff $\mathcal{M}_0$ accepts $x$, and

(2) $\mathcal{M}$ has a unique accepting configuration.

$\mathcal{M}$ simulates $\mathcal{M}_0$, but, before accepting, $\mathcal{M}$ erases the tape, moves the head to the leftmost cell, and then enters its unique final state (a new state not present in $\mathcal{M}_0$). In this way, $\mathcal{M}$ satisfies (2).

Let $\mathcal{M} = (K, \Sigma, \Gamma, \delta, q_1, q_2, \#, \$)$, where $K$ is the set of states, $\Sigma$ the alphabet, $\Gamma \supseteq \Sigma \cup \{\#, \$\}$ is the set of tape symbols, $\delta$ is the transition relation, $q_1$ the initial state, $q_2$ the final state, and # and \$ are the boundary symbols. Moreover, let $K = \{q_1, \ldots, q_m\}$, $\Gamma = \{a_1, \ldots, a_p\}$, $n$ = the size of $\#x\$$, and $\beta = K \times \Gamma \times \{C, R, L\} \times K \times \Gamma$ (i.e., the transition relation is a subset of $\beta$).

We construct a 1-safe net $N = (P, T, F, M_0)$ as follows:

- $P = \{A_{i,j} \mid 1 \le i \le n, 1 \le j \le p\} \cup \{Q_{i,j} \mid 1 \le i \le n, 1 \le j \le m\} \cup \{B, C\}$

  $P$ contains a place $A_{i,j}$ for every tape cell $i$ and every tape symbol $a_j$; a token in $A_{i,j}$ means that the symbol on tape cell $i$ is $a_j$. It also contains a place $Q_{i,j}$ for every tape cell $i$ and every state $q_j$; a token in $Q_{i,j}$ means that the automaton scans the cell $i$ in state $q_j$. Given a configuration $c$ of the automata $\mathcal{M}$, $c$ can be encoded as a subset of $P$ in the following way:

  - if the automaton is in state $q_j$ scanning the $i$-th tape cell, then $Q_{i,j}$ belongs to the set,
  - if the tape cell $i$ contains the symbol $a_j$, then $A_{i,j}$ belongs to the set, and
  - no other place belongs to the set.

Denote the set of places associated to the configuration $c$ by $M(c)$. Notice that $M(c)$ can also be interpreted as a 1-safe marking of $N$.

$B$ and $C$ play the role of a switch, as follows. If there is a token on $B$, then the net simulates $\mathcal{M}$; if there is a token on $C$, then the net behaves nondeterministically in such a way that any marking corresponding to a configuration of the linear automaton can be reached.

- $T$ contains the following transitions for every element of $\beta$:
  - If $(q_s, a_t, R, q_r, a_l) \in \delta$ (move right), then $T$ includes for every cell $1 \leq i < n$ a transition

    $$Q_{i,s} + A_{i,t} \;\; \rightarrow \;\; Q_{i+1,r} + A_{i,l}$$

    (where we use $+$ instead of set union to use the notation of [JLL77]; notice that no transition is needed for the $n$-th cell). Similarly for left moves and no motion. The transitions corresponding to an element of $\beta \setminus \delta$ have $C$ in their preset, and can therefore only occur if $C$ is marked.
  - If $(q_s, a_t, R, q_r, a_l) \in \beta \setminus \delta$, then $T$ includes for every cell $1 \leq i < n$ a transition

    $$C + Q_{i,s} + A_{i,t} \rightarrow Q_{i+1,r} + A_{i,l} + C$$

    Similarly for left moves and no motion.
  - $T$ contains the following two transitions $t_{B\rightarrow C}$, $t_{C\rightarrow B}$, where $c_i$ is the initial configuration of $\mathcal{M}$, and $c_f$ its unique accepting configuration.

    $$t_{B\rightarrow C}: \qquad B + M(c_f) \;\; \rightarrow \;\; C + M(c_f)$$

    If the net reaches the marking corresponding to the accepting configuration $c_f$, then the transition $t_{B\rightarrow C}$ can occur and the net starts behaving nondeterministically in such a way that for any configuration $c$, the marking $C + M(c)$ is reachable.

    $$t_{C\rightarrow B}: \qquad C + M(c_i) \;\; \rightarrow \;\; B + M(c_i)$$

    The net can return to simulating $\mathcal{M}$ if, while behaving nondeterministically, it reaches the marking corresponding to the initial configuration.
- The initial marking $M_0$ is the one corresponding to the initial configuration, plus one token on the place $B$ i.e., $M_0 = B + M(c_i)$

If $\mathcal{M}$ does not accept $x$, then $N$ never reaches the marking $B + M(c_f)$, corresponding to the accepting configuration $c_f$. This implies that the transition $t_{B\rightarrow C}$ can never occur, and therefore $N$ is not live.

If $\mathcal{M}$ accepts $x$, then the net reaches the accepting configuration $c_f$. So the transition $t_{B\rightarrow C}$ can occur, and $N$ starts behaving nondeterministically. Now, for every possible configuration $c$, the net can reach $C + M(c)$. Hence every transition, but $t_{B\rightarrow C}$, can become enabled at some reachable marking containing $C$. In particular, the marking $M(c_i) + C$ can be reached too; this marking enables $t_{C\rightarrow B}$. Therefore, the net can return to simulating $\mathcal{M}$, and everything starts anew, in particular $t_{B\rightarrow C}$ can occur again. ■

We now consider the reachability problem. It is again possible to use a reduction from linear bounded automaton acceptance. we prefer to give another reduction from quantified boolean formulas. This reduction has some interest in itself in that it is easier to modify as will be clear from Sect. 2.6, and moreover shows that the problem is still PSPACE-complete even if restricted to unary 1-safe nets.

First we prove the following useful lemma.

**Lemma 9** *Given a 1-safe net $N$ and a 1-safe marking $M$, checking whether $M$ is reachable in $N$ is in PSPACE.*

**Proof.** Store $M_0$ as the current marking. Set up an $m$-bit counter initialised to 0, where $m = |P|$. Repeatedly do the following. Check if the current marking equals $M$. If so, $M$ is reachable. If not, check if the counter's value equals $2^m$. If so, $M$ is not reachable, since any occurrence sequence longer than $2^m$ must have loops which do not contribute to exploring the state space. If not, the counter's value is less than $2^m$. If the current marking is a deadlock, then $M$ is not reachable. Otherwise, choose an enabled transition, fire it, store the new marking as the current marking, and increment the counter.

The above algorithm uses $2|P|$ bits, and is thus in PSPACE. ■

**Theorem 10** *The reachability problem for both 1-safe nets and unary 1-safe nets is PSPACE-complete.*

**Proof.** By Lemma 9, the reachability problem is in PSPACE.

To prove PSPACE-hardness, we show that QUANTIFIED BOOLEAN FORMULAS (which is PSPACE-complete [GJ79]) is polynomial-time reducible to the reachability problem. The problem is defined as follows:

> Given: A well-formed quantified Boolean formula
>
> $$\mathcal{F} = (Q_1 x_1)(Q_2 x_2) \cdots (Q_n x_n) E$$
>
> where $E$ is a Boolean expression involving the variables $x_1, x_2, \ldots, x_n$ and each $Q_i$ is either "$\exists$" or "$\forall$".
>
> To decide: is $\mathcal{F}$ true?

If we are given a quantified boolean formula $\mathcal{F}$, then we construct a unary 1-safe net $N$ and a marking $M$ of $N$ such that $M$ is reachable if and only if $\mathcal{F}$ is true.

Before constructing the net and the marking, we rewrite $\mathcal{F}$, in polynomial time, into an equivalent closed formula $G$ generated by the grammar:

$$P \quad ::= \quad x \mid \neg P \mid P \wedge P \mid \exists x.P \ ,$$

and such that all bound variables in $G$ are distinct. Notice that $G$ needs not be a quantified boolean formula: the quantifiers in $G$ need not occur at the outermost level.

The construction of the net for $G$ is illustrated in Figure 2.6–2.5. Intuitively, the idea is to try all possible assignments of bound variables. The construction is essentially compositional. The only complication is the interpretation of variables.

The net for $G$ contains the places:

$$\{P\_in, P\_T, P\_F \mid P \text{ is an occurrence of a subformula of } G\} \cup$$
$$\{x\_is\_T, x\_is\_F \mid x \text{ is bound in } G\}$$

For readability, when in the following we name places and transitions, we write $not\_P$ for $\neg P$, we write $P\_and\_Q$ for $P \wedge Q$, and we write $Ex.P$ for $\exists x.P$.

The initial marking is $\{G\_in\}$.

The net for $G$ contains the following transitions for each occurrence of a subformula of $G$:

| Occurrence | Transitions | | |
|---|---|---|---|
| $x$ | $read\_x\_is\_T$ | : | $x\_in + x\_is\_T \rightarrow x\_T + x\_is\_T$ |
| | $read\_x\_is\_F$ | : | $x\_in + x\_is\_F \rightarrow x\_F + x\_is\_F$ |
| $\neg P$ | $call\_P$ | : | $not\_P\_in \rightarrow P\_in$ |
| | $not\_P\_is\_F$ | : | $P\_T \rightarrow not\_P\_F$ |
| | $not\_P\_is\_T$ | : | $P\_F \rightarrow not\_P\_T$ |
| $P \wedge Q$ | $call\_P$ | : | $P\_and\_Q\_in \rightarrow P\_in$ |
| | $P\_T\_and\_Q\_?$ | : | $P\_T \rightarrow Q\_in$ |
| | $P\_F\_and\_Q\_?$ | : | $P\_F \rightarrow P\_and\_Q\_F$ |
| | $P\_T\_and\_Q\_T$ | : | $Q\_T \rightarrow P\_and\_Q\_T$ |
| | $P\_T\_and\_Q\_F$ | : | $Q\_F \rightarrow P\_and\_Q\_F$ |
| $\exists x.P$ | $call\_P\_with\_x\_T$ | : | $Ex.P\_in \rightarrow P\_in + x\_is\_T$ |
| | $call\_P\_with\_x\_F$ | : | $x\_is\_T + P\_F \rightarrow x\_is\_F + P\_in$ |
| | $x\_T\_P\_T$ | : | $x\_is\_T + P\_T \rightarrow Ex.P\_T$ |
| | $x\_F\_P\_T$ | : | $x\_is\_F + P\_T \rightarrow Ex.P\_T$ |
| | $Ex.P\_is\_F$ | : | $x\_is\_F + P\_F \rightarrow Ex.P\_F$ |

To avoid name clashes we could let the name of an occurrence of a subformula of $G$ contain its position in the syntax tree for $G$. We omit these details, for readability.

Intuitively, when $P\_in$ ("the in-place for $P$") becomes marked, then the checking of the truth of $P$ begins. When either $P\_T$ ("true") or P_F ("false") becomes marked, this checking is completed. Let us consider in turn the construction for each of the productions of the above grammar.

First, consider a variable $x$, see Figure 2.2. The places $x\_is\_T$ ("$x$ is true") and $x\_is\_F$ ("$x$ is false") are not part of the net for $x$ but are included to indicate that they will be added when treating the quantification that binds $x$. Note that all occurrences of the same variable $x$ share these two places. The two transitions implements the reading of the current value of $x$.

Second, consider a negation $\neg P$, see Figure 2.3. The transition $call\_P$ transfers the "control" to the subnet for $P$. The two other transitions implement the negation.

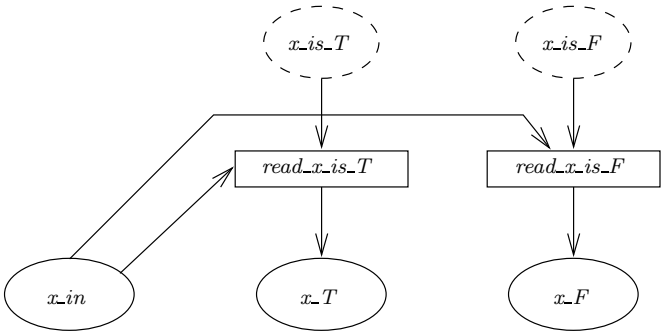



Figure 2.2: Reduction from quantified boolean formulas, variable.

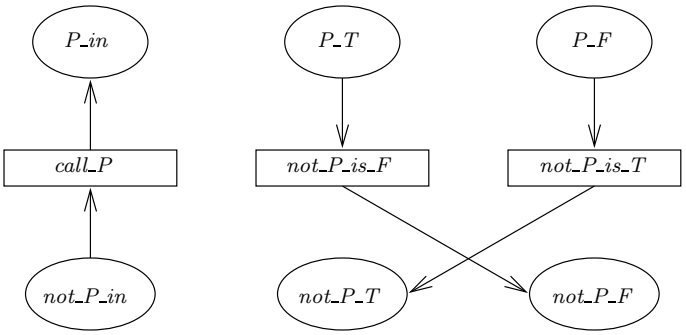



Figure 2.3: Reduction from quantified boolean formulas, negation.

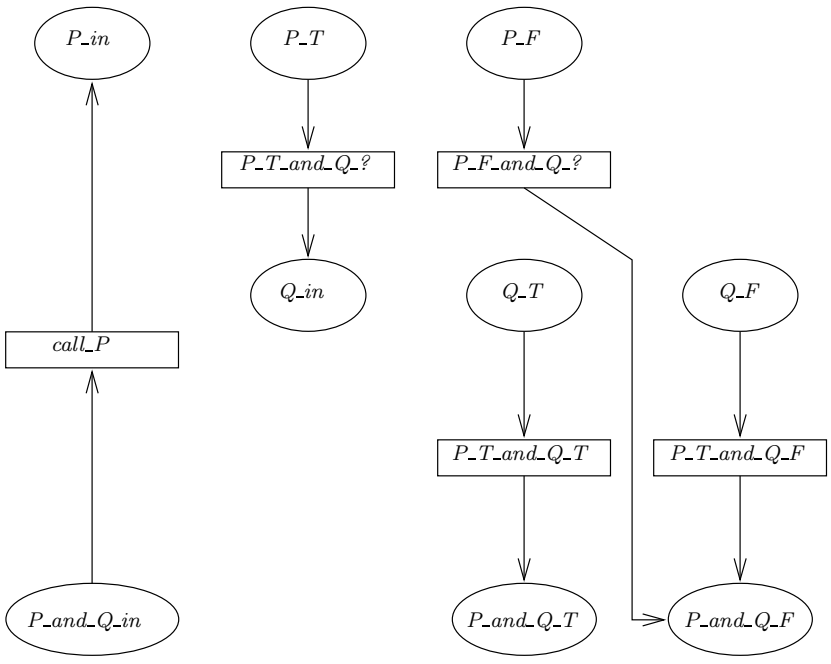



Figure 2.4: Reduction from quantified boolean formulas, conjunction.

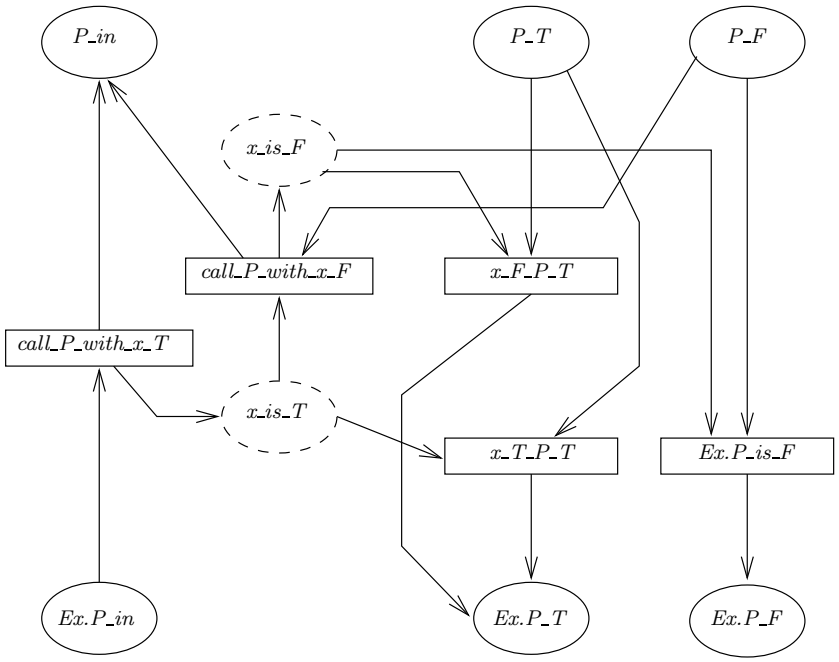



Figure 2.5: Reduction from quantified boolean formulas, existential quantifier.

Third, consider a conjunction $P \wedge Q$, see Figure 2.4. The transition $call\_P$ transfers the "control" to the subnet for $P$. The four other transitions implement the conjunction.

Fourth, consider an existential quantification $\exists x.P$, see Figure 2.5. The places $x\_is\_T$ ("$x$ is true") and $x\_is\_F$ ("$x$ is false") are the ones we mentioned above. The transition $call\_P\_with\_x\_T$ assigns true to $x$ and transfers the "control" to the subnet for $P$. In case $P$ was not true, the transition $call\_P\_with\_x\_F$ assigns false to $x$ and transfers again the "control" to the subnet for $P$.

If a formula $P$ is open, then we can obtain an *extended* net for $P$ as follows. For every free variable $x$ in $P$ we extend the net with two places $x\_is\_T$ and $x\_is\_F$ and mark exactly one of them. This marking may be thought of as assigning a value to $x$.

The following fact expresses a relation between each formula $P$ and the extended net for $P$. The proof is by straightforward induction on the structure of $P$.

**Fact 11** *Let $P$ be a formula generated from the above grammar and consider the extended net for $P$. In the following we discount the marking of the places for free variables; the marking of these are invariant. From the marking $\{in\_P\}$, eventually either $\{P\_T\}$ or $\{P\_F\}$ will be reached. The former is reached if and only if $P$ is true under the given assignment of its free variables, and the latter if not.*

Using this observation it is easy to see that the marking $\{G\_T\}$ is reachable in the net for $G$ if and only if $G$ is true.

Clearly, the net for $G$ is 1-safe. Notice that for each reachable marking at most one transition is enabled. ■


Figure 2.6: Reduction from quantified boolean formulas.

**Theorem 12** *The deadlock problem for 1-safe nets is PSPACE-complete.*

**Proof.** To show that the deadlock problem is in PSPACE, given a 1-safe net $N$ guess a marking $M$ of $N$, and check in linear, and by Lemma 9, check in PSPACE if $M$ is reachable.

To prove completeness, we reduce the problem QUANTIFIED BOOLEAN FORMULAS to the deadlock problem. Extend the net in the proof of Theorem 10 with the transition

$$G\_F \;\rightarrow\; G\_F$$

Clearly, the new net has a deadlock if and only if $\mathcal{F}$ is true. ■

The deadlock and reachability problems turn out to be PSPACE-complete even for 1-conservative unary 1-safe nets. This follows directly from the constructions in the proof of Theorem 10 and the following "conservativeness" observation.

First, we define the notion of reachability graph. The reachability graph of a net $N$ is the edge-labelled graph whose vertices are the reachable markings of $N$; if $M \xrightarrow{t} M'$ for a reachable marking $M$, then there is an edge from $M$ to $M'$ labelled with $t$.

**Fact 13** *There is a linear time algorithm which converts a 1-safe net $N$ into a 1-conservative 1-safe net $N'$ with the following property: there exists a simple function $f$ from the markings of $N$ to the markings of $N'$ such that (1) $M$ is reachable in $N$ iff $f(M)$ is reachable in $N'$; (2) the initial marking of $N$ is mapped by $f$ to the initial marking of $N'$; and (3) $M$ is a deadlock of $N$ iff $f(M)$ is a deadlock of $N'$. Hence the construction 'preserves' reachability and the existence of deadlocks.*

For $N = (P, T, F, M_0)$, the net $N'$ is constructed by adding for every place $p$ of $P$ a new place $\overline{p}$ called the *complement of* $p$. Then, for every arc $(p, t)$ of $F \setminus F^{-1}$, a new arc $(t, \overline{p})$ is added; similarly, for every arc $(t, p)$ of $F \setminus F^{-1}$, a new arc $(\overline{p}, t)$ is added. Finally $M_0'$ is defined by $M_0'(p) = M_0(p)$ for every place $p$ of $N$, and $M_0'(\overline{p}) = 1 - M_0(p)$ for each complement place. The construction is very similar to the one of [Rei85], and therefore we omit the proof of the result; the only difference is the special treatment of the case in which two arcs $(p, t)$ and $(t, p)$ exist.

## 2.5 Subclasses

In this section we study the complexity of the three chosen problems for three subclasses of nets which have been often studied in the literature. Most results are already known;

we have collected them and filled some gaps. The nets of these subclasses satisfy some structural condition that rules out some basic kind of behaviours. In our first case, the *acyclic* nets, recursive or iterative behaviours are forbidden. The *conflict-free* nets do not allow nondeterministic behaviours (actually, this depends slightly on the notion of nondeterminism used). Finally, *free-choice* nets restrict the interplay between nondeterminism and synchronisations. In particular, in 1-safe free-choice net the phenomenon known as *confusion* [Thi87] is ruled out.

### 2.5.1 Acyclic nets

**Definition 14** A net $N = (P, T, F, M_0)$ is said to be acyclic if $F^+$ (the transitive closure of $F$) is irreflexive. □

The reachability problem remains intractable for acyclic 1-safe nets, although the problem is no longer PSPACE-complete (assuming NP $\neq$ PSPACE).

**Theorem 15** *The reachability problem for acyclic 1-safe nets is NP-complete.*

**Proof.** The problem is in NP because in an occurrence sequence of a 1-safe acyclic net each transition occurs at most once. So we can guess an occurrence sequence in linear time and check in polynomial time if it leads to the given marking.

For the completeness part, see the paper by Stewart [Ste92]. The result is proved by means of a reduction from the HAMILTONIAN CIRCUIT problem. ■

Since all 1-safe acyclic nets contain deadlocks, the liveness and deadlock problems are trivial.

We can compare these results with the ones corresponding to the general case.

**Theorem 16** *The reachability problem for acyclic Place/Transition nets is NP-complete.*

**Proof.** The problem can be polynomial-time reduced to INTEGER LINEAR PROGRAMMING, because in an acyclic net $N$ with initial marking $M_0$, a marking $M$ is reachable iff the system of equations corresponding to the state equation $M = M_0 + C \cdot X$, where $C$ is the incidence matrix of $N$, has an integer vector solution $X$ (for the definitions of incidence matrix and state equation, see, for instance, [Mur89]). Since INTEGER LINEAR PROGRAMMING is in NP [HU79], so is our problem.

The completeness follows trivially from the completeness of the problem for the 1-safe case. ■

It is easy to see that an acyclic net has no deadlocks if and only if some of its transitions has empty preset; therefore the deadlock problem can easily be solved in linear time. Similarly, an acyclic net is live if and only if every place has some input transition; so the liveness problem is also linear. So, as we can see, there are no essential differences between the general and the 1-safe case.

### 2.5.2 Conflict-free nets

Conflict-free nets are a subclass in which conflicts are structurally ruled out (actually, this depends slightly on the notion of conflict used). Their complexity has been deeply studied in several papers; in particular, the complexity of our three problems.

**Definition 17** A net $N = (P, T, F, M_0)$ is *conflict-free* if for every place $p$, if $|p^\bullet| > 1$, then $p^\bullet \subseteq {}^\bullet p$. □

It is shown by Howell and Rosier in [HR88, HR89] that the reachability, liveness, and deadlock problems for 1-safe conflict-free nets are solvable in polynomial time. They also show that, for Place/Transition nets, the deadlock and liveness problems are still polynomial, whereas the reachability problem becomes NP-complete [HR88, HR89].

### 2.5.3 Free-Choice nets

Free-choice nets are a well studied class, commonly acknowledged to be about the largest class having a nice theory.

**Definition 18** A net $N = (P, T, F, M_0)$ is *free-choice* if for any pair $(p, t) \in F \cap (P \times T)$ it is the case that $p^\bullet = \{t\}$ or ${}^\bullet t = \{p\}$. □

In a free-choice net, if some transitions share an input place $p$, then $p$ is their unique input place. It follows that if any of them is enabled, then all of them are enabled. Therefore, it is always possible to freely choose which of them occurs.

The reachability problem is still PSPACE-complete for 1-safe free-choice nets. The reason is that for a 1-safe net $N$ and a marking $M$, we can construct a 1-safe free-choice net $N'$ containing all the places of $N$ (and possibly more), such that $M$ is reachable in $N$ if and only if it is reachable in $N'$. $N'$ is the so called 'released form' of $N$. Intuitively, every arc $(p, t)$ such that $|p^\bullet| > 1$ and $|{}^\bullet t| > 1$ is removed and replaced by new arcs $(p, t'), (t', p'), (p', t)$, where $p'$ and $t'$ are a new place and a new transition. The interested reader can find a formal definition in [JLL77, Hac76]. Figure 2.7 shows a non-free-choice net (on the left), and its released form (on the right).

Perhaps surprisingly, the liveness problem is polynomial for this class.

**Theorem 19** *The liveness problem for free-choice 1-safe nets is solvable in polynomial time.*

**Proof.** See the paper by Esparza and Silva [ES92], and the paper by Desel [Des92]. ■

We now show that the deadlock problem for 1-safe free-choice nets is NP-complete. Membership in NP is non-trivial, and requires to introduce some concepts and results of net theory.

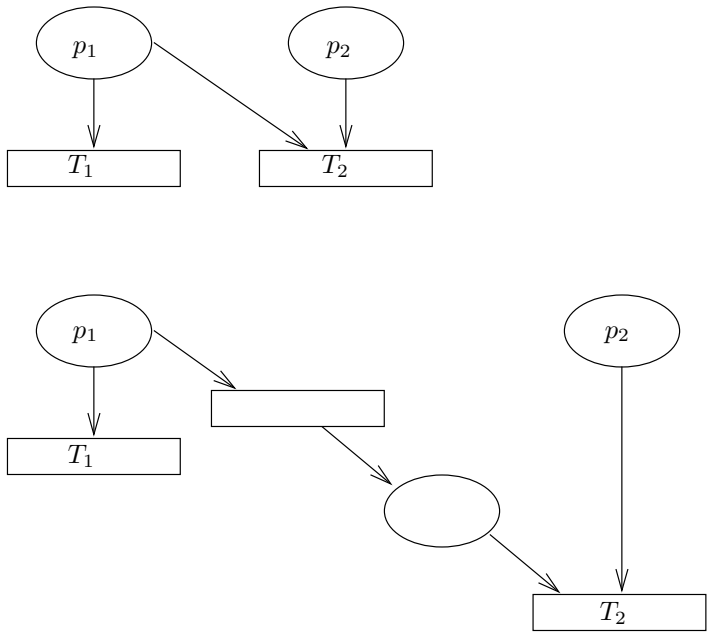



Figure 2.7: A net and its released form.

Let $N$ be a net and $Q$ a set of places of $N$. For a marking $M$ of $N$, $M(Q)$ denotes the total number of tokens that $M$ puts in the places of $Q$ (formally, $M(Q) = \sum_{p \in Q} M(p)$. The set $Q$ is said to be marked at $M$ if $M(Q) > 0$, and unmarked at $M$ if $M(Q) = 0$.

A subset $Q$ of places of $N$ is a *siphon* if ${}^\bullet Q \subseteq Q^\bullet$, and a *trap* if $Q^\bullet \subseteq {}^\bullet Q$.

We use some well known lemmata about siphons and traps. They can all be found in [Hac72] or—a more accessible reference—in [BD90].

**Lemma 20** *Let $N$ be a net, and $M$ a marking of $N$.*

*(1) If $Q$ is a siphon of $N$ unmarked at $M$, then $Q$ remains unmarked at all markings reachable from $M$.*

*(2) If $Q$ is a trap of $N$ marked at $M$, then $Q$ remains marked at all markings reachable from $M$.*

**Proof.** Follows easily from the definitions of siphon, trap, and the occurrence rule. ■

**Lemma 21** *Let $M$ be a deadlock of a net $N$. Then, the set of places of $N$ unmarked at $M$ is a siphon of $N$.*

**Proof.** Let $Q$ be the set of places of $N$ unmarked at $M$. It suffices to observe that, since $M$ is a deadlock, every transition has some place in its preset which is unmarked at $M$. So $Q^\bullet$ contains all the transitions of $N$ and, since ${}^\bullet Q$ is a subset of them, $Q$ is a siphon. ■

**Lemma 22** *Let $N$ be a free-choice net with initial marking $M_0$. Let $Q$ be a siphon of $N$ which contains no trap marked at $M_0$. Then, there exists a reachable marking $M$ such that $Q$ is unmarked at it.*

**Proof.** See [Hac72, BD90]. This result is part of the proof of Commoner's theorem. ∎

Using these lemmata, we can now characterise when a free-choice net has a deadlock.

**Lemma 23** *Let $N$ be a free-choice net. $N$ has a deadlock iff there exists a siphon $Q$ of $N$ such that:*

*(1) for every transition $t$ of $N$, $Q$ contains some place of ${}^\bullet t$, and*

*(2) $Q$ contains no trap marked at the initial marking.*

**Proof.** ($\Rightarrow$): Let $M$ be a deadlock of $N$. Define $Q$ as the set of places of $N$ unmarked at $M$. By Lemma 21, $Q$ is a siphon. Since no transition of $N$ is enabled at $M$, we have that, for every transition $t$, $Q$ contains some place of ${}^\bullet t$.

To prove (2), assume that $Q$ contains a trap marked at the initial marking. Then, since marked traps remain marked by Lemma 20, this trap is marked at $M$. So $Q$ is marked at $M$ too, which contradicts the definition of $Q$.

($\Leftarrow$): By Lemma 22, there exists a reachable marking $M$ such that $M(Q) = 0$. Since $Q$ contains some place of the preset of each transition, no transition is enabled at $M$. So $M$ is a deadlock. ∎

**Theorem 24** *The deadlock problem for 1-safe free-choice nets is NP-complete.*

**Proof.** To solve the problem in nondeterministic polynomial time, we use Lemma 23. Guess for each transition $t$ of the net a place of ${}^\bullet t$. Check in polynomial time if the guessed set of places is a siphon; then, check in polynomial time that it contains no trap marked at the initial marking using Starke's algorithm to find the maximal trap contained in a given siphon [Sta90] (see [DE91] for a reference in English).

We prove completeness by reducing the satisfiability problem of propositional formulas in conjunctive normal form (CON-SAT) to the deadlock problem.

An instance $\varphi$ of CON-SAT is a conjunction of clauses $C_1, \ldots, C_m$ over variables $x_1, \ldots, x_n$. A clause is a disjunction of literals. A literal $l_i$ is either a variable $x_i$ or its negation $\overline{x_i}$.

Given an instance $\varphi$ of CON-SAT, we construct a free-choice net $N$ in polynomial time and show that that it has a deadlock iff $\varphi$ is satisfiable. The construction is very similar to the one used in [JLL77] to prove the NP-completeness of liveness in general free-choice nets. We describe the set $P$ of places and the set $T$ of transitions of $N$, together with their presets and postsets. The set $P$ contains the following elements:

(a) for every $1 \leq i \leq n$, places $A_i$, $x_i$, $\overline{x_i}$,

(b) for each clause $C_j$ and every literal $l_i$ appearing in $C_j$, a place $(l_i, C_j)$, and

(c) for each clause $C_j$, a place $F_j$.

The transitions in $T$ are defined as follows:

(1) for each literal $l_i$, $A_i \to l_i$,

(2) for each literal $l_i$, $l_i \to \sum_{\overline{l_i} \in C_j} (l_i, C_j)$,

(3) for each clause $C_j$, $\sum_{l_i \in C_j} (l_i, C_j) \to F_j$, and

(4) for each clause $C_j$, $F_j \to F_j$.

The marking $M_0$ is the set $\{A_i \mid 1 \leq i \leq n\}$.

An occurrence sequence of $N$ is a *truth sequence* if:

- for every variable $x_i$, it contains one of the two transitions $A_i \to x_i$, $A_i \to \overline{x_i}$, and
- it only enables transitions of type (3), if any.

A truth sequence $\sigma$ is associated to the assignment $f\colon \{x_1, \ldots, x_n\} \to \{\mathsf{true}, \mathsf{false}\}$ given by $f(x_i) = \mathsf{true}$ iff the transition $A_i \to x_i$ occurs in $\sigma$.

The following fact follows easily from the construction of $N$:

**Fact 25** *The marking reached by a truth sequence enables a type (3) transition iff the corresponding clause $C_j$ is false under $f$.*

Assume $\varphi$ is satisfiable. Then, there exists an assignment $f$ which makes all clauses true. By the fact above, any truth sequence associated to $f$ leads to a deadlock.

Now, assume that $M$ is a deadlock of $N$. It follows from the construction that $M$ only marks places of the form $(l_i, C_j)$, and that any occurrence sequence that leads to $M$ is a truth sequence. By the fact above, no clause is false under the assignment associated to $\sigma$. So $\varphi$ is satisfiable. ■

There are differences between the 1-safe and the Place/Transition free-choice nets. Using the releasing technique it is easy to show that the reachability problem for free-choice nets is as hard as the reachability problem for arbitrary Place/Transition nets, and therefore EXPSPACE-hard. The liveness problem was shown to be NP-complete in [JLL77]. Finally, our proof of membership in NP for the deadlock problem did not rely on 1-safeness; therefore, the deadlock problem is also NP-complete for Place/Transition free-choice nets.

## 2.6 Other Problems

There exist other problems concerning Petri nets which have received attention in the literature.

**Definition 26** The *containment problem* for two nets with the same set of places is the problem of deciding whether all reachable markings of the first are reachable in the second. □

**Definition 27** Given two 1-safe markings $M$, $M'$ of a net, $M$ is *covered* by $M'$ if $M \subseteq M'$. The *coverability problem* for a given net $N$ and a marking $M$ of $N$ is the problem of deciding whether some reachable marking of $N$ covers $M$. □

**Definition 28** A net $N$ is said to be *persistent* [LR75] if for every reachable marking $M$, if two different transitions $t, t'$ are enabled at $M$ then $M \xrightarrow{t} M' \xrightarrow{t'} M''$ for some markings $M'$, $M''$. The *persistency problem* for a net is the problem of deciding whether the net is persistent. Notice that unary nets are persistent. □

**Definition 29** Let $N = (P, T, F, M_0)$ be a net. For any subset $T_0$ of $T$ let $h_{T_0}$ be the "erasing" homomorphism from $T^*$ to $T_0^*$ which erases elements from $T \setminus T_0$. For a transition $t \in T \setminus T_0$ we say that $T_0$ controls $t$ by an occurrence sequence $\gamma$ in $T_0^*$ if for every occurrence sequence $\sigma$ from $M_0$, if $h_{T_0}(\sigma) = \gamma$ then $t$ is not enabled at the marking $M$ reached by the occurrence of $\sigma$. Crudely speaking, once $\gamma$ has occurred, even interleaved with transitions of $T \setminus T_0$, $t$ cannot occur until some transition of $T_0$ occurs. $T_0$ is said to control $t$ if $T_0$ can control $t$ by at least one sequence $\gamma$. The *controllability problem* [JLL77] for a net is the problem of deciding whether $T_0$ controls $t$ given $N$, $T_0$, and $t$ as above. □

For arbitrary Petri nets, the containment problem is undecidable [Hac76], whereas the coverability, persistency, and controllability problems are EXPSPACE-hard. It is shown by Howell and Rosier in [HR88, HR89] that the coverability problem for 1-safe conflict-free nets is solvable in polynomial time.

We study the first three of these problems in the 1-safe case.

**Theorem 30** *The containment, coverability and persistency problems for 1-safe nets are PSPACE-complete.*

**Proof.** We show that each of the three problems is in PSPACE. First, consider the containment problem. Given two nets, guess a marking, and by Lemma 9, check in PSPACE that the marking is reachable in the first net and unreachable in the second net. This shows that the containment problem is in PSPACE, since co-NPSPACE equals PSPACE (by Savitch's theorem and because space complexity classes are closed under complementation).

Second, consider the coverability problem. Given a 1-safe net $N$ and a marking $M$ of $N$, guess a marking $M' \supseteq M$ and, by Lemma 9, check in PSPACE that the marking $M'$ is reachable.

Third, consider the persistency problem. Proceed as above, this time guessing a marking $M$ of $N$ that enables two different transitions $t$ and $t'$. If $M$ is reachable, then check in linear space that $t'$ cannot occur after the occurrence of $t$.

To prove that each of the three problems is PSPACE-hard, we use the same construction as in the proof of PSPACE-hardness of reachability. For each of the following arguments, suppose we are given a quantified boolean formula $\mathcal{F}$. To begin with, transform it into an equivalent formula $G$ as was done for Theorem 10.

First, consider the containment problem. Construct both the same 1-safe net $N$ as in the proof of Theorem 10 and the following net $N'$. The net $N'$ is obtained from $N$ by removing all transitions, and taking $\{G\_T\}$ as initial marking. For convenience we construct a net whose places have empty presets and postsets (isolated nodes), see remark at the beginning of Section 2.2. The PSPACE-hardness can be shown for nets satisfying the assumptions of no isolated nodes. Clearly, the set of reachable markings of $N'$ is $\{G\_\mathcal{T}\}$, and therefore it is contained in the set of reachable markings of $N$ if and only if $\mathcal{F}$ is true.

Second, consider the coverability problem. Clearly, there is a reachable marking in $N$ that covers $\{G\_\mathcal{T}\}$ if and only if $\mathcal{F}$ is true.

Third, consider the persistency problem. Extend the net in the proof of Theorem 10 with two new places $\{V, W\}$ and the transitions

$$\begin{aligned} G\_F &\rightarrow V \\ G\_F &\rightarrow W \end{aligned}$$

Clearly, the new net is persistent if and only if $\mathcal{F}$ is true. ■

The proof of the result that controllability is EXPSPACE-complete [JLL77, Theorem 4.1] was in fact given for 1-conservative free-choice nets, and also works when restricted to 1-safe nets. This is the only one of the problems we consider for which the complexity does not decrease for 1-safe nets.

Using the techniques from the proofs of Theorem 10 and 30 one can proceed to prove that numerous other problems for 1-safe nets are PSPACE-complete: "is there an infinite occurrence sequence?", "can a certain transition ever occur?", "is a certain transition live?", etc. The interested reader will find no problem in carrying out the corresponding proofs.

## 2.7 Summary

In this chapter we showed that all problems remain intractable, although, as could be expected, their complexity decreases in comparison with Place/Transition nets. The usual observation is that problems are EXPSPACE-hard for Place/Transition nets and PSPACE-complete in the 1-safe case.

Also, most problems remain intractable even for unary 1-safe nets, which are sequential and deterministic. So it is not possible to relate intractability to nondeterminism or concurrency.

However, some problems become tractable when restricted to subclasses of 1-safe nets defined using *structural* constraints, i.e., constraints on the flow relation.

The most interesting direction for further research is probably the study of the complexity of a problem when a certain desirable property is known to hold, for instance liveness. The result of [DE93b] can be seen as a first step in this direction: it is shown that for live and 1-safe free-choice nets the reachability problem is in NP, by proving that every reachable marking can be reached by an occurrence sequence of polynomial length. Also, the problem is solvable in polynomial time if the net is live, bounded, cyclic, and free-choice [DE93a]. So far nothing is known about the complexity of deciding if a marking is reachable when the Petri net is known to be live.

# Chapter 3

# Compact Systems

**Contents**

| Logic | Problem Instance | Complexity |
|---|---|---|
| CTL | R-structure (Kripke) and a formula | P |
| $\mathsf{L}(F)$ | R-structure and a formula | NP-complete |
| $\mathsf{L}(X,U,S)$ | R-structure and a formula | PSPACE-complete |
| CTL $\mathsf{L}(F)$ $\mathsf{L}(X,U,S)$ | Compact system and a formula | PSPACE |

Table 3.1: Complexity in terms of Kripke structures and compact systems.

## 3.1 Introduction

In the following sections we choose a rather general setting both in terms of the systems considered and the problems to be solved. Namely, we formalise the notion of *compact systems* and investigate the computational complexity of the model-checking problem in terms of the size of a compact system $s$ and the length of the formula $\varphi$. We show that for any class of compact systems the model-checking problems for the well-known temporal logics CTL, $\mathsf{L}(F)$, and $\mathsf{L}(X,U,S)$ are in PSPACE. We have summarised this in Table 3.1.

As an example of the intended use of our results we consider $K$-bounded Petri nets. From the previous chapter we easily conclude that the model-checking problems for CTL, $\mathsf{L}(F)$, and $\mathsf{L}(X,U,S)$ are PSPACE-hard. Having obtained the lower bounds it is now sufficient only to show how $K$-bounded Petri nets can be viewed as a class of compact systems, i.e, show that it is possible to transform the description of the $K$-bounded Petri net—in the problem instance used to obtain the lower bound—into the representation as a compact system, using at most a polynomial amount of space. Our results then allow us to conclude that the problems are PSPACE-complete. In terms of the size of the state spaces of the $K$-bounded Petri nets (and the length of the formulas) the computational complexities are polynomial time [CES86], NP-complete, and PSPACE-complete [SC85], respectively. In terms of the size of the $K$-bounded Petri nets the algorithms in [SC85, CES86] for CTL and $\mathsf{L}(F)$ would both require exponential space since they require the full state space to be explicitly represented.

Since the model-checking problem for temporal logics containing the $F$ (future) operator is usually PSPACE-hard for most nontrivial classes of compact systems, such as $K$-bounded Petri nets, our results provide matching upper bounds.

In general, one cannot obtain similar PSPACE-hardness results for all classes of compact systems; we allow classes of trivial systems as classes of compact systems. $R$-structures represent state spaces in a direct and non-compact manner. Also, for a class of models, lower bounds are obtained by reducing a known hard problem *to* that class,

whereas upper bounds can be obtained by reducing *from* that class (to a class of, e.g., compact systems).

The results are organised as follows. In Sect. 3.2 we give the necessary definitions. This includes compact systems, the logics and their interpretations, and the model-checking problems. Then, in Sect. 3.3 we give the upper bounds on the model-checking problems. In Sect. 3.4 we apply our results to $K$-bounded Petri nets and in Sect. 3.5 we summarise and give suggestions for future work.

## 3.2 Definitions

Intuitively, the systems we consider all have the property that we can associate (Kripke) structures to them which correspond to their state spaces. Moreover, the size of the associated structure is at most exponential in the size of the description of the system. For example, $K$-bounded Petri nets have this property; a net with $n$ places has at most $(K+1)^n$ reachable states. We refer to these systems as *compact systems*. We continue by formalising compact systems, defining the logics $\mathsf{B}(X,U)$ (CTL) and $\mathsf{L}(X,U,S)$ and their interpretations, and finally the model-checking problems.

### 3.2.1 Compact Systems

Intuitively, a class of compact systems consists of string encodings of systems and a polynomial space bounded nondeterministic Turing machine which given an encoding of a system will "simulate" it. The state space of the Turing machine will then be used to define the state space of the input string (the compact system). Below we present $K$-bounded Petri nets as a class of compact systems. For the ease of the presentation we assume that the reader is familiar with Turing machines and basic complexity theory, see [HU79] for an introduction.

**Definition 31** A *class of compact systems*, $(\mathcal{C}, \mathcal{M}_{\mathcal{C}})$, is a set $\mathcal{C} = \{s_1, s_2, \ldots\}$ of strings referred to as systems, over some alphabet $\Sigma$, together with a polynomial space bounded nondeterministic Turing machine $\mathcal{M}_{\mathcal{C}}$ which has a distinguished "signal" state $q^\star$ such that

- for any $s \in \mathcal{C}$, $\mathcal{M}_{\mathcal{C}}$ has a unique configuration [1] $c_s$, such that any computation on $s$ reaches $c_s$ and $c_s$ is the first configuration along the computation whose machine state is $q^\star$. Intuitively, $c_s$ is the initial state of the system $s$.
- for any string $s \notin \mathcal{C}$, $\mathcal{M}_{\mathcal{C}}$ can never enter the state $q^\star$ for any computation on input $s$. Hence, $\mathcal{M}_{\mathcal{C}}$ implicitly specifies $\{s_1, s_2, \ldots\}$.

□

In the following we assume a fixed class of compact systems $(\mathcal{C}, \mathcal{M}_{\mathcal{C}})$.

[1] A configuration of $\mathcal{M}_{\mathcal{C}}$ consists of the contents of the tape, a machine state, and a position on the tape.

**Fact 32** *Since $\mathcal{M}_{\mathcal{C}}$ is polynomial space bounded on inputs of length $n$, say by the polynomial $q'(n)$, $\mathcal{M}_{\mathcal{C}}$ has on any input $s$ at most exponentially many reachable configurations whose machine state is $q^\star$; there exists a $B > 0$ and a polynomial $q(n)$, independent of $s$, such that given $s$ there are at most $B^{q(|s|)}$ possible configurations.*

Call the configurations whose machine state is $q^\star$ *signal configurations*, and let $Sig_{\mathcal{M}_{\mathcal{C}}}(s) = \{c \mid c \text{ is a reachable signal configuration of } \mathcal{M}_{\mathcal{C}} \text{ on input } s\}$. We can now define the state space associated to $s \in \mathcal{C}$.

**Definition 33** For $s \in \mathcal{C}$, let $(V_s, E_s, i_s)$ be the rooted graph whose nodes $V_s$ are $Sig_{\mathcal{M}_{\mathcal{C}}}(s)$, the signal configurations of $\mathcal{M}_{\mathcal{C}}$ on input $s$, whose edges are pairs of nodes $(c, c')$ such that $(c, c') \in E_s$ if and only if $\mathcal{M}_{\mathcal{C}}$ can reach $c'$ from $c$ without entering any other signal configurations, and whose initial/root node $i_s$ is the unique configuration $c_s$. □

*Remark.* We refer to $(V_s, E_s, i_s)$ as the *state space of the system* $s$, whenever $s \in \mathcal{C}$. Also, the nodes are referred to as states of $s$ and will be ranged over by $v, w, \ldots$. Notice that any system $s$ has at most $B^{q(|s|)}$ states. Whenever $(V_s, E_s, i_s)$ is understood from the context, we use the notation $v_0 \longrightarrow v_1 \longrightarrow \cdots \longrightarrow v_n$ instead of $(v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n) \in E_s$. For a state $v$ of $s$, we use the notation $v \not\rightarrow$ to indicate that there exists no state $v' \in V_s$ such that $v \longrightarrow v'$.

A run (initial run) of the system $s$ is any sequence $v_0 \longrightarrow v_1 \longrightarrow \cdots$ that is either infinite or ends in a state $v_n$ such that $v_n \not\rightarrow$ (and $v_0 = i_s$). The length of a finite run $v_0 \longrightarrow \cdots \longrightarrow v_n$ is $n$. We use Greek letters $\sigma, \gamma, \ldots$ to denote runs of the system $s$.

*Example.* As an example, let us see how $K$-bounded Petri nets can be viewed as a class of compact systems.

Let $K \in \mathbb{N}$. A *$K$-bounded Petri net*, or just a *$K$-bounded net*, is a tuple $N = (P, T, F, M_{init})_K$ such that

- $P$ and $T$ are finite disjoint nonempty sets; their elements are called *places* and *transitions*, respectively.
- $F \subseteq (P \times T) \cup (T \times P)$; $F$ is called the *flow relation*.
- $M_{init}\colon P \to \{0, 1, 2 \ldots, K\} \subseteq \mathbb{N}$; $M_{init}$ is called the *initial marking* of $N$; in general, a mapping $M\colon P \to \{0, 1, 2 \ldots, K\} \subseteq \mathbb{N}$ is called a *marking* of $N$. We shall use the notation $p \in M$ if $M(p) > 0$.

Next, we define the behaviour of $K$-bounded nets. The definitions are similar to that of Place/Transition nets, see Definition 1, except that places may only contain at most $K$ tokens.

- A transition $t \in T$ is *enabled* at a marking $M$ of $N$ if $M(p) > 0$ for every place $p$ in ${}^\bullet t = \{p \mid (p, t) \in F\}$, the preset of $t$, and $M(p) < K$ for every place $p$ in $t^\bullet \setminus {}^\bullet t$, where $t^\bullet = \{p \mid (t, p) \in F\}$, the postset of $t$. Henceforth, we shall assume that there are no isolated elements, i.e., $(\forall p \in P.\, {}^\bullet p^\bullet \neq \emptyset)$ and $(\forall t \in T.\, {}^\bullet t^\bullet \neq \emptyset)$.

- Given a transition $t$, we define a relation $\xrightarrow{t}$ between markings as follows: $M \xrightarrow{t} M'$ if $t$ is enabled at $M$ and $M'(s) = M(s) + F(t,s) - F(s,t)$, where $F(x,y)$ is 1 if $(x,y) \in F$ and 0 otherwise. The transition $t$ is said to *occur* (or *fire*) at $M$. A marking $M$ is a *deadlock*, denoted $M \not\rightarrow$, if it enables no transitions.

- If $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} M_n$ for some markings $M_0, M_1, \ldots M_n$, then the sequence $\sigma = t_1 \ldots t_n$ is called an *occurrence sequence* from $M_0$. $M_n$ is the marking *reached* by $\sigma$, and this is denoted $M_0 \xrightarrow{\sigma} M_n$. In general, we use the notation $\sigma$ for a finite or infinite sequence of transitions and use the notation $M_0 \xrightarrow{\sigma}$ to indicate that all finite prefixes of $\sigma$ are occurrence sequences from $M_0$.

- An occurrence sequence $\sigma$ from a marking $M_0$ is maximal if it is either infinite or it is finite and reaches a deadlock.

- A marking $M$ is *reachable* from $M_0$ if it is the marking reached by some occurrence sequence from $M_0$. $[M_0\rangle$ will denote the set of markings reachable from $M_0$. $[M_{init}\rangle$ is the set of reachable markings of $N$.

- The reachability graph of $N$ is the edge-labelled graph, $(V_N, E_N)$, whose vertices are the reachable markings of $N$; if $M \xrightarrow{t} M'$ for a reachable marking $M$, then there is an edge from $M$ to $M'$ labelled $t$. Notice that $N$ has at most $(K+1)^{|P|}$ markings.

The encoding of a $K$-bounded net can be done along the lines of, e.g., Chapter 8.3 in [HU79]. We choose the following encoding: The string $s_N$ encodes (in binary) the number of places, the number of transitions, the pairs in $F$, and the initial marking.

The machine $\mathcal{M}_\mathcal{C}$ will do the following: First, it checks that the input string encodes a $K$-bounded net. Assume that the net $N$ described by the input has $n$ places. Since $N$ is assumed to have no isolated elements, the length of the input is at least $n$ (due to the encoding of $F$). Hence, $\mathcal{M}_\mathcal{C}$ may then use $n\lceil \log K \rceil$ tape squares to store a marking (notice $K$ is fixed for the class of $K$-bounded nets). It starts by storing the initial marking of the net. It then enters its signal state to signal that the configuration is a marking of the net. Then, it leaves the signal state, guesses a transition, and checks if it is enabled. If it is, $\mathcal{M}_\mathcal{C}$ "fires" it by updating the stored marking accordingly. Having done that $\mathcal{M}_\mathcal{C}$ enters the signal state, signaling that it has computed a new marking of the net. Then it continues as before; leaving the signal state, guessing a new transition to fire etc. If the guessed transition is not enabled, $\mathcal{M}_\mathcal{C}$ just halts. Notice that if $s_N$ is the encoding of the $K$-bounded net $N$, then the state space of $s_N$ is isomorphic to the reachability graph of $N$ such that $i_{s_N}$ corresponds to the initial marking of $N$. Hence, $K$-bounded nets can be specified as a class of compact systems.

Henceforth, we assume a fixed class of compact systems $(\mathcal{C}, \mathcal{M}_\mathcal{C})$ and continue by giving the syntax of the temporal logics we are considering.

### 3.2.2 Temporal Logics

Let $\mathcal{A}$ be a set of *atomic propositions*. We assume it to be fixed in the following. We consider the temporal logics $\mathsf{L}(X,U,S)$ and $\mathsf{B}(X,U)$, both described in detail in [Eme90].

The formulas of the logic $\mathsf{L}(X,U,S)$ over $\mathcal{A}$ are defined inductively:

- $t$, $f$, or any $p \in \mathcal{A}$.
- $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $X(\varphi_1)$, $\varphi_1 U \varphi_2$, and $\varphi_1 S \varphi_2$ are formulas, where $\varphi_1$ and $\varphi_2$ are formulas.

$\mathsf{L}(X,U,S)$ is a linear time temporal logic, whose formulas are interpreted over runs of a systems $s$.

In order to interpret the formulas of $\mathsf{L}(X,U,S)$ over the runs of a system $s$ we need a valuation $\eta_s : \mathcal{A} \times Sig_{\mathcal{M}_C}(s) \longrightarrow \mathbf{2}$ which tells us if an atomic proposition holds at a state. Furthermore, we require that the valuation $\eta_s$ is computable by a polynomial space bounded deterministic Turing machine which we denote $T_{\eta_s}$. We assume the atomic propositions to be encoded as strings.

Given $s$, a run $\sigma = v_0 \longrightarrow v_1 \longrightarrow \cdots$ of $s$, a natural number $0 \leq i \leq |\sigma|$, and $\eta_s$. Relative to $\eta_s$ we interpret the formulas of $\mathsf{L}(X,U,S)$ at $(\sigma, i)$ as follows:

- $(\sigma,i) \models t$ and $(\sigma,i) \not\models f$.
- $(\sigma,i) \models p$ iff $\eta_s(p, v_i) = \mathbf{1}$.
- $(\sigma,i) \models \neg\varphi$ iff $(\sigma,i) \not\models \varphi$.
- $(\sigma,i) \models \varphi_1 \wedge \varphi_2$ iff $(\sigma,i) \models \varphi_1$ and $(\sigma,i) \models \varphi_2$.
- $(\sigma,i) \models X(\varphi)$ iff $i < |\sigma|$ and $(\sigma, i+1) \models \varphi$.
- $(\sigma,i) \models \varphi_1 U \varphi_2$ iff there exists a natural number $i \leq j \leq |\sigma|$ such that $(\sigma,j) \models \varphi_2$ and for all $i \leq k < j$, $(\sigma,k) \models \varphi_1$.
- $(\sigma,i) \models \varphi_1 S \varphi_2$ iff there exists a $0 \leq j \leq i$ such that $(\sigma,j) \models \varphi_2$ and for all $j < k \leq i$, $(\sigma,k) \models \varphi_1$.

The interpretation of the logic should be clear, except perhaps for $\varphi_1 U \varphi_2$ and $\varphi_1 S \varphi_2$. Intuitively the former expresses that $\varphi_2$ holds somewhere in the future and that $\varphi_1$ holds "until" then; the latter expresses that somewhere in the past $\varphi_2$ holds and "since" then, $\varphi_1$ holds. Remember that the "past" and "future" is relative to $\sigma$.

Next, we consider a well-known branching time temporal logic. The formulas of the logic $\mathsf{B}(X,U)$ over $\mathcal{A}$ (also known as CTL [CES86]) are also defined inductively:

- $t$, $f$, or any $p \in \mathcal{A}$.
- $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $EX(\varphi_1)$, $AX(\varphi_1)$, $E(\varphi_1 U \varphi_2)$, or $A(\varphi_1 U \varphi_2)$ are formulas, where $\varphi_1$ and $\varphi_2$ are formulas.

$\mathsf{B}(X,U)$ is a branching time logic whose formulas are interpreted at state of a system $s$.

Given a system $s$, a state $v$ of $s$, and a valuation $\eta_s$. Then, the formulas of $\mathsf{B}(X,U)$ are interpreted relative to $\eta_s$ as follows:

- $v \models t$ and $v \not\models f$.
- $v \models p$ iff $\eta_s(p,v) = \mathbf{1}$.
- $v \models \neg\varphi$ iff $v \not\models \varphi$.
- $v \models \varphi_1 \wedge \varphi_2$ iff $v \models \varphi_1$ and $v \models \varphi_2$.
- $v \models EX(\varphi)$ iff there exists $v \longrightarrow v'$ such that $v' \models \varphi$.
- $v \models AX(\varphi)$ iff for all $v \longrightarrow v'$, $v' \models \varphi$.
- $v \models E(\varphi_1 U \varphi_2)$ iff there exists $v \longrightarrow v_1 \longrightarrow \cdots \longrightarrow v_n$ such that $v_n \models \varphi_2$ and for all $0 \leq j < n$, $v_j \models \varphi_1$, where $v_0 = v$.
- $v \models A(\varphi_1 U \varphi_2)$ iff for all $v \longrightarrow v_1 \longrightarrow \cdots$ there exists an $n$ such that $v_n \models \varphi_2$ and for all $0 \leq j < n$, $v_j \models \varphi_1$, where $v_0 = v$.

The interpretation of the temporal formulas shows the branching nature of $\mathsf{B}(X,U)$. At a state several possible successor states or paths have to be taken into account. The intuition behind the interpretation of the "until" formulas corresponds well to that of $\mathsf{L}(X,U,S)$ except that we quantify existentially or universally over paths from the state $v$.

### 3.2.3 Model-Checking Problems

**Definition 34** An instance of the *model-checking problem for* $\mathsf{L}(X,U,S)$ is a tuple $(s, T_{\eta_s}, \varphi)$, where $s \in \mathcal{C}$, $\eta_s$ is a valuation, and $\varphi$ is a $\mathsf{L}(X,U,S)$ formula. The model-checking problem for $(s, T_{\eta_s}, \varphi)$ is to decide whether or not there exists an initial run $\sigma$ of the system $s$ such that $(\sigma, 0) \models \varphi$. □

*Remark.* Notice that this definition is not the standard definition, which required $(\sigma, 0) \models \varphi$ of *all* initial runs $\sigma$ of the system. However, this definition is equivalent to ours, since the logic $\mathsf{L}(X,U,S)$ contains negation.

**Definition 35** An instance of the *model-checking problem for* $\mathsf{B}(X,U)$ is a tuple $(s, T_{\eta_s}, \varphi)$, where $s \in \mathcal{C}$, $\eta_s$ is a valuation, and $\varphi$ is a $\mathsf{B}(X,U)$ formula. The model-checking problem for $(s, T_{\eta_s}, \varphi)$ is to decide whether or not $i_s \models \varphi$. □

*Remark.* Notice that the valuation is usually implicitly assumed part of the problem instances to the model-checking problem [SC85] or assume to be easily computable [CES86]. We could also have defined a valuation to be relative to $(\mathcal{C}, \mathcal{M}_{\mathcal{C}})$ and $\mathcal{A}$. This wouldn't lead to any significant changes of the results.

*Example.* Interpreting the atomic propositions $\mathcal{A}$ as places of $K$-bounded nets gives us a valuation $\eta$ for all nets. $\eta$ maps a pair consisting of an atomic proposition $a$ and a place $p$ to $\mathbf{1}$ if and only if the encoding of $a$ equals the encoding of $p$. We therefore only need one $T_\eta$ for the class of $K$-bounded nets. Having one for each $N$ would also be possible.

## 3.3 PSPACE Upper Bounds

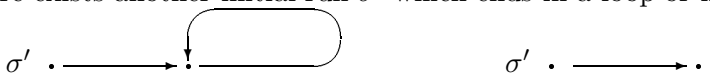
In this section we provide PSPACE upper bounds for the model-checking problems defined in Definition 34 and Definition 35.

### 3.3.1 Linear Time

The result of this section is based on the idea behind the decision procedure for the logic $\mathsf{L}(X, U, S)$ given in [SC85]. There, Sistla and Clarke also reduce the problem of determining truth in an R-structure (model-checking problem over Kripke structures) to the satisfiability problem (PSPACE-complete) by encoding an R-structure into a formula. Since a compact system may have exponentially many states, encoding the state space (Kripke structure) of a system in a formula would yield an exponentially long formula.

Instead, one could try to encode the system itself in the logic. This is easily done for systems like 1-safe Petri nets. However, when considering other models like $K$-bounded Petri nets, this encoding quickly becomes more troublesome. One of the reasons why we have chosen the setting of $(\mathcal{C}, \mathcal{M}_\mathcal{C})$ is that describing a class of systems as a class of compact systems is often more straightforward than exhibiting such encodings.

Our solution is based on the following observation. If $(\sigma, 0) \models \varphi$ for an initial run $\sigma$, then there exists another initial run $\sigma'$ which ends in a loop or in a dead state $v' \not\rightarrow$,




such that $(\sigma', 0) \models \varphi$. Moreover, we can find bounds on the lengths of the paths.

**Definition 36** Given a system $s$, a run $\sigma = v_0 \longrightarrow v_1 \longrightarrow \cdots$ of $s$, a formula $\varphi$ of $\mathsf{L}(X, U, S)$, and a valuation $\eta_s$. Then, $Sub(\sigma, i, \varphi)$ is the set of subformulas $\varphi'$ of $\varphi$ such that $(\sigma, i) \models \varphi'$. □

**Lemma 37** *Given a system $s$, a run $\sigma = v_0 \longrightarrow v_1 \longrightarrow \cdots$ of $s$, a formula $\varphi$ of $\mathsf{L}(X, U, S)$, and a valuation $\eta_s$. If $i < j \leq |\sigma|$, $Sub(\sigma, i, \varphi) = Sub(\sigma, j, \varphi)$, and $v_i = v_j$, then*

$$\forall 0 \leq l \leq i.\, Sub(\sigma, l, \varphi) = Sub(\sigma', l, \varphi)$$

$$\forall j \leq l \leq |\sigma|.\, Sub(\sigma, l, \varphi) = Sub(\sigma', l - (j - i), \varphi)\ .$$

*where all indices range over natural numbers and $\sigma' = v'_0 \longrightarrow v'_1 \longrightarrow \cdots = v_0 \longrightarrow \cdots \longrightarrow v_i \longrightarrow v_{j+1} \longrightarrow v_{j+2} \longrightarrow \cdots$.*

**Proof.** Induction in $\varphi$. ∎

Given a run $\sigma = v_0 \longrightarrow v_1 \longrightarrow \cdots$ of a system $s$, two indices $i$ and $j$, and a formula $\varphi$ of $\mathsf{L}(X, U, S)$. $\varphi$ is said to be fulfilled between $i$ and $j$ if and only if $i < j$ and there exists a $k$, $i \leq k < j$, such that $(\sigma, k) \models \varphi$.

The next lemma states that from a run $\sigma$ which has two identical states satisfying the same subformulas of $\varphi$ and in between which all "until" formulas are fulfilled, one can obtain a new run $\sigma'$, consisting of a prefix and a period which is a loop, such that it essentially satisfies the same subformulas of $\varphi$ as $\sigma$.

**Lemma 38** *Given a system $s$, an infinite run $\sigma = v_0 \longrightarrow v_1 \longrightarrow \cdots$ of $s$, a formula $\varphi$ of $\mathsf{L}(X, U, S)$, an index $i$, and a natural number $p > 0$ (the period) such that $v_i = v_{i+p}$, $Sub(\sigma, i, \varphi) = Sub(\sigma, i+p, \varphi)$, and for every formula $\varphi_1 U \varphi_2$ in $Sub(\sigma, i, \varphi)$, $\varphi_2$ is fulfilled between $i$ and $i + p$. Let $\sigma' = v'_0 \longrightarrow v'_1 \longrightarrow \cdots$ be the run $v_0 \longrightarrow \ldots \longrightarrow v_i \longrightarrow \cdots \longrightarrow v_{i+p-1} \longrightarrow v_i \longrightarrow \cdots \longrightarrow v_{i+p-1} \longrightarrow v_i \longrightarrow \cdots$, i.e., the period $v_i \longrightarrow \cdots \longrightarrow v_{i+p}$ is repeated infinitely often. Then,*

$$\forall 0 \leq l \leq i + p.\ Sub(\sigma, l, \varphi) = Sub(\sigma', l, \varphi)$$

$$\forall i \leq l.\ Sub(\sigma', l, \varphi) = Sub(\sigma', (l + p), \varphi)\ .$$

**Proof.** Induction in $\varphi$, case based analysis. ∎

We continue with the theorem which our upper bound result is based upon. It is a variant of "Ultimately Periodic Model Theorem" [SC85].

**Theorem 39** *Given a system $s$, $\eta_s$ a valuation, $\sigma$ a run of $s$, and $\varphi$ a formula of $\mathsf{L}(X, U, S)$ such that $(\sigma, 0) \models \varphi$, then*

1) *if $|\sigma| < \infty$, then there exists a run $\gamma$ of $s$ such that $|\gamma| \leq B^{q(|s|)} 2^{|\varphi|}$ and $(\gamma, 0) \models \varphi$.*

2) *if $|\sigma| = \infty$, then there exists a run $\gamma = w_0 \longrightarrow w_1 \longrightarrow \cdots$ and indices $0 \leq i \leq B^{q(|s|)} 2^{|\varphi|}$, $0 < p \leq |\varphi| B^{q(|s|)} 2^{|\varphi|}$ such that $w_l = w_{l+p}$ for $l \geq i$ and $(\gamma, 0) \models \varphi$.*

*where $B$ and $q(n)$ were described in Sect. 3.2.1.*

**Proof.** We only consider the case where $\sigma$ is infinite. The other case can be handled similarly. So, assume $(\sigma, 0) \models \varphi$. Since $s$ has at most $B^{q(|s|)}$ states and $\varphi$ has at most $2^{|\varphi|}$ subformulas there must exist indices $i < j$ such that

i) $v_i = v_j$, $Sub(\sigma, i, \varphi) = Sub(\sigma, j, \varphi)$, and for all $\varphi_1 U \varphi_2 \in Sub(\sigma, i, \varphi)$, $\varphi_2$ is fulfilled between $i$ and $j$.

Now applying Lemma 37 repeatedly to the initial part of $\sigma$ we obtain a new run $\sigma'$ and indices $i' < j'$ such that

ii) $(\sigma', 0) \models \varphi$, $v'_{i'} = v'_{j'}$, $Sub(\sigma, i, \varphi) = Sub(\sigma', i', \varphi) = Sub(\sigma', j', \varphi)$, $i' \leq B^{q(|s|)}2^{|\varphi|}$, and for all $\varphi_1 U \varphi_2 \in Sub(\sigma', i', \varphi)$, $\varphi_2$ is fulfilled between $i'$ and $j'$.

Also, repeatedly applying Lemma 37 between $v'_{i'}$ and $v'_{j'}$ we can assume that $(j' - i') \leq |\varphi| B^{q(|s|)}2^{|\varphi|}$; if at any time during the application of Lemma 37 the current value of $(j' - i')$ is larger than $|\varphi| B^{q(|s|)}2^{|\varphi|}$, there must exist more than $|\varphi|$ identical states $v'_l$ among $v'_{i'+1}, \ldots, v'_{j'}$ satisfying the same set $Sub(\sigma', l, \varphi)$ of formulas. Since $\varphi$ has less then $|\varphi|$ subformulas of the form $\varphi_1 U \varphi_2$, we can remove a loop between two of these states and still have all such $\varphi_2$'s fulfilled between $i'$ and (the new value of) $j'$ (in the resulting new run). We therefore conclude that there exists a run $\sigma''$ of $s$ and indices $i'' < j''$ such that

iii) $(\sigma'', 0) \models \varphi$, $v''_{i''} = v''_{j''}$, $Sub(\sigma'', i'', \varphi) = Sub(\sigma'', j'', \varphi)$, $i'' \leq B^{q(|s|)}2^{|\varphi|}$, $(j'' - i'') \leq |\varphi| B^{q(|s|)}2^{|\varphi|}$, and for all $\varphi_1 U \varphi_2 \in Sub(\sigma'', i'', \varphi)$, $\varphi_2$ is fulfilled between $i''$ and $j''$.

Using Lemma 38 we conclude that the run $\gamma = v''_0 \longrightarrow \cdots \longrightarrow v''_{i''} \longrightarrow \cdots \longrightarrow v''_{j''} \longrightarrow v''_{i''+1} \longrightarrow \cdots \longrightarrow v''_{j''} \longrightarrow \cdots$ of $s$ has the property $(\gamma, 0) \models \varphi$. ∎

**Theorem 40** *Fix any class of compact systems $(\mathcal{C}, \mathcal{M}_\mathcal{C})$ and set of atomic propositions $\mathcal{A}$. Then, the model-checking problem for $\mathsf{L}(X, U, S)$ is in PSPACE.*

**Proof.** We describe an algorithm in a Pascal like programming language which given any instance $(s, T_{\eta_s}, \varphi)$ of the model-checking problem for L(X,U,S) solves it using only an amount of space polynomial in the sum $|s| + |T_{\eta_s}| + |\varphi|$. Let $n$ denote this sum. Henceforth, whenever we write polynomial, we implicitly mean polynomial in $n$. The algorithm can be encoded as a nondeterministic polynomial space bounded Turing machine.

Notice that given possible configurations $v$ and $v'$ of $\mathcal{M}_\mathcal{C}$ run on input $s$, there are polynomial space bounded nondeterministic Turing machines which decide properties such as whether or not (1) $v \in V_s$, (2) $v \longrightarrow v'$ given $v \in V_s$, or (3) $v \not\longrightarrow$. Moreover, we can also compute $i_s$ given $s$ using only a polynomial amount of space. For example, let us consider the case where given $v \in V_s$ and a possible configuration $v'$ we have to decide if $v \longrightarrow v'$. Notice that $v'$ is reachable from $v$ if and only if it can be reached from $v$ in at most $B^{q(|s|)}$ computation steps of $\mathcal{M}_\mathcal{C}$ (simulated on input $s$). A nondeterministic Turing machine will simulate $\mathcal{M}_\mathcal{C}$ from $v$. It guesses a number $k_1 \leq B^{q(|s|)}$. Then, storing at most two configurations of $\mathcal{M}_\mathcal{C}$, it guesses, one step at the time, a computation of $\mathcal{M}_\mathcal{C}$ of length $k_1$. For each step it decrements $k_1$ and checks if $\mathcal{M}_\mathcal{C}$ can go from the old configuration to the new (guessed) configuration. Also, it checks that none of these intermediate configurations of $\mathcal{M}_\mathcal{C}$ are signal configurations. Finally, if it reaches $k_1 = 0$ it check that the guessed configuration equals $v'$ and is a signal configuration. It should be clear that this machine uses at most a polynomial amount of space. Now applying the technique from [HU79] Theorem 12.10, we obtain a deterministic polynomial space bounded Turing machine, since we can compute an exponential bound for the maximal number of configurations of the nondeterministic machine.

Hence, in the algorithm we shall refer freely to the states of $s$ and use the notation $v \longrightarrow v'$. Whenever a property is checked and is found not to hold, the algorithms fails. Let $Sub(\varphi)$ be the set of subformulas of $\varphi$, let $\mathcal{S}$ range over subsets of $Sub(\varphi)$, and let $Un(\varphi)$ be the set $\{\varphi_2 \mid \varphi_1 U \varphi_2 \in Sub(\varphi)\}$.

First, we consider the case where the answer to $(s, T_{\eta_s}, \varphi)$ is Yes, due to the existence of an infinite initial run satisfying $\varphi$.

Algorithm 1a

**01:** **Guess** $0 \leq n_1 \leq B^{q(|s|)}2^{|\varphi|}$
**02:** **Guess** $\mathcal{S}_{\mathbf{current}} \subseteq Sub(\varphi)$
**03:** **Let** $v_{\mathbf{current}} = i_s$
**04:** **Check boolean consistency of** $\mathcal{S}_{\mathbf{current}}$ **and** $v_{\mathbf{current}}$**, i.e., that**
**05:** $(\forall p \in Sub(\varphi).\ s \in \mathcal{S}_{\mathbf{current}} \Leftrightarrow \eta_s(p, v_{\mathbf{current}}))$
**06:** $(\forall \varphi_1 \wedge \varphi_2 \in Sub(\varphi).\ \varphi_1 \wedge \varphi_2 \in \mathcal{S}_{\mathbf{current}} \Leftrightarrow \varphi_1 \in \mathcal{S}_{\mathbf{current}}$ **and** $\varphi_2 \in \mathcal{S}_{\mathbf{current}})$
**07:** $(\forall \neg\varphi' \in Sub(\varphi).\ \neg\varphi' \in \mathcal{S}_{\mathbf{current}} \Leftrightarrow \varphi' \notin \mathcal{S}_{\mathbf{current}})$
**08:** **Check that** $\varphi \in \mathcal{S}_{\mathbf{current}}$
**09:** **Check that** $(\forall \varphi_1 S \varphi_2 \in Sub(\varphi).\ \varphi_2 \in \mathcal{S}_{\mathbf{current}})$
**10:** **Let** $count := 0$
**11:** **While** $count < n_1$ **do**
**12:** **Guess a configuration** $v_{\mathbf{next}}$ **and check that** $v_{\mathbf{current}} \longrightarrow v_{\mathbf{next}}$
**13:** **Guess** $\mathcal{S}_{\mathbf{next}} \subseteq Sub(\varphi)$
**14:** **Check boolean consistency of** $\mathcal{S}_{\mathbf{next}}$ **and** $v_{\mathbf{next}}$
**15:** **Check that**
**16:** $(\forall X\varphi' \in Sub(\varphi).\ X\varphi' \in \mathcal{S}_{\mathbf{current}} \Leftrightarrow \varphi' \in \mathcal{S}_{\mathbf{next}})$
**17:** $(\forall \varphi_1 S \varphi_2 \in Sub(\varphi).\ \varphi_1 S \varphi_2 \in \mathcal{S}_{\mathbf{next}} \Leftrightarrow$
$(\varphi_2 \in \mathcal{S}_{\mathbf{next}} \vee (\varphi_1 \in \mathcal{S}_{\mathbf{next}} \wedge \varphi_1 S \varphi_2 \in \mathcal{S}_{\mathbf{current}})))$
**18:** $(\forall \varphi_1 U \varphi_2 \in Sub(\varphi).\ \varphi_1 U \varphi_2 \in \mathcal{S}_{\mathbf{current}} \Leftrightarrow$
$(\varphi_2 \in \mathcal{S}_{\mathbf{current}} \vee (\varphi_1 \in \mathcal{S}_{\mathbf{current}} \wedge \varphi_1 U \varphi_2 \in \mathcal{S}_{\mathbf{next}})))$
**19:** **Let** $count := count + 1$
**20:** **Let** $v_{\mathbf{current}} = v_{\mathbf{next}}$
**21:** **Let** $\mathcal{S}_{\mathbf{current}} := \mathcal{S}_{\mathbf{next}}$
**22:** **Endwhile**
**23:** **Let** $v_{\mathbf{loop}} := v_{\mathbf{current}}$
**24:** **Let** $\mathcal{S}_{\mathbf{loop}} := \mathcal{S}_{\mathbf{current}}$
**25:** **Guess** $0 \leq n_1 \leq |\varphi| B^{q(|s|)}2^{|\varphi|}$
**26:** **Let** $count := 0$
**27:** **Let** $\mathcal{S}_U := \emptyset$
**28:** **While** $count < n_1$ **do**
**29:** $(*$ **Lines 29 to 38 are a copy of lines 12 to 21** $*)$
**39:** **Let** $\mathcal{S}_U := \mathcal{S}_U \cup (\mathcal{S}_{\mathbf{current}} \cap Un(\varphi))$
**40:** **Endwhile**
**41:** **Check that** $v_{\mathbf{current}} = v_{\mathbf{loop}}$

**42:** **Check that** $\mathcal{S}_{\mathbf{current}} = \mathcal{S}_{\mathbf{loop}}$
**43:** **Check that** $(\forall \varphi_1 U \varphi_2 \in \mathcal{S}_{\mathbf{loop}} . \varphi_2 \in \mathcal{S}_U)$
**44:** **Answer** Yes

For the case where the satisfying initial run is finite the following algorithm is derived:

Algorithm 1b

**01:** (∗ **Lines 02 to 23 are a copy of lines 01 to 22 of** Algorithm 1a ∗)
**24:** **Check that** $v_{\mathbf{current}} \not\rightarrow$
**25:** **Check that** $(\forall \varphi_1 U \varphi_2 \in \mathcal{S}_{\mathbf{current}} . \varphi_2 \in \mathcal{S}_{\mathbf{current}})$
**26:** **Answer** Yes

Our final algorithm chooses nondeterministically between Algorithm 1a and Algorithm 1b. The correctness of the algorithm is straightforward to establish; if the algorithm answers Yes, examine the the values of $\mathcal{S}_{\text{current}}$, $\mathcal{S}_U$, and the guessed run of $s$. Let $\sigma' = v'_1 \longrightarrow v'_2 \longrightarrow \ldots$ denote the (induced) guessed run. Then, by induction in $\varphi' \in \mathit{Sub}(\varphi)$ show that $(\sigma', j) \models \varphi' \Leftrightarrow \varphi' \in \mathcal{S}_{\text{current}}$ for any $j$ and $\mathcal{S}_{\text{current}}$ corresponding to $v'_j$. Conclude that this run indeed shows that the answer to $(s, T_{\eta_s}, \varphi)$ is Yes ; conversely, if the answer to $(s, T_{\eta_s}, \varphi)$ is Yes, then by Theorem 39 there exists a run of the algorithm which answers Yes.

It should be clear from the assumptions about $\mathcal{M}_{\mathcal{C}}$, $T_{\eta_s}$, and the above comments about deciding properties about the nodes and edges of $(V_s, E_s)$ using only a polynomial amount of space, that this algorithm can be implemented by a nondeterministic polynomial space bounded Turing machine. By Savitch's Theorem (NPSPACE = PSPACE) we conclude that the model-checking problem for $\mathsf{L}(X, U, S)$ is in PSPACE. ■

### 3.3.2 Branching Time

In this section we describe an algorithm which solves the model-checking problem for $\mathsf{B}(X, U)$ using only a polynomial amount of space.

Our solution use Turing machines and results about the complexity of constructing and transforming such machines.

The idea behind our construction is to construct a $C$-tape Turing machine $\mathcal{M}$, for some constant $C$. Denote these tapes by $T_1, \ldots, T_C$. Given a problem instance as input, $\mathcal{M}$ will, in polynomial time, construct a (description of a) deterministic polynomial space bounded machine $M_{\varphi'}$ for each occurrence of a subformula [2] $\varphi'$ of $\varphi$. Let us call these machines *subformula machines*.

The subformula machines have the extra ability that they may call certain subformula machines as subroutines. A machine $M_{\varphi'}$ will call as subroutines the subformula machines corresponding to the immediate subformulas of $\varphi'$; e.g., $M_{\varphi_1 U \varphi_2}$ will call $M_{\varphi_1}$

[2] For convenience we shall use the same notation for both subformulas of and occurrences of subformulas of $\varphi$.

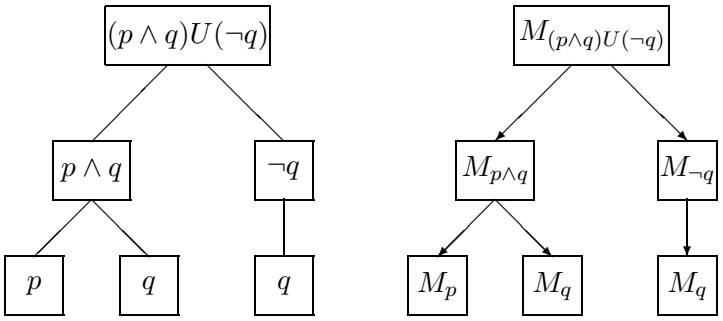



Figure 3.1: Formula $(p \wedge q)U(\neg q)$ and the corresponding subformula machines.

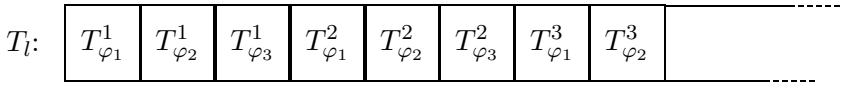



Figure 3.2: $T_l$ simulates 3 tapes: $T_{\varphi_1}$, $T_{\varphi_2}$, and $T_{\varphi_3}$.

and $M_{\varphi_2}$ as subroutines. These calls may be considered as single computation steps from the calling machines point of view much in the same way as Turing machines use oracles. $\mathcal{M}$ will simulate all of the subformula machines and their subroutine calls. Notice that $\mathcal{M}$ will need a stack of depth at most $|\varphi|$ to simulate the subroutine call sequence. Figure 3.1 illustrates this, where the arrows indicate which machines may be used as subroutines.

The subformula machines will be constructed in an bottom up fashion, i.e., when constructing $M_{\varphi'}$, all subformula machines corresponding to the proper subformulas of $\varphi'$ have been constructed. The subformula machines will be stored on one of $\mathcal{M}$'s tape using only a polynomial amount of tape. Each of these machines will compute the function $f_{\varphi'} : V_s \longrightarrow \{\mathbf{0},\mathbf{1}\}$, defined by $f_{\varphi'}(v) = \mathbf{1}$ if and only if $v \models \varphi'$. Since there are at most $|\varphi|$ subformula occurrences in $\varphi$, say $m$, one of $\mathcal{M}$'s $C$ tapes, say $T_l$, will be used to simulate tapes for each of the subformula machines; assuming we have the $m$ machines enumerated, the $j$th machine tape will consist of all of $T_l$'s cells whose index $i$ equals $j$ modulo $m$. Figure 3.2 illustrates this for the case where there are three subformula occurrences. The $j$th cell on a simulated tape is indicated by superscript $j$.

Having constructed all the machines $M_{\varphi'}$, $\mathcal{M}$ then proceeds by giving $M_\varphi$ the input $i_s$ and starts simulating $M_\varphi$.

**Theorem 41** *Fix any class of compact systems $(\mathcal{C}, \mathcal{M}_\mathcal{C})$ and set of atomic propositions $\mathcal{A}$. Then, the model-checking problem for $\mathsf{B}(X,U)$ is in PSPACE.*

**Proof.** Given $(s, T_{\eta_s}, \varphi)$. Let $n$ denote the size of this problem instance. The machine $\mathcal{M}$ (informally described above) is defined as follows:

- $\mathcal{M}$ stores its input $(s, \eta_s, \varphi)$ on tape $T_1$.
- It then enumerates all subformulas occurrences of $\varphi$ and remembers how many there are. This is done on $T_2$.

*Remark.* Having found the number of subformula occurrences $\mathcal{M}$ will use $T_3$ to simulate one tape $T_{\varphi'}$ for each subformula occurrence $\varphi'$ as indicated in Figure 3.2.

- $\mathcal{M}$ now proceeds to construct the deterministic subformula machines as follows, starting with the smallest subformula occurrences. For a subformula occurrence $\varphi'$:
  - If $\varphi'$ is an atomic proposition $p$, then $M_{\varphi'}$ is the machine which given $v$ uses $T_{\eta_s}$ to compute $\eta_s(p, v)$.
  - If $\varphi'$ is of the form $\neg\varphi''$, then $M_{\varphi'}$ is the machine which given $v$ calls $M_{\varphi''}$ as subroutine with $v$ as parameter and returns the negated value obtained from $M_{\varphi''}$.
  - If $\varphi'$ is of the form $\varphi_1 \wedge \varphi_2$, then $M_{\varphi'}$ is the machine which given $v$ first calls $M_{\varphi_1}$ and then $M_{\varphi_2}$, remembering the values returned. It then returns **1** if and only if both values were **1**.
  - If $\varphi'$ is of the form $EX(\varphi'')$, then $\mathcal{M}$ first constructs a nondeterministic machine $M^n_{\varphi'}$ which does the following:

    Given $v$ it guesses $v'$, a possible configuration of $s$. Then it checks that $v \longrightarrow v'$. Finally it calls $M_{\varphi''}$ with parameter $v'$. If the returned value is **1** it returns **1**, else it returns **0**.

    Then, $\mathcal{M}$ constructs a deterministic version of this machine [3] using, e.g., a construction similar to the one sketched in [HU79] Theorem 12.10. This can be done since a (fully space constructible) polynomial bound on the space consumption of the nondeterministic machine can be computed. This machine will systematically examine all successor states of $v$ and check if $\varphi''$ holds at one of these states; in fact, this deterministic verion can be constructed such that if $M^n_{\varphi'}$ halts on one of these states as input, then **1** is returned; if no, **0** is returned. This is the machine $M_{\varphi'}$.

    In the rest of the proof we shall implicitly be using this type of deterministic constructions to obtain machines which always returns **1** or **0**.
  - If $\varphi'$ is of the form $AX(\varphi'')$, then $\mathcal{M}$ first constructs a nondeterministic machine which does the following:

    Given $v$ it guesses $v'$, a possible configuration of $s$. Then it check that $v \longrightarrow v'$. Finally it calls $M_{\varphi''}$ with parameter $v'$. If the returned value is **0** it returns **0**, else it returns **1**.

    Then, $\mathcal{M}$ constructs a deterministic version of this machine. This is the machine $M_{\varphi'}$.

[3] or actually, a slightly modified version of $M^n_{\varphi'}$ which halts when $M^n_{\varphi'}$ would have returned **1** and loops when $M^n_{\varphi'}$ would have ouput **0**

– If $\varphi'$ is of the form $E(\varphi_1 U \varphi_2)$, then $\mathcal{M}$ first constructs a nondeterministic machine which does the following, given $v$:

**Guess a number** $0 \leq k \leq B^{q(|s|)}$
**Let** $v_{\textbf{current}} := v$
**While** $k > 0$ **do**
    **Call** $M_{\varphi_1}$ **to check that** $v_{\textbf{current}} \models \varphi_1$
    **Guess** $v'$**, a possible state of** $s$
    **Check that** $v_{\textbf{current}} \longrightarrow v'$
    **Let** $v_{\textbf{current}} := v'$
    **Decrement** $k$ **by 1**
**Endwhile**
**Call** $M_{\varphi_2}$ **to check that** $v_{\textbf{current}} \models \varphi_2$

If the machine reaches the last test and it is successful, the machine returns **1**, else it returns **0**.

Then, $\mathcal{M}$ constructs a deterministic version of this machine that returns **1** if some run of the nondeterministic machine returns **1**, and **0** else. This is the machine $M_{\varphi'}$.

– If $\varphi'$ is of the form $A(\varphi_1 U \varphi_2)$, then $\mathcal{M}$ first constructs a nondeterministic machine which does the following, given $v$:

It nondeterministically chooses to execute one of the following (nondeterministic) programs:

**a)** **Guess a number** $0 \leq k \leq B^{q(|s|)}$
**Let** $v_{\mathbf{current}} := v$
**While** $k > 0$ **do**
**Call** $M_{\varphi_1}$ **to check that** $v_{\mathbf{current}} \models \varphi_1$
**Call** $M_{\varphi_2}$ **to check that** $v_{\mathbf{current}} \not\models \varphi_2$
**Guess** $v'$**, a possible state of** $s$
**Check that** $v_{\mathbf{current}} \longrightarrow v'$
**Let** $v_{\mathbf{current}} := v'$
**Decrement** $k$ **by 1**
**Endwhile**
**Check that either**
$v_{\mathbf{current}} \not\rightarrow$ **and**
$v_{\mathbf{current}} \models \varphi_1$ **(by calling** $M_{\varphi_1}$**) and**
$v_{\mathbf{current}} \not\models \varphi_2$ **(by calling** $M_{\varphi_2}$**)**
**or**
$v_{\mathbf{current}} \not\models \varphi_1$ **and**
$v_{\mathbf{current}} \not\models \varphi_2$

If the machine reaches the last test and it was successful, the machine returns **0**, else it returns **1**.

**b)** **Guess a number** $0 \leq k \leq 2B^{q(|s|)}$
**Guess a possible configuration** $v_{\mathbf{loop}}$
**Let** $b := \mathbf{0}$
**Let** $v_{\mathbf{current}} := v$
**While** $k > 0$ **do**
**If** $v_{\mathbf{current}} = v_{\mathbf{loop}}$ **then let** $b := \mathbf{1}$
**Call** $M_{\varphi_1}$ **to check that** $v_{\mathbf{current}} \models \varphi_1$
**Call** $M_{\varphi_2}$ **to check that** $v_{\mathbf{current}} \not\models \varphi_2$
**Guess** $v'$**, a possible state of** $s$
**Check that** $v_{\mathbf{current}} \longrightarrow v'$
**Let** $v_{\mathbf{current}} := v'$
**Decrement** $k$ **by 1**
**Endwhile**
**Check that** $v_{\mathbf{current}} = v_{\mathbf{loop}}$ **and that** $b = \mathbf{1}$

If the machine reaches the last test and it was successful, the machine returns **0**, else it returns **1**.

Then, $\mathcal{M}$ constructs a deterministic version of this machine that returns **0** if some run of the nondeterministic machine returns **0**, else **1**. This is the

machine $M_{\varphi'}$.

*Remark.* Notice that all nondeterministic machines above always answer either **1** or **0**, as do their deterministic versions. The algorithm for constructing the subformula machines is based on the observation that 1) $v \models E(\varphi_1 U \varphi_2)$ if and only if there exists a path $v_0 \longrightarrow v_1 \longrightarrow \cdots \longrightarrow v_k$ starting at $v$ such that $(\forall 0 \leq i < k.\, v_i \models \varphi_1)$ and $v_k \models \varphi_2$, and $0 \leq k \leq B^{q(|s|)}$; and 2) $v \not\models A(\varphi_1 U \varphi_2)$ if and only if 2a) there exists a path $v_0 \longrightarrow v_1 \longrightarrow \cdots \longrightarrow v_k$ starting at $v$ such that $(\forall 0 \leq i < k.\, v_i \models \varphi_1 \wedge \neg\varphi_2)$, $0 \leq k \leq B^{q(|s|)}$, and either $(v_k \models \neg\varphi_1 \wedge \neg\varphi_2)$ or $(v_k \not\rightarrow \wedge\, v_k \models \varphi_1 \wedge \neg\varphi_2)$; or 2b) there exists paths of the form $v_0 \longrightarrow \cdots \longrightarrow v_{k_1}$ and $v'_0 \longrightarrow \cdots \longrightarrow v'_{k_2}$ such that $v_0 = v$, $v_{k_1} = v'_0 = v'_{k_2}$, $0 \leq k_1, k_2 \leq B^{q(|s|)}$, and $(\forall 0 \leq j \leq k_1.\, v_j \models \varphi_1 \wedge \neg\varphi_2) \wedge (\forall 0 \leq j \leq k_2.\, v'_j \models \varphi_1 \wedge \neg\varphi_2)$.

By an induction argument one can show that the machine $M_{\varphi'}$ indeed computes $f_{\varphi'}$.

- Having constructed all subformula machines, $\mathcal{M}$ starts simulating a call of $M_\varphi$ with argument $v$. This simulation will use at most polynomial space and the answer returned by $M_\varphi$ is **1** if and only if $i_s \models \varphi$. So, $\mathcal{M}$ solves the model-checking problem.

It remains to argue for the complexity of $\mathcal{M}$. Any states of $s$ can be stored in polynomial space. $\mathcal{M}_\mathcal{C}$ and $T_{\eta_s}$ are polynomial bounded space machines which can be simulated by $\mathcal{M}$ in polynomial space. Properties about $(V_s, E_s)$ can be decided in polynomial space, see the proof of Theorem 40. The subformula machines can be constructed in polynomial time and each of them use at most polynomial space. When simulating the subformula machines, $\mathcal{M}$ only needs a stack of depth at most $|\varphi|$, where for each waiting subroutine call, $\mathcal{M}$ only needs a polynomial amount of space. We therefore conclude that $\mathcal{M}$ solves the model-checking problem for $\mathsf{B}(X, U)$ using only polynomial space. ■

## 3.4 Example of an Application

In this section we continue by showing how our results can be applied to $K$-bounded nets.

**Definition 42** An instance of the *model-checking problem* for $K$-bounded nets and $\mathsf{L}(F)$ ($\mathsf{L}(X, U, S)$, $\mathsf{B}(X, U)$ ) is a tuple $(N, \varphi)$, where $N$ is a $K$-bounded net and $\varphi$ is a $\mathsf{L}(F)$ ($\mathsf{L}(X, U, S)$, $\mathsf{B}(X, U)$ ) formula. The model-checking problem for $(N, \varphi)$ ($\varphi \in \mathsf{L}(F)$) is to decide whether or not there exists a maximal occurrence sequence $\sigma$ from $M_{init}$ such that $\sigma \models \varphi$. The model-checking problem for $(N, \varphi)$ ($\varphi \in \mathsf{L}(X, U, S)$ or $\varphi \in \mathsf{B}(X, U)$ ) is to decide whether or not $M_{init} \models \varphi$. □

*Remark.* Without loss of generality we have assumed that the set of atomic propositions is the set of places of $K$-bounded nets. An atomic proposition $p$ will hold at a marking $M$ if and only if $p \in M$. Also, we have used the symbol $\models$ for the obvious analogs of the satisfiability relations given i Sect. 3.2.2.

**Theorem 43** *The model-checking problems for $K$-bounded nets and the logics $\mathsf{L}(F)$, $\mathsf{L}(X,U,S)$, and $\mathsf{B}(X,U)$ are PSPACE-hard.*

**Proof.** Sketch: The PSPACE-hardness of the problem follows from the fact that the logics can express the reachability of a marking $M$ in a 1-safe net [CEP93] (A 1-safe net has the property that it is $K$-bounded for any $K \geq 1$). This can be done using an obvious formula linear in the size of $P$: $\varphi \equiv F((\bigwedge_{p \in M} p) \wedge (\bigwedge_{p \notin M} \neg p))$. Actually, along the lines in [CEP93] one can prove that for a given 1-safe net $N$ and a place $p$ of $N$, the problem of deciding whether or not there exists a reachable marking $M$ such that $p \in M$ $(M(p) = 1)$ is PSPACE-complete. Since this is expressed by the formula $F(p)$, model-checking any reasonable propositional branching time temporal logic must be PSPACE-hard. ∎

**Corollary 44** *The model-checking problems for $K$-bounded nets and the logics $\mathsf{L}(F)$, $\mathsf{L}(X,U,S)$, and $\mathsf{B}(X,U)$ are PSPACE-complete.*

**Proof.** Sketch: Notice that the size of an instance $(N, \varphi)$ is proportional to the size of the instance $(s_N, T_\eta, \varphi)$ and that the satisfiability relations $\models$ for $K$-bounded nets correspond to those for $K$-bounded nets as a class of compact systems given in Sect. 3.2.2. Since $\mathsf{L}(F)$ is a fragment of $\mathsf{L}(X,U,S)$ ($F(\varphi)$ is short for $t\,U\varphi$), Theorem 40 and Theorem 41 give us the matching PSPACE upper bounds. ∎

## 3.5 Summary

We have provided algorithms which give us an upper bound on the computational complexity of the model-checking problem for a well-known class of temporal logics interpreted over any class of compact systems, i.e., any class of descriptions of systems satisfying certain conditions which limit the "succinctness" of the description; the system's associated state graphs can be at most be exponentially larger than the descriptions themselves. Our results gave an upper bound for both $\mathsf{L}(X,U,S)$ and CTL. [4]

As an application of our results, we showed in Sect. 3.4 that the model-checking problems for $\mathsf{L}(\mathsf{F})$, $\mathsf{L}(X,U,S)$, and $\mathsf{B}(X,U)$ over $K$-bounded nets are PSPACE-complete.

Also, the presented approach is applicable if, e.g., compact systems are defined such that $\mathcal{M}_\mathcal{C}$ is exponential space bounded, i.e., the systems $s$ describe a double exponentially large state space. One would then get EXPSPACE upper bounds.

[4] In [BVW94, Kup95], a treatment of CTL$^*$ in the setting of concurrent programs has been presented, exhibiting a PSPACE upper bound.

Chapter 4

# Petri Nets, Traces, and Local Model Checking

Contents

## 4.1 Introduction

In the following we study the problem of model-checking a CTL-like logic, P-CTL, over models which incorporate a notion of fair progress. More specifically, we choose labelled 1-safe nets and incorporate a notion of fair progress using Mazurkiewicz trace-theory. In this framework, the maximal traces of a labelled 1-safe net capture exactly the fair progress of independent events. This model theoretic incorporation of progress is not reflected in the syntax of our logic P-CTL. In fact, P-CTL has almost the same syntax as CTL. The crucial difference is that we restrict the set of paths over which path quantified formulas are interpreted.

We give a set of sound and complete tableaux rules and a state labelling based model-checking algorithm. Both are a conservative extension of the work in [Lar88] and [CES86], respectively, in the sense that if we restrict attention to their choice of models, (labelled) transition systems, our rules and algorithm are equivalent.

In Sect. 4.2, we provide the necessary definitions. In Sect. 4.3, we present the logic and its interpretation. Section 4.4 contains a motivating example followed by the tableau rules and the definition of tableaux. In Sect. 4.4.3, we present the main result, soundness and completeness of the proposed tableau rules, and state the complexity of our model-checking problem. Finally, Sect. 4.7 contains the summary and suggestions for future work.

*Remark.* We define labelled 1-safe nets directly, instead of as a subclass of Place/ Transition nets. The firing rule is slightly different from that in Definition 1. As mention in the ending remark of Sect. 2.1, there are several definitions of 1-safe nets and, again, our results are independent of the definition used.

## 4.2 Definitions

### 4.2.1 Traces

In this section we recall some basic definitions. We start by defining *concurrent alphabets*, the fundamental structure in Mazurkiewicz trace theory [Maz86].

**Definition 45 Concurrent alphabet and traces**

- A *concurrent alphabet* $(A, I)$ consists of a finite set $A$ (the alphabet) and a symmetric and irreflexive relation $I \subseteq A \times A$—the independence relation.

In the following, assume a fixed concurrent alphabet $(A, I)$.

- Define $A^\infty = A^* \cup A^\omega$, i.e., $A^\infty$ is the set of all finite and infinite sequences of elements from $A$. Define concatenation $\circ$ of elements $u \in A^*$ and $v \in A^\infty$ as:

$$u \circ v = \begin{cases} u & if\ |u| = \omega \\ uv & else \end{cases}$$

For notational convenience we will write $uv$ instead of $u \circ v$.

- Let $\leq_{pref}$ be the usual prefix ordering on sequences and $\pi_{(a,b)}$ the projection on $\{a,b\}^\infty$. Define a preorder $\preceq$ on $A^\infty$ which requires the relative order of elements $a$ and $b$ which are in conflict, i.e., $(a,b) \notin I$, to be the same when ignoring other elements of the sequences. Formally:

$$u \preceq v \text{ if and only if } (\forall (a,b) \notin I.\, \pi_{(a,b)}(u) \leq_{pref} \pi_{(a,b)}(v))$$

- Define an equivalence relation $\equiv$ on $A^\infty$ by $u \equiv v$ if and only if $u \preceq v$ and $v \preceq u$. The elements of $A^\infty/\equiv$ are called *traces*. The equivalence class of $u$—the trace containing $u$—is denoted $[u]$.

- Fact: $\equiv$ is a congruence with respect to $\circ$.

- For $[u],[v] \in A^\infty/\equiv$ define $[u] \preceq [v]$ if and only if $u \preceq v$. It can be shown that $\preceq$ is a partial order over traces. We write $[u] \prec [v]$ if and only if $u \preceq v$ and $u \not\equiv v$.

- Fact: for $u, v \in A^*$:
  - $[u] \preceq [v]$ if and only if $(\exists u' \in A^*.\, [uu'] = [v])$
  - $u \equiv v$ if and only if $u \equiv_M v$, where $\equiv_M$ is the well known equivalence on finite sequences presented in [Maz86] to define finite traces.

□

*Example.* Consider the concurrent alphabet $(A, I)$, where $A = \{a,b,c\}$ and $I = \{(a,b)(b,a)\}$. Then, $abc \equiv bac$, $abc \not\equiv acb$, $(abbac)^\omega \equiv (aabbc)^\omega$, and $(abbac)^\omega \not\equiv (abcba)^\omega$.

*Remark.* We have chosen to present traces using projections $\pi_{(a,b)}$ because finite as well as infinite traces are handled in a uniform way. Similar definitions can be found in, e.g., [Kwi89].

### 4.2.2 Labelled 1-safe Nets

We continue by defining *labelled 1-safe nets*, the labelled version of 1-safe nets.

**Definition 46 1-safe nets**

A *1-safe net*, or just a *net*, is a structure $N = (P, T, F, M_0)$ such that

- $P$ and $T$ are finite nonempty 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*.

- $M_0 \subseteq P$; $M_0$ is called the *initial marking* of $N$; in general, a set $M \subseteq P$ is called a *marking* or a *state* of $N$.

Given $a \in P \cup T$, the *preset* of $a$, denoted ${}^\bullet a$, is defined as $\{a' \mid a'Fa\}$; the *postset* of $a$, denoted $a^\bullet$, is defined as $\{a' \mid aFa'\}$. The union of ${}^\bullet a$ and $a^\bullet$ is denoted ${}^\bullet a^\bullet$. The irreflexive symmetric *independence relation* $I$ over $T$ is defined by $t_1 I t_2$ if and only if ${}^\bullet t_1^\bullet \cap {}^\bullet t_2^\bullet = \emptyset$. Two transitions $t_1$ and $t_2$ are said to be *independent* if $t_1 I t_2$ and in conflict otherwise. Notice that $(T, I)$ is a concurrent alphabet. For $D \subseteq T$ and $t \in T$ we define $tID = DIt = \{t' \in D \mid t' I t\}$. □

Next, we give the definition of firing sequences.

**Definition 47 Firing sequences**

Let $N = (P, T, F, M_0)$ be a net.

- A transition $t \in T$ is *enabled* at a marking $M$ of $N$ if ${}^\bullet t \subseteq M$ and $t^\bullet \cap (M - {}^\bullet t) = \emptyset$. Denote the set of transitions enabled at a marking $M$ by $next(M)$.
- Given a transition $t$, define a relation $\stackrel{t}{\longrightarrow}$ between markings as follows: $M \stackrel{t}{\longrightarrow} M'$ if and only if $t$ is enabled at $M$ and $M' = (M - {}^\bullet t) \cup t^\bullet$. The transition $t$ is said to *occur* (or *fire*) at $M$. If $M_0 \stackrel{t_1}{\longrightarrow} M_1 \stackrel{t_2}{\longrightarrow} \cdots \stackrel{t_n}{\longrightarrow} M_n$ for some markings $M_1, M_2, \ldots M_n$, then the sequence $\sigma = t_1 \ldots t_n$ is called an *occurrence sequence*. $M_n$ is the marking *reached* by $\sigma$, and this is denoted $M_0 \stackrel{\sigma}{\longrightarrow} M_n$. A marking $M$ is *reachable* if it is the marking reached by some occurrence sequence. $M \not\longrightarrow$ denotes that there are no enabled transitions at $M$, i.e., $next(M) = \emptyset$, in which case it is said to be *dead* or be a *deadlock*.
- Given a marking $M$ of $N$, the set of reachable markings of $(P, T, F, M)$—the net obtained replacing the initial marking $M_0$ by $M$—is denoted by $[M\rangle$.
- A *labelled* 1-safe net $N = (P, T, F, M_0, l)$ is a 1-safe net extended with a labelling function $l : T \to Act$ mapping each transition to an action in some labelling set $Act$.

□

The behaviour of a net is captured by its reachability graph.

**Definition 48 Reachability graph**

The *reachability graph* of a net $N$ is the edge-labelled graph $(V, E)_N$, whose set of vertices (or states), $V$, is $[M_0\rangle$. The labelled edges are induced by the firing relations $\stackrel{t}{\longrightarrow}$, and hence conveniently denoted $M \stackrel{t}{\longrightarrow} M'$. □

### 4.2.3 Partial Order Semantics

In the following, we assume a fixed labelled 1-safe net $N$ and consider its reachability graph $(V, E)_N$. We use the symbols $p, q, \ldots$ to denote states in $(V, E)_N$. If nothing else is mentioned, it is implicitly assumed that $(T, I)$ is used to generate the congruence $\equiv$.

**Definition 49 Paths**

- Define a *path* from $p_0 \in V$ as a sequence, finite or infinite, of transitions $t_1, t_2, \ldots$, for which there exists states $p_1, p_2, \ldots$, such that $p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots$. Notice that the firing rules of the net ensure the uniqueness of the $p_i$'s, if they exist. We therefore refer to $p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots$ as a path from $p_0$ (also denoted $p_0 \xrightarrow{\sigma}$), where $\sigma = t_1 t_2 \cdots$. Define $path(p_0) \subseteq T^\infty$ to be all paths from $p_0$.
- Define $comp(p)$ as the maximal elements of $path(p)/\equiv$ with respect to $\preceq$. For $\sigma \in [\sigma'] \in comp(p)$ we refer to $p \xrightarrow{\sigma}$ as a *computation* from $p$.

□

Notice $path(p)$ is limit closed, i.e., if $\sigma_1\gamma_1, \sigma_1\sigma_2\gamma_2, \sigma_1\sigma_2\sigma_3\gamma_3, \ldots \in path(p)$, where all $\sigma_i$s are finite, then $\sigma_1\sigma_2\cdots \in path(p)$.

*Example.* In the process agent example from Figure 1.3 in Chap. 2, $\tau$ is cc-enabled along $i \xrightarrow{a^\infty}$, when we use $a$, $b$, and $\tau$ to refer to the corresponding transitions. Also, $\tau b a^\infty$ is a computation from $i$, while $a^\infty$ is not.

Due to the firing rules of nets, the congruence $\equiv$ respects the property of being a path.

**Lemma 50** *Given a net $N = (P, T, F, M_0)$, and a state $p$ of $(V, E)_N$. Then,*

$$(\forall \sigma \in path(p).\ (\forall \sigma' \in [\sigma].\ p \xrightarrow{\sigma'}))\ .$$

**Proof sketch.** If $\sigma$ is finite, the result easily follows from the commutativity of consecutive independent transitions. If $\sigma$ is infinite, notice that by interchanging a finite number of consecutive independent transitions of $\sigma$ we conclude that any finite prefix of $\sigma'$ is an element of $path(p)$. Since $path(p)$ is limit closed, we conclude $\sigma' \in path(p)$. ■

Hence, $path(p)$ can be partitioned into elements of $T^\infty/\equiv$. Moreover, if $\sigma$ is finite, then $p \xrightarrow{\sigma} q$ implies $(\forall \sigma' \in [\sigma].\ p \xrightarrow{\sigma'} q)$.

**Definition 51 Continuously Concurrently Enabled Transitions**

Given $\sigma \in path(p_0), |\sigma| = \omega, \sigma = t_1 t_2 \cdots$. A transition $t$ is said to be *continuously concurrently enabled* (cc-enabled) along $p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots$ if and only if $t$ is enabled at some $p_i$, $i \geq 0$, and independent of the remaining transitions of $\sigma$ from that state. Formally: $(\exists n \in I\!N.\ (\forall j \geq n.\ p_j \xrightarrow{t} \ \wedge\ t I t_{j+1}))$. Notice that the irreflexivity of $I$ implies that $n$, $t \neq t_j$, $j \geq n$. Whenever $p_0$ is clear from the context, $t$ is said to be cc-enabled along $\sigma$. □

The next two lemmas state properties of traces.

**Lemma 52** *Given a concurrent alphabet $(A, I)$. For $\sigma, \sigma' \in A^\infty$ we have that*

$$\sigma \preceq \sigma' \Leftrightarrow (\forall \sigma_1 \in \mathit{pref}_{fin}(\sigma).\, (\exists \sigma_1' \in \mathit{pref}_{fin}(\sigma').\sigma_1 \preceq \sigma_1')) \ .$$

**Proof.**
The "if" direction is proved by an easy contradiction argument. For the "only if" direction, first choose a finite prefix $\sigma_1'$ of $\sigma'$ such that its Parikh vector (for each $a \in A$ this vector provides the number of occurrences of $a$'s in $\sigma_1'$) is greater than or equal to that of $\sigma_1$. Assuming $\sigma_1 = a_1 \cdots a_n$ and $\sigma_1' = b_1 \cdots b_m$ find the first occurrence of $a_1$, say $b_{j_1}$, in $\sigma_1'$. Then for any $1 \leq j < j_1$ it must be the case that $b_j I b_{j_1}$, since we have $\sigma \preceq \sigma'$. Hence, $b_1 \cdots b_m \equiv b_{j_1} b_1 \cdots b_{j_1-1} b_{j_1+1} \cdots b_m$. Continuing this procedure for $a_2, \ldots, a_n$ we eventually get that $\sigma_1' \equiv \sigma_1 \gamma$ for some $\gamma \in A^*$. But then clearly $\sigma_1 \preceq \sigma_1'$. ∎

**Lemma 53** *Given a net $N = (P, T, F, M_0)$, a state $p$ of $(V, E)_N$, $\sigma \in \mathit{path}(p)$ such that $|\sigma| = \omega$, and $t \in T$ that is cc-enabled along $\sigma$. Then, for any $\sigma' \in [\sigma]$, $t$ is cc-enabled along $\sigma'$), i.e., $\equiv$ respects cc-enabledness.*

**Proof.** Clearly, by definition there exists a finite $\sigma_1 \in \mathit{path}(p)$, a $p' \in S$, and a $\sigma_2 \in \mathit{path}(p')$ such that $p \xrightarrow{\sigma} = p \xrightarrow{\sigma_1} p' \xrightarrow{\sigma_2}$ and $t$ is enabled at $p'$ and independent of all transitions in $\sigma_2$. Choose any $\sigma' \in [\sigma]$. Since $\sigma \preceq \sigma'$, if follows from Lemma 52 that there exists a finite prefix of $\sigma'$, say $\sigma_1'$, such that $\sigma_1 \preceq \sigma_1'$. Using the technique from the proof of Lemma 52 we see that there exists a $\gamma \in T^\omega$, such that $\sigma_1 \gamma \equiv \sigma_1'$ and all transitions in $\gamma$ are independent of $t$. We conclude that $t$ must be enabled at $p''$, where $p \xrightarrow{\sigma_1'} p''$, since $p \xrightarrow{\sigma_1 \gamma} p''$. Choosing $\sigma_2'$ such that $\sigma' = \sigma_1' \sigma_2'$ we also conclude that all transitions in $\sigma_2'$ are independent of $t$, since all transitions in $\sigma_2'$ occur in $\sigma_2$. Hence, $t$ is cc-enabled along $\sigma'$. ∎

Hence, based on Lemma 53 we may safely write $t \in T$ is cc-enabled along $[\sigma]$, meaning $t$ is cc-enabled along $\sigma \in \mathit{path}(p)$.

Next, we identify maximal traces as maximal elements in a partial order. The following lemma explains why we focus on these traces. They can be thought of representing executions of a concurrent system which are fair with respect to progress of independent processes. In [MOP89] the term "concurrency fairness" is used for such behaviours. Compared to other notions of "fairness" in the context of concurrent systems "progress fairness" is a weak assumption, see [MP92] for a comparison to other notions of fairness.

**Lemma 54** *Given a net $N = (P, T, F, M_0)$ and a state $p$ of $(V, E)_N$. For $[\sigma] \in \mathit{comp}(p)$ such that $|\sigma| = \omega$ we have*

$$(\exists\, [\sigma'] \in \mathit{comp}(p).[\sigma] \prec [\sigma']) \ \Leftrightarrow \ (\exists\, t \in T.\, t \textit{ is cc-enabled along } \sigma) \ .$$

**Proof.** The "if" direction is easy, and hence omitted. For the "only if" direction first observe the following: since $[\sigma] \prec [\sigma']$, there must exist a $t \in T$ such that $\pi_{(t,t)}(\sigma) < \pi_{(t,t)}(\sigma')$. Clearly, $|\pi_{(t,t)}(\sigma)| = n < \omega$ for some $n \in I\!N$. Let $\sigma = \sigma_1 \sigma_2$, where $\#_t(\sigma_1) =$

$n, \#_t(\sigma_2) = 0$, $|\sigma_1| < \omega$, and $\#_t(\sigma)$ is the number of $t$'s in $\sigma$. By Lemma 52 we know that there exists a finite prefix $\sigma_3$ of $\sigma'$ such that $[\sigma_1] \preceq [\sigma_3]$. Furthermore, there must exists a suffix of $\sigma$ such that all transitions of it are independent of $t$. To see this, assume that there were infinitely many indexes $i_j \in I\!N$ for $0 \leq j$ such that $(t_{i_j}, t) \notin I$, where $\sigma = t_1 t_2 \cdots$. Since $(\forall j \in I\!N.\ \pi_{(t,t_{i_j})}(\sigma) <_{pref} \pi_{(t,t_{i_j})}(\sigma'))$, all $t_{i_j}$'s must occur before the $(n+1)$'th $t$ in $\pi_{(t,t_{i_j})}(\sigma')$. But this clearly means that there must be infinitely many transitions between the $n$'th and $(n+1)$'th $t$ in $\sigma'$, which is impossible.

Next, we show that there must exist a transition $t'$ which is cc-enabled along $\sigma$. First, choose the first occurrence of a $t' \in T$ along $\sigma'$ such that $\#_{t'}(\sigma) < \#_{t'}(\sigma')$. Next, split $\sigma$ into $\sigma_1$ (finite) and $\sigma_2$ such that $\sigma = \sigma_1\sigma_2$ and all transitions in $\sigma_2$ are independent of $t'$. Then, choose $\sigma_1'$ as the shortest prefix of $\sigma'$ such that $\#_{t'}(\sigma_1') \geq \#_{t'}(\sigma_1) + 1$ and the Parikh vector of $\sigma_1'$ is greater than that of $\sigma_1$. By an argument similar to that above, one can rearrange $\sigma_1'$ by continuously interchanging adjacent independent transitions and obtain $\sigma_1' \equiv \sigma_1\gamma \in path(p)$. Now $\#_{t'}(\gamma) > 0$. Let $\gamma = t_1' \cdots t_r'\, t'\, t_1'' \cdots t_s''$, where $r, s \geq 0$ and all $t_i'$'s are different from $t'$. Now assume that $(\exists 1 \leq j \leq r.\ (t_j', t') \notin I)$. Choose the first such $j$. Then, $\pi_{(t_j',t')}(\sigma_1') >_{pref} \pi_{(t_j',t')}(\sigma_1)$ and since the relative occurrence of $t'$ and $t_j'$'s in $\sigma_1'$ and $\sigma_1\gamma$ are the same, a $t_j'$ must occur before the $(\#_{t'}(\sigma_1)+1)$'th $t'$ in $\sigma'$. But $\#_{t_j'}(\sigma) = \#_{t_j'}(\sigma')$ by choice of $t'$. Then, there must exist a $t_j'$ in $\sigma_2$ and this contradicts the assumption that all transitions in $\sigma_2$ were independent of $t'$. By using the properties of $(V, E)_N$ and $I$ (*e.g.*, permutation of consecutive independent transitions: if $M \xrightarrow{t} M' \xrightarrow{t'} M''$ and $tIt'$, then there exists an $M'''$ such that $M \xrightarrow{t'} M''' \xrightarrow{t} M''$),[1] we conclude that $t'$ must be enabled at $p'$ where $p \xrightarrow{\sigma_1} p'$. Hence, $t'$ is cc-enabled along $\sigma$. ■

## 4.3 The Logic P-CTL and its Interpretation

In this section, we assume a fixed labelled 1-safe net $N = (P, T, F, M_0, l)$. We introduce the logic P-CTL, whose syntax is

$$A \quad ::= \quad tt \mid \neg A \mid A_1 \wedge A_2 \mid \bigcirc_\alpha A \mid A_1\, U_\exists\, A_2 \mid A_1\, U_\forall\, A_2\ ,$$

where $\alpha \in Act$ and $tt$ is an abbreviation for $\mathsf{true}$. In Hennessy-Milner logic [Mil89], $<a>A$ expresses the fact that one can perform an action $a$ from a state and, in doing so, reach another state at which $A$ holds. Similarly, the $\bigcirc_\alpha A$ expresses that a transition labelled $\alpha$ can be fired reaching a state where $A$ holds. The "until" operators $U_\exists$ and $U_\forall$ will only quantify over computations rather than paths.
The logic is interpreted over the reachability graph $(V, E)_N$ of $N$ as follows, where $p \in V$, $\alpha \in Act$, and we have written $\models$ instead of $\models_N$.

- $p \models tt$

[1] To be more precise, we use the axioms of the corresponding labelled asynchronous transition system, which intuitively is $(V, E)_N$ augmented with $I$ [WN94].

- $p \models \neg A$ if and only if $p \not\models A$
- $p \models A_1 \ \wedge \ A_2$ if and only if $p \models A_1$ and $p \models A_2$
- $p \models \bigcirc_\alpha A$ if and only if $(\exists t \in T, q \in V.\, l(t) = \alpha \ \wedge \ p \xrightarrow{t} q \ \wedge \ q \models A)$
- $p \models A_1 \, U_\exists \, A_2$ if and only if $(\exists\, [\sigma] \in comp(p),\ p \xrightarrow{\sigma} = p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots.$
  $(\exists\, 0 \leq n \leq |\sigma|.\, (p_n \models A_2) \ \wedge \ (\forall 0 \leq i < n.\, p_i \models A_1)))$
- $p \models A_1 \, U_\forall \, A_2$ if and only if $(\forall [\sigma'] \in comp(p).\, (\forall\, \sigma \in [\sigma'],\ p \xrightarrow{\sigma} = p \xrightarrow{t_1} p_1 \xrightarrow{t_2}$
  $p_2 \cdots.$
  $(\exists\, 0 \leq n \leq |\sigma|.\, (p_n \models A_2) \ \wedge \ (\forall 0 \leq i < n.\, p_i \models A_1))))$

Furthermore, we define $\mathit{ff} \equiv \neg tt$, $<\alpha> A \equiv \bigcirc_\alpha A$, $[\alpha] A \equiv \neg <\alpha> \neg A$, $\mathsf{F}(A) \equiv tt \, U_\exists \, A$, $\mathsf{G}(A) \equiv \neg \mathsf{F}(\neg A)$, $\mathsf{Ev}(A) \equiv tt \, U_\forall \, A$, and $\mathsf{AI}(A) \equiv \neg \mathsf{Ev}(\neg A)$. The intended meaning of $\mathsf{Ev}(A)$ is that eventually/inevitably $A$ will hold along any computation, while $\mathsf{AI}(A)$ means that along some computation $A$ always holds. Notice that the path quantified formulas are no longer necessarily interpreted over a limit closed set of paths, as is the case with the standard interpretation of CTL.

*Example.* In the process agent example from Figure 1.3 we have $i \models \mathsf{Ev}(<b>tt)$.

Having given the necessary definitions, we end this section by defining the model-checking problem.

**Definition 55** Given a labelled 1-safe net $N = (P, T, F, M_0, l)$ and a formula $A$. The *model-checking problem of $N$ and $A$* is the problem of deciding whether or not $M_0 \models A$. □

## 4.4 A Tableau Method for Model-Checking

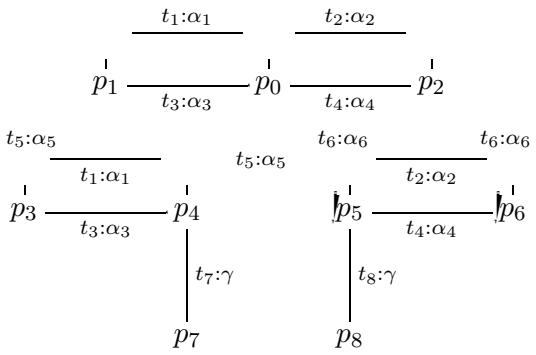
In this section we present a local model-checker based on a tableau system for model-checking formulas from our logic.

Local model-checking based on tableau systems has been presented in, *e.g.*, [Lar88, SW89]. As opposed to a global model-checker—as the on presented in [CES86]—which checks if all states of the system satisfies a formula, a local model-checker only checks if a specific state satisfies a given formula. For local model-checkers based on tableau systems this is done by only visiting (other) states if the tableau rules require it. Hence, the local model-checker may well be able to show that a state satisfies a formula without visiting all states of the system. For systems such as 1-safe nets a local model-checker may thus postpone the generation of the entire reachability graph and only generate the parts the tableau rules require. Since the size of the reachability graph can be exponentially bigger than the size of the net, a local model-checker sometimes has an advantage over a global model-checker, since it might avoid the "state space explosion problem".

We begin by considering an example to give some intuition about the problems we are faced with when looking for a tableau system. Since our interpretation of the logical operators in P-CTL coincides with the usual interpretation when the independence relation is empty, we would also like the tableau system to be a conservative extension of those presented in [Lar88, SW89]. The main difficulty is how to generalise the unfolding of formulas in P-CTL which correspond to minimal fixed-point assertions.

### 4.4.1 Unfolding Minimal Fixed-Point Assertions

Below we consider a very simple reachability graph $g_1$, which is generated by the 1-safe net $N_1$ in Figure 4.1.




The $t_i$'s are the transitions, the Greek letters the labels, and $p_0$ the initial marking. The independence relation is the smallest such containing $(t_1, t_5)$, $(t_3, t_5)$, $(t_2, t_6)$, and $(t_4, t_6)$. Clearly, $p_0 \models_{N_1} \neg\mathsf{Ev}(<\gamma>tt)$ since $[(t_1t_3t_2t_4)^\omega] \in comp(p_0)$ and no state along the computation $(t_1t_3t_2t_4)^\omega$ satisfies $<\gamma>tt$. However, if we drop the transitions $t_2$, $t_4$, $t_6$, and $t_8$ and call this reduced net $N_2$, we do indeed have $p_0 \models_{N_2} \mathsf{Ev}(<\gamma>tt)$, since every computation from $p_0$ must eventually reach $p_4$ — $t_5$ cannot be continuously ignored while repeatedly firing $t_1$ and $t_3$ since they are both *independent* of $t_5$.

Let us consider what a tableau (proof tree) for $p_0 \models_{g_2} \mathsf{Ev}(<\gamma>tt)$ might look like:

$$\dfrac{\dfrac{\dfrac{p_0 \vdash \mathsf{Ev}(<\gamma>tt)}{p_0 \vdash \mathsf{Ev}(<\gamma>tt)} \quad \dfrac{\dfrac{\dfrac{p_1 \vdash \mathsf{Ev}(<\gamma>tt)}{p_3 \vdash \mathsf{Ev}(<\gamma>tt)}}{p_4 \vdash \mathsf{Ev}(<\gamma>tt)}}{p_4 \vdash <\gamma>tt} \quad \dfrac{\dfrac{p_4 \vdash \mathsf{Ev}(<\gamma>tt)}{p_4 \vdash <\gamma>tt}}{p_7 \vdash tt}}{p_7 \vdash tt}$$

The above tree is constructed according to some intuitive tableau rules. Although informal, the example provides the first important observation. The leftmost branch begins and ends with the sequent $p_0 \vdash \mathsf{Ev}(<\gamma>tt)$. In the $\mu$-calculus $\mathsf{Ev}(<\gamma>tt)$ is expressed by the formula $\mu X.\ <\gamma>tt \vee ([Act]X \wedge\ <Act>tt)$. Hence, based on the tableau methods from [Lar88, SW89] one might expect that the above tree should be discarded as a tableau since the unfolding of the minimal fixed-point assertion reaches itself. However, in the current framework we interpret the logic over maximal traces and

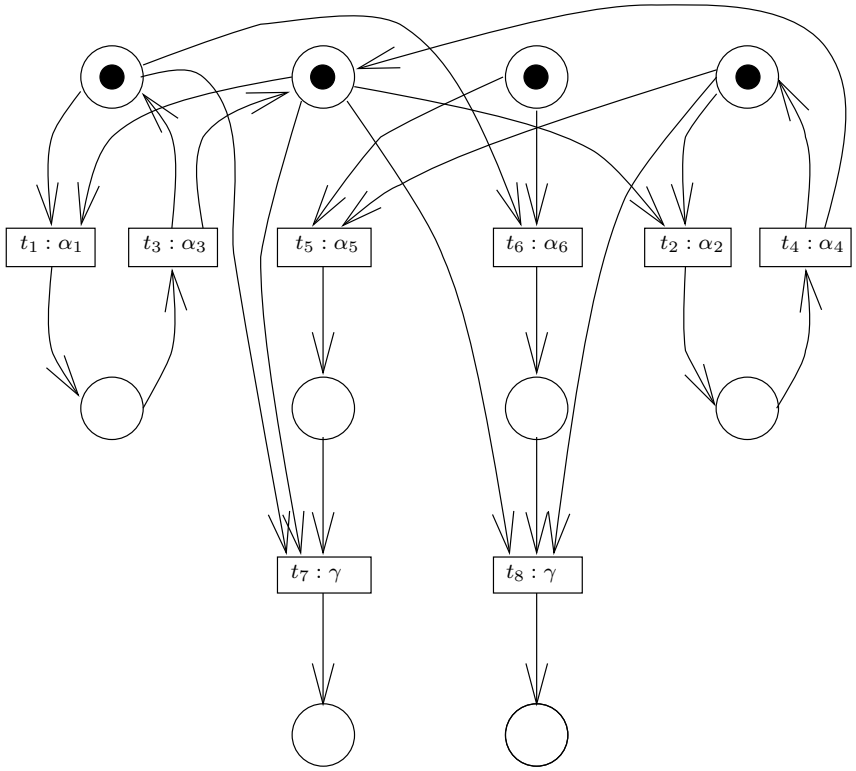



Figure 4.1: The net $N_1$.

the detected loop, $(p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_3} p_0)^\omega$, is not a computation from $p_0$ since the transition $t_5$ is cc-enabled. This example suggests that in certain cases we might allow the unfolding of a minimal fixed-point assertion to reach itself. These cases should certainly include the existence of a transition that is cc-enabled along the loop represented by such a branch. Our solution to this problem is to annotate the logic used in the tableau rules. The idea is to keep track of the transitions which are cc-enabled and update this information as one unfolds the reachability graph via the tableau rules. So in our case $t_5$ would be "kept track of" along the $p_0 \longrightarrow p_1 \longrightarrow p_0$ branch.

Let us consider a second example. This time we use $g_1$. Again, we construct in an intuitive and informal manner a tableau rooted in the sequent $p_0 \vdash \mathsf{Ev}(<\gamma>tt)$:

$$\dfrac{p_0 \vdash \mathsf{Ev}(<\gamma>tt)}{\text{Two subtrees as above} \quad \dfrac{p_5 \vdash \mathsf{Ev}(<\gamma>tt)}{\dfrac{p_5 \vdash <\gamma>tt}{p_8 \vdash tt}} \quad \dfrac{p_2 \vdash \mathsf{Ev}(<\gamma>tt)}{\dfrac{p_6 \vdash \mathsf{Ev}(<\gamma>tt)}{\dfrac{p_5 \vdash \mathsf{Ev}(<\gamma>tt)}{\dfrac{p_5 \vdash <\gamma>tt}{p_8 \vdash tt}}} \quad p_0 \vdash \mathsf{Ev}(<\gamma>tt)}}$$

Again, the interesting parts are the branches that unfold a minimal fixed-point assertion into itself. There are two such branches, the leftmost and the rightmost. However, along both of these there are transitions which are cc-enabled — $t_5$ for the left branch and $t_6$ for the right branch. According to the previous remarks these branches shouldn't discard the tree from being a tableau. But we do wish to discard the tree as a tableau since $p_0 \models_{g_1} \neg\mathsf{Ev}(<\gamma > tt)$. The problem is that by composing the two loops $(p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_3} p_0)$ and $(p_0 \xrightarrow{t_2} p_2 \xrightarrow{t_4} p_0)$ we can obtain an infinite computation $(p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_3} p_0 \xrightarrow{t_2} p_2 \xrightarrow{t_4} p_0)^\omega$ along which no state satisfies $<\gamma > tt$. This fact should discard the tree from being a tableau.

One solution to the problem of detecting such "combined" loops is to continue to unfold the minimal fixed-point assertions $p_0 \vdash \mathsf{Ev}(<\gamma>tt)$. If we unfold the fixed point assertion once more in the above example, still updating and propagating the information kept in the annotation, we will obtain a leaf with the information that we have found a looping path along which no transition is cc-enabled. This will discard the tree from being a tableau.

It turns out that the remaining problem is to find some general bound on the number of times we allow the unfolding of a minimal fixed-point assertion. The bound we use is at most $|T|$, the number of transitions in the labelled 1-safe net. In the next section we provide the necessary definitions.

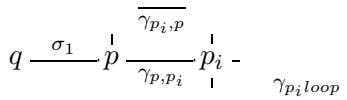
### 4.4.2 Tableau Rules

In this section we consider a fixed labelled 1-safe net $N$ and its reachability graph $(V, E)_N$.

We want to perform local model-checking by unfolding parts of the reachability graph into a tree structure. The tableau rules are supposed to guide this unfolding by imposing

constraints which restrict the size and shape of the tree structure. The main difficulty is handling the $U_\forall$ operator.

Consider a state $q$ such that $q \not\models A_1\ U_\forall\ A_2$. Then either there exists (i) a computation $\sigma$ such that $A_1 \wedge \neg A_2$ holds at all states along $q \xrightarrow{\sigma}$ until either a deadlock is reached or a state such that $\neg A_1 \wedge \neg A_2$ holds reached, or (ii) an infinite computation $\sigma$ such that $A_1 \wedge \neg A_2$ holds at all states along $q \xrightarrow{\sigma}$, referred to as an invalidating computation. Since the formulas are interpreted at states and the state space is finite, case (ii) reduces (simply by removing a finite number of loops from $\sigma$) to the existence of an infinite computation $\sigma_1\sigma_2$ from $q$, where $\sigma_1$ is finite and all states along $q \xrightarrow{\sigma_1}$ occur only once along $q \xrightarrow{\sigma_1} p \xrightarrow{\sigma_2}$ while all states along $p \xrightarrow{\sigma_2}$ occur infinitely often. Notice, $A_1 \wedge \neg A_2$ still holds at all states, as will be the case for the following computations. Using Lemma 54 it is possible to obtain from $\sigma_2$ an infinite path $\sigma_3$ from $p$ of the form $(\gamma_{p,p_1}\gamma_{p_1 loop}\gamma_{p_1,p}\cdots\gamma_{p,p_k}\gamma_{p_k loop}\gamma_{p_k,p})^\omega$, where all $\gamma$'s are finite and made up from subsequences of $\sigma_2$ and $1 \leq k \leq |T|$. The indices are intended to illustrate the structure of the loops as follows.




Also, since $\sigma_2$ was a computation from $p$, the $\gamma$'s can be chosen such that for any $t \in next(p)$ one of the $\gamma_{p_i loop}$'s will contain a transition in conflict with $t$. Hence, $\sigma_3$ is a computation from $p$. We refer to the illustrated loops $\gamma_{p,p_i}\gamma_{p_i loop}\gamma_{p_i,p}$ as *critical loops*. To conclude, $\sigma_1\sigma_3$ is an invalidating computation from $q$ along which all states satisfy $A_1 \wedge \neg A_2$.

In the example from Sect. 4.4.1, if we chose $p_0$ as $p$, then $p_0 \xrightarrow{t_1}\xrightarrow{t_3} p_0$ and $p_0 \xrightarrow{t_2}\xrightarrow{t_4} p_0$ would constitute critical loops. Actually, the sizes of the $\gamma$'s can be bound since the state space is finite. The important observation is that together with $|T|$ we obtain a bound on the length and number of $\gamma$'s we have to consider. The bounds will be encoded in the annotated logic.

**The Annotated Logic.**

The syntax of the annotated logic which is used in the tableau rules differs only from the previous in that the $U_\exists$ and $U_\forall$ operators are replaced by labelled counterparts. The $U_\exists$ operator is replaced by $U_\exists^{\mathcal{C}}$, where $\mathcal{C} \subseteq V$. The intuition is that $\mathcal{C}$ keeps track of which states have been visited and prevents unnecessary unfolding. For the $U_\forall$ operator we use a more elaborate annotation, $U_\forall^{\mathcal{C}}$, $U_\forall^{(p,n,T')}$, $U_\forall^{(p,n,T',V',\rightarrow)}$, and $U_\forall^{(p,n,T',V',\leftarrow)}$, where $p \in V$, $T' \subseteq T$, $V' \subseteq V$, and $0 \leq n \leq |T|$. $V'$ plays a role similar to $\mathcal{C}$, $n$ bounds the number of critical loops the tableau rules allow to explore, and $T'$ keeps track of which transitions have been concurrently enabled but ignored so far along a path. The emptiness of $T'$ will indicate that an invalidating computation has been found.

Let $Ann$ be the obvious homomorphism which annotates a formula $A$ (generated by the grammar in Sect. 4.3) by transforming every $U_\exists$ and $U_\forall$ into $U_\exists^\emptyset$ and $U_\forall^\emptyset$, respectively.

An annotated formula $B$ is said to be *clean* if there exists a formula $A$ such that $B$ equals $Ann(A)$.

**The Tableau Rules.**

The tableau rules will consist of rules for sequents of the form $p \vdash B$. The rules can be read from top to bottom as: "the top sequent (or conclusion) holds ($B$ holds at $p$) if the bottom sequents (or antecedents) and side conditions hold". $B$, $B_1$, and $B_2$ are assumed to be clean annotated formulas.

1) $$\frac{p \vdash B_1 \wedge B_2}{p \vdash B_1 \quad p \vdash B_2}$$

2) $$\frac{p \vdash \bigcirc_\alpha B}{q \vdash B}$$
- $t \in T, q \in V, p \xrightarrow{t} q$,
- $l(t) = \alpha$.

3) $$\frac{p \vdash B_1 \, U_\exists^{\mathcal{C}} \, B_2}{p \vdash B_2}$$
- $p \notin \mathcal{C}$.

4) $$\frac{p \vdash B_1 \, U_\exists^{\mathcal{C}} \, B_2}{p \vdash B_1 \quad q \vdash B_1 \, U_\exists^{\mathcal{C}\cup\{p\}} \, B_2}$$
- $p \notin \mathcal{C}, t \in T, q \in V, p \xrightarrow{t} q$.

5) $$\frac{p \vdash B_1 \, U_\forall^{\mathcal{C}} \, B_2}{p \vdash B_2}$$
- $p \notin \mathcal{C}$.

6) $$\frac{p \vdash B_1 \, U_\forall^{\mathcal{C}} \, B_2}{p \vdash B_1 \quad q_1 \vdash B_1 \, U_\forall^{\mathcal{C}\cup\{p\}} \, B_2 \cdots q_m \vdash B_1 \, U_\forall^{\mathcal{C}\cup\{p\}} \, B_2}$$
- $next(p) = \{t_1, \ldots, t_m\}$, $0 < m \in I\!N, p \notin \mathcal{C}$,
- $(\forall 1 \leq i \leq m.\, p \xrightarrow{t_i} q_i)$.

7) $$\frac{p \vdash B_1 \, U_\forall^{\mathcal{C}} \, B_2}{p \vdash B_1 \, U_\forall^{(p,|next(p)|,next(p))} \, B_2}$$
- $p \in \mathcal{C}$.

8) $$\frac{p \vdash B_1 \, U_\forall^{(p,n,T')} \, B_2}{p \vdash B_1 \, U_\forall^{(p,n-1,T',\emptyset,\rightarrow)} \, B_2}$$
- $0 < n \in I\!N, T' \neq \emptyset$.

9) $$\frac{q \vdash B_1 \, U_\forall^{(p,n,T',V',\rightarrow)} \, B_2}{q \vdash B_1 \quad q_i \vdash B_1 \, U_\forall^{(p,n,t_i IT',V'\cup\{q\},\rightarrow)} \, B_2}$$
- $q \notin V', next(q) = \{t_1, \ldots, t_m\}$, $0 < m \in I\!N$,
- $(\forall 1 \leq i \leq m.\, q \xrightarrow{t_i} q_i)$,

10) $$\frac{q \vdash B_1 \, U_\forall^{(p,n,T',V',\rightarrow)} \, B_2}{q \vdash B_2}$$
- $q \notin V'$.

11) $$\frac{q \vdash B_1 \, U_\forall^{(p,n,T',V',\rightarrow)} \, B_2}{q \vdash B_1 \, U_\forall^{(p,n,T',\emptyset,\leftarrow)} \, B_2}$$
- $q \in V'$.

12) $$\frac{q \vdash B_1 \, U_\forall^{(p,n,T',V',\leftarrow)} \, B_2}{q \vdash B_1 \quad q_i \vdash B_1 \, U_\forall^{(p,n,t_i IT',V'\cup\{q\},\leftarrow)} \, B_2}$$
- $q \notin V', next(q) = \{t_1, \ldots, t_m\}$, $0 < m \in I\!N, q \neq p$,
- $(\forall 1 \leq i \leq m.\, q \xrightarrow{t_i} q_i)$.

13) $\dfrac{q \vdash B_1\ U_{\forall}^{(p,n,T',V',\leftarrow)}\ B_2}{q \vdash B_2}$ - $q \notin V'$.

14) $\dfrac{p \vdash B_1\ U_{\forall}^{(p,n,T',V',\leftarrow)}\ B_2}{p \vdash B_1\ U_{\forall}^{(p,n,T')}\ B_2}$

Figure 4.2: The tableau rules.

Rule 1 to 4 need no further explanation. Referring to the notation from Sect. 4.4.2, Rule 5 and 6 are intended to detect $\sigma_1$, Rule 7 is intended to detect the "switch" to $\sigma_3$, Rule 8 to 10 are intended to detect $\gamma_{p,p_i}\gamma_{p_i loop}$, Rule 11 it intended to detect the "switch" to $\gamma_{p_i,p}$, and Rule 12 to 14 is intended to detect $\gamma_{p_i,p}$.

The next step is to define derivation trees which are build up according to the tableau rules.

**The Derivation Trees and Tableaux.**

In this section we define the tableaux. This is done by first defining a larger class of trees, derivation trees, which are generated according to the tableau rules. The next step is to restrict the class of derivation trees, using the annotation of the formulas, to a subclass of derivation trees which will be defined to be the tableaux.

Derivation trees are defined inductively in the usual manner, except perhaps for case of negated formulas. That is, if $\mathcal{T}_1, \ldots, \mathcal{T}_n$ are derivation trees with roots matching the antecedents of a rule and the side conditions are fulfilled, then one obtains a new derivation tree by "pasting the derivation trees together" according to the rule. The root of the new derivation tree is labelled by the conclusion of the rule. A tree consisting of a single node labelled with one of the following sequents is a derivation tree.

- $p \vdash tt$
- $p \vdash \neg B$
- $p \vdash B_1\ U_{\forall}^{(p,n,T')}\ B_2$, where $n = 0$ or $T' = \emptyset$
- $q \vdash B_1\ U_{\forall}^{(p,n,T',V',\leftarrow)} B_2$, where $q \in V'$

By applying the rules we can obtain new derivation trees, for example:

- If $\mathcal{T}_1$ is a derivation tree with root $p \vdash B_1$, $\mathcal{T}_2$ is a derivation tree with root $q \vdash B_1\ U_{\exists}^{\mathcal{C}\cup\{p\}}\ B_2$, where $p \notin \mathcal{C}$, and there exists a $t \in T$ such that $p \xrightarrow{t} q$, then $\dfrac{p \vdash B_1\ U_{\exists}^{\mathcal{C}}\ B_2}{\mathcal{T}_1 \quad \mathcal{T}_2}$ is a derivation tree with root $p \vdash B_1\ U_{\exists}^{\mathcal{C}}\ B_2$.
- If $\mathcal{T}$ is a derivation tree with root $p \vdash B_2$ and $p \notin \mathcal{C}$, then $\dfrac{p \vdash B_1\ U_{\forall}^{\mathcal{C}}\ B_2}{\mathcal{T}}$ is a derivation tree with root $p \vdash B_1\ U_{\forall}^{\mathcal{C}}\ B_2$.

Nothing else is a derivation tree.

We continue by defining the tableaux. In this step we get rid of derivation trees as for example $p \vdash \neg tt$. Sequents of the form $q \vdash B_1 \, U_\forall^{(q,n,\emptyset)} \, B_2$, where $n \in \mathbb{N}$ and $q \in V$, are called terminal sequents. A tableau is a derivation tree $\mathcal{T}$ with root $p \vdash Ann(A)$ such that either

- $A = tt$ or
- $A = \neg A'$ and there exists no tableau with root $p \vdash Ann(A')$ or
- $A$ is not of the above form and
  1. every proper subtree $\mathcal{T}'$ of $\mathcal{T}$ whose root is labelled with a clean formula is itself a tableau and
  2. $\mathcal{T}$ has no leaves labelled with terminal sequents.

A sequent $p \vdash B$ is *proved* by exhibiting a tableau with root $p \vdash B$.

### 4.4.3 Soundness and Completeness

Having given the necessary definitions we are now ready to state the main result.

**Theorem 56** *Given a finite labelled net $N = (P, T, F, M_0, l)$. Then for any state $p$ of $(V, E)_N$ ($p \in V$) and any formula $A$ we have:*

$$p \models A \text{ if and only if there exists a tableau with root } p \vdash Ann(A)$$

**Proof.** The proof proceeds by structural induction in $A$, showing soundness and completeness simultaneously. The main difficulty is the $U_\forall$ operator. For the soundness part, our observations from Sect. 4.4.2 provide the basis for a proof by contradiction. For the completeness part, using the induction hypothesis one can give a direct construction of a tableau. Intuitively, if $p \models A_1 \, U_\forall \, A_2$, then a tableau will be constructed (top-down from $p$) by always proving $q \vdash Ann(A_2)$ if $q \models A_2$ for any reached state $q$. Else, if $q \not\models A_2$, then one proves $q \vdash Ann(A_1)$, starts unfolding the graph from $q$, and continues by trying to prove $Ann(A_1 \, U_\forall \, A_2)$ at the states that are reached.

$\underline{tt}$ Clearly, we always have $p \models tt$ and the tableau $p \vdash tt$.

$\underline{\neg}$ We have $p \models \neg A$ if and only if $p \not\models A$ if and only if (induction hypothesis) there exists no tableau with root $p \vdash Ann(A)$ if and only if $p \vdash \neg Ann(A)$ is a tableau.

$\underline{\wedge}$ We have $p \models A_1 \wedge A_2$ if and only if $p \models A_1$ and $p \models A_2$ if and only if (induction hypothesis) there exists tableaux $\mathcal{T}_1$ with root $p \vdash Ann(A_1)$ and $\mathcal{T}_2$ with root $p \vdash Ann(A_2)$ if and only if there exists a tableau with root $p \vdash Ann(A_1 \wedge A_2)$, because $Ann(A_1 \wedge A_2) = Ann(A_1) \wedge Ann(A_2)$.

$\underline{\bigcirc_\alpha}$ We have $p \models \bigcirc_\alpha A$ if and only if $(\exists t \in T, q \in V.\ p \xrightarrow{t} q \ \wedge\ q \models A \ \wedge\ l(t) = \alpha)$ if and only if (induction hypothesis) there exists a tableau $\mathcal{T}$ and $(\exists t \in T, q \in V.\ p \xrightarrow{t} q \ \wedge\ l(t) = \alpha \ \wedge\ \mathcal{T}$ has root $q \vdash Ann(A))$ if and only if there exists a tableau $\mathcal{T}$ with root $p \vdash Ann(\bigcirc_\alpha A)$, since $Ann(\bigcirc_\alpha A) = \bigcirc_\alpha Ann(A)$.

$\underline{U_\exists}$ We have $p \models A_1\ U_\exists\ A_2$ if and only if $(\exists\, p_1, p_2, \ldots, p_n \in V, t_1, t_2, \ldots, t_n \in T, n \geq 0.$ $p = p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots \xrightarrow{t_n} p_n \ \wedge\ p_n \models A_2 \ \wedge\ (\forall 0 \leq i < n.\ p_i \models A_1) \ \wedge\ (\forall 0 \leq i < j \leq n.\ p_i \neq p_j))$ if and only if (induction hypothesis) there exists tableaux $\mathcal{T}_0, \ldots, \mathcal{T}_{n-1}$ with roots $p_0 \vdash Ann(A_1)\ , \ldots,\ p_{n-1} \vdash Ann(A_1)$ and $\mathcal{T}_n$ with root $p_n \vdash Ann(A_2)$ and transitions $t_1, \ldots, t_n$ such that $p = p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots \xrightarrow{t_n} p_n$ is loop free if and only if there exists a tableau with root $p \vdash Ann(A_1)\ U_\exists^\emptyset\ Ann(A_2)$, because $Ann(A_1\ U_\exists\ A_2) = Ann(A_1)\ U_\exists^\emptyset\ Ann(A_2)$.

$\underline{U_\forall}$ We show the bi-implication by showing the left and right implications separately.

"only if" direction (completeness):

We show how to obtain a derivation tree with root $p \vdash Ann(A_1\ U_\forall\ A_2)$. This will be done by providing an algorithm which will be shown to terminate and produce the desired tree. We then argue that it is a tableau.

The tree will be constructed from the root and expanded downwards. Only so-called "active" leaves of the current tree will be expanded. We try to keep the tree as small as possible by first trying to prove that $B_2$ holds at a state. Only if this isn't possible do we expand the tree.

During the presentation of the algorithm several claims will be stated. All of them will be shown to be valid in the succeeding paragraph. For convenience, we will write $B_1$ for $Ann(A_1)$ and $B_2$ for $Ann(A_2)$. So $Ann(A_1\ U_\forall\ A_2) = B_1\ U_\forall^\emptyset B_2$. The algorithm consists of the following steps:

1. Start by creating the root which is labelled by $p \vdash B_1\ U_\forall^\emptyset B_2$. Mark this node as *active*.
2. If possible choose an active node $N$, labelled by a sequence of one of the following forms:
   *i)* $q \vdash B_1\ U_\forall^{\mathcal{C}}\ B_2$
   *ii)* $q \vdash B_1\ U_\forall^{(q,n,T')}\ B_2$
   *iii)* $q \vdash B_1\ U_\forall^{(p,n,T',S',\leftrightarrow)}\ B_2$

   where $\leftrightarrow$ stands for either $\leftarrow$ or $\rightarrow$. Else terminate.
3. If $q \models A_2$, then by induction we have the existence of a tableau $\mathcal{T}'$ with root $q \vdash B_2$. Deactivate $N$ and paste $\mathcal{T}'$ below $N$ using rule 5, 10, or 13. None of the added nodes are active. Note that $q \models A_2$ excludes *ii)* because of the way the current tree has been expanded.

4. Else if $q \models \neg A_2$, then necessarily (Claim 1) $q \models A_1$. By induction there exists a tableau $\mathcal{T}'$ with root $q \vdash B_1$.
   * If $N$ is of the form $i$), and $q \notin \mathcal{C}$, then (Claim 2) $next(q) \neq \emptyset$ and apply rule 6, using $\mathcal{T}'$. Deactivate $N$ and activate the new leaves labelled $q_i \vdash B_1\ U_\forall^{\mathcal{C}\cup\{q\}}\ B_2$ that were added by application of rule 6.
   * If $N$ is of the form $i$) and $q \in \mathcal{C}$, then deactivate $N$ and, using rule 7, add a node below $N$ labelled $q \vdash B_1\ U_\forall^{(q,|T|,next(q))}\ B_2$. Using rule 8, because (Claim 3) $next(q) \neq \emptyset$, add yet another node below labelled $q \vdash B_1\ U_\forall^{(q,|T|-1,next(q),\emptyset,\rightarrow)}\ B_2$ which is activated.
   * If $N$ is of the form $ii$), then (Claim 4) $T' \neq \emptyset$. If $n = 0$, then deactivate $N$. Else if $n > 0$, then deactivate $N$ and apply rule 8, adding a node below $N$ labelled $q \vdash B_1\ U_\forall^{(q,n-1,T',\emptyset,\rightarrow)}$. Activate this node.
   * If $N$ is of the form $iii$) ($\rightarrow$) and $q \notin S'$, then (Claim 5) $next(q) \neq \emptyset$ and we deactivate $N$. By induction we have the existence of the tableau $\mathcal{T}'$ with root labelled $q \vdash B_1$. Using rule 9 add this tree below $N$ and add nodes labelled $q_i \vdash B_1\ U_\forall^{(p,n,t_iIT',S'\cup\{q\},\rightarrow)}\ B_2$. Only the last nodes are activated.
   * If $N$ is of the form $iii$) ($\rightarrow$) and $q \in S'$, then deactivate $N$ and using rule 11 add a node below $N$ labelled $q \vdash B_1\ U_\forall^{(p,n,T',\emptyset,\leftarrow)}\ B_2$. Activate this node.
   * If $N$ is of the form $iii$) ($\leftarrow$), $q \notin S'$, and $q \neq p$, then deactivate $N$. Because (Claim 6) $next(q) \neq \emptyset$, we can use rule 12 and the induction hypothesis to add a tableau $\mathcal{T}'$ with root labelled $q \vdash B_1$. Also, add nodes labelled $q_i \vdash B_1\ U_\forall^{(p,n,t_iIT',S'\cup\{q\},\leftarrow)}\ B_2$. Only these last nodes will be activated.
   * If $N$ is of the form $iii$) ($\leftarrow$) and $q \in S'$ and $q \neq p$, then deactivate $N$.
   * If $N$ is of the form $iii$) ($\leftarrow$) and $q = p$, then apply rule 14. Deactivate $N$ and activate the added node labelled $q \vdash B_1\ U_\forall^{(q,n,T')}\ B_2$
5. Goto 2.

We now observe the following:

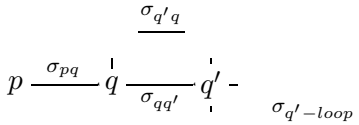
– The above "algorithm" terminates: One only expands *active* nodes and since $(V, E)_N$ is finite expansion cannot continue indefinitely because of the annotation of the formulas.

– All claims stated in the algorithm are valid: since the strategy used to compute the tree is to first try to prove that $A_2$ holds at a state, and if not, then expand the tree, we conclude that:

  * Claim 1 is valid: If $q \models \neg A_2$ and $q \models \neg A_1$, then because of the way the tree is expanded we could exhibit a finite path from $p$ along which $A_1 \wedge \neg A_2$ holds until $\neg A_1 \wedge \neg A_2$ holds. But since any finite path can

be extended to a computation ($K$ is assumed to be finite) we obtain a contradiction with the assumption $p \models A_1\ U_\forall\ A_2$.

* Claim 2 is valid: If $next(q) = \emptyset$, then we would have found a finite path starting at $p$ and ending in $q$, a deadlock. This would be a computation from $p$ to $q$ along which no state satisfied $A_2$. Again, this would contradict $p \models A_1\ U_\forall\ A_2$.
* Claim 3 is valid: Since $q \in \mathcal{C}$ we conclude $next(q) \neq \emptyset$.
* Claim 4 is valid: If $T' = \emptyset$, then because $T'$ keeps track of which transitions have been concurrently enabled along the loop starting and ending at $q$ (along the branch from the root of the tree to the current node), we would have detected one or more loops (see figure)




along which $A_2$ never holds, and by repeating these loops we could exhibit an infinite computation along which $A_2$ never holds. This contradicts $p \models A_1\ U_\forall\ A_2$.

* Claim 5 and Claim 6 are valid: As for Claim 2.

Assume the produced tree is not tableau. Then, using the induction hypothesis, we conclude that the only reason why the tree is not a tableau is that it has leaves labelled by terminal sequents. But then an argument similar to that used to show the validity of Claim 4 gives us a contradiction with the assumption $p \models A_1\ U_\forall\ A_2$.

"if" direction (soundness):

We show that if there exists a tableau $\mathcal{T}$ with root $p \vdash Ann(A_1\ U_\forall\ A_2)$, then $p \models A_1\ U_\forall\ A_2$. So assume that $p \models \neg(A_1\ U_\forall\ A_2)$, i.e., $p \models \neg A_2$ and there exists a $\sigma \in [\sigma'] \in comp(p)$ such that one of the following cases hold:

– $|\sigma| < \omega, p = p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} \cdots \xrightarrow{t_m} p_m \not\rightarrow, \sigma = t_1 \cdots t_m$, and

$$(\forall n \leq |\sigma|.\ (p_n \models \neg A_2)\ \vee\ (\exists 0 \leq i < n.\ p_i \models \neg A_1))$$

There are two cases.

* Assume $(\exists\, 0 < i \leq m.\ p_i \models A_2)$. Let $i_0 > 0$ denote the least such index. We know that there must exist an index $0 \leq j < i_0$ such that $p_j \models \neg A_1$. Let $j_0$ denote the least such index. Clearly, the path $t_1 \cdots t_{j_0}$ can be assumed to be loop free and traceable in $\mathcal{T}$ along nodes $q$, such that there exists a tableau with root $q \vdash B_1\ U_\forall\ B_2$ (using the induction hypothesis to obtain contradictions). But this gives a contradiction since $\mathcal{T}$ must

then have a subtree which is a tableau labelled with root $p_{j_0} \vdash Ann(A_1)$, i.e., $p_{j_0} \models A_1$.

* No states along $\sigma$ satisfies $A_2$. If there is a state which satisfies $\neg A_1$ along the path, the argument above can be applied. Else, for any $0 \leq i \leq m$ we have $p_i \models A_1 \ \wedge \ \neg A_2$. But then there must exist a loop free path from $p$ to $p_m$ such that $A_1 \ \wedge \ (\neg A_2)$ is satisfied along it and this path must be traceable in $\mathcal{T}$. But this means there must exist a leaf labelled $p_m \vdash Ann(A_1) \ U_\forall^{\mathcal{C}} \ Ann(A_2)$ such that $p_m \notin \mathcal{C}$, and since $p_m \not\longrightarrow$, $\mathcal{T}$ cannot be a derivation tree.

– $|\sigma| = \omega, p = p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} \cdots, \sigma = t_1 t_2 \cdots$, and

$$(\forall n \in I\!N.\ (p_n \models \neg A_2) \ \vee \ (\exists 0 \leq i < n.\ p_i \models \neg A_1))$$

As before, we extract two cases:

* $(\exists i \in I\!N.\ p_i \models A_2)$. Let $i_0 > 0$ be the least such index. As before we have a least index $0 \leq j_0 < i_0$ such that $p_{j_0} \models \neg A_1$. By repeating the above argument, we obtain a contradiction.

* $(\forall n \in I\!N.\ p_n \models \neg A_2)$. If there is a state which satisfies $\neg A_1$ along the path, the above argument can be applied. Else, we can obtain a path $\sigma' \in [\sigma'] \in comp(p)$ from $\sigma$ such that $p \xrightarrow{\sigma'} = p'_0 \xrightarrow{t'_1} p'_1 \xrightarrow{t'_2} \cdots \xrightarrow{t'_{m-1}} p'_{m-1} \xrightarrow{t'_m} p''_{k_0} \xrightarrow{t'''_{k_0+1}} p''_{k_0+1} \cdots$, where $p'_0, \ldots, p'_{m-1}$ occur only once along $\sigma'$ while $p''_{k_0}, \ldots$ occur infinitely often along $\sigma'$. (We simply remove a finite number of loops in $\sigma$, since $(V, E)_N$ is finite.) The common suffix ensures that no transition is cc-enabled along $\sigma'$.
  Since no transition is cc-enabled along $\sigma'$ there must exist finitely many nonempty loops $\sigma_{loop_1}, \ldots, \sigma_{loop_r}$ starting and ending at $p''_{k_0}$, such that no transition from $next(p''_{k_0})$ is cc-enabled along $(\sigma_{loop_1} \cdots \sigma_{loop_r})^\omega$, i.e., no enabled transition at $p_{k_0}$ is independent of all transitions taken in the $r$ loops. Notice that these loops might themselves contain loops. Also, since $|next(p''_{k_0})| \leq |T|$ we may assume that $1 \leq r \leq |T|$. Let $next(p''_{k_0}) = \{t'''_{j_1}, \ldots, t'''_{j_r}\}$. We may also assume that $\sigma_{loop_l}$ corresponds to a loop along which some transition in conflict with $t'''_{j_l}$ is taken.
  From each loop $\sigma_{loop_l}$ we can extract, by deleting inner loops, three paths $\sigma'_{loop_l}$, $\sigma''_{loop_l}$, and $\sigma'''_{loop_l}$ such that
  · $\sigma''_{loop_l}$ contains a transition in conflict with $t'''_{j_l}$
  · $p''_{k_0} \xrightarrow{\sigma'_{loop_l}} q_{k_0} \xrightarrow{\sigma''_{loop_l}} q_{k_0} \xrightarrow{\sigma'''_{loop_l}} p''_{k_0}$, for some $q_{k_0}$
  · $p''_{k_0} \xrightarrow{\sigma'_{loop_l}} q_{k_0}$ and $q_{k_0} \xrightarrow{\sigma'''_{loop_l}} p''_{k_0}$ are loop free
  · $q_{k_0} \xrightarrow{\sigma''_{loop_l}} q_{k_0}$ is a simple loop
  · all states along this new loop satisfy $A_1 \ \wedge \ \neg A_2$

$$\dfrac{i \vdash tt\, U_\forall^{\emptyset}(<b>tt)}{i \vdash tt \quad \dfrac{i \vdash tt\, U_\forall^{\{i\}}(<b>tt)}{\dfrac{i \vdash tt\, U_\forall^{(i,2,\{t_1,t_2\})}(<b>tt)}{\mathcal{T}_1}} \quad \dfrac{s_1 \vdash tt\, U_\forall^{\{i\}}(<b>tt)}{\dfrac{s_1 \vdash <b>tt}{s_2 \vdash tt}}}$$

where $\mathcal{T}_1$ is

$$\dfrac{i \vdash tt\, U_\forall^{(i,1,\{t_1,t_2\},\emptyset,\rightarrow)}(<b>tt)}{\dfrac{\dfrac{i \vdash tt\, U_\forall^{(i,1,\{t_2\},\{i\},\rightarrow)}(<b>tt)}{\dfrac{i \vdash tt\, U_\forall^{(i,1,\{t_2\},\emptyset,\leftarrow)}(<b>tt)}{i \vdash tt\, U_\forall^{(i,1,\{t_2\})}(<b>tt)}}{\mathcal{T}_2} \quad \dfrac{s_1 \vdash tt\, U_\forall^{(i,1,\{t_1\},\{i\},\rightarrow)}(<b>tt)}{\dfrac{s_1 \vdash <b>tt}{s_2 \vdash tt}} \quad i \vdash tt}$$

where $\mathcal{T}_2$ is

$$\dfrac{i \vdash tt\, U_\forall^{(i,0,\{t_2\},\emptyset,\rightarrow)}(<b>tt)}{i \vdash tt \quad \dfrac{i \vdash tt\, U_\forall^{(i,0,\{t_2\},\{i\},\rightarrow)}(<b>tt)}{\dfrac{i \vdash tt\, U_\forall^{(i,0,\{t_2\},\emptyset,\leftarrow)}(<b>tt)}{i \vdash tt\, U_\forall^{(i,0,\{t_2\})}(<b>tt)}} \quad \dfrac{s_1 \vdash tt\, U_\forall^{(i,0,\emptyset,\{i\},\rightarrow)}(<b>tt)}{\dfrac{s_1 \vdash <b>tt}{s_2 \vdash tt}}}$$

Figure 4.3: Example of a tableau.

But then, using the induction hypothesis, a prefix of the following path must be traceable in the tableau $\mathcal{T}$:

$$p \xrightarrow{t'_1} \cdots \xrightarrow{t'_m} p''_{k_0} \xrightarrow{\sigma} p''_{k_0} \xrightarrow{\sigma'_{loop_1}} \xrightarrow{\sigma''_{loop_1}} \xrightarrow{\sigma'''_{loop_1}} \cdots \xrightarrow{\sigma'_{loop_r}} \xrightarrow{\sigma''_{loop_r}} \xrightarrow{\sigma'''_{loop_r}} p''_{k_0}$$

where $\sigma$ is a nonempty simple loop obtained by deleting inner loops from the loop $\sigma'_{loop_1}\sigma''_{loop_1}\sigma'''_{loop_1}$ (rule 7 is going to be applied). The path must also end in a leaf labelled $p''_{k_0} \vdash Ann(A_1)\, U_\forall^{(p''_{k_0},n,\emptyset)}\, Ann(A_2)$, because the rules 9 and 12 keep track (in the annotation) of which transitions have been concurrently enabled. In our case there are no such transitions, so $\mathcal{T}$ cannot be a tableau and we obtain the desired contradiction.

■

As an example, we show that the process agent from Figure 1.3 will eventually be able to fire a transition labelled by a $b$ action (assume the transitions are $t_1$, $t_2$, and $t_3$ and are labelled $a$, $\tau$, and $b$, respectively). By the previous theorem, to show $i \models \mathsf{Ev}(<b>tt)$ it is sufficient to construct a tableau with root $i \vdash tt\, U_\forall^{\emptyset}\, (<b>tt)$. Figure 4.3 show such a tableau.

Notice that if we restrict ourselves to labelled 1-safe nets where the independence relation is empty and translate $A_1\, U_\exists\, A_2$ into $\mu X.\, A_2 \vee (A_1 \wedge <Act>X)$ and $A_1\, U_\forall\, A_2$

into $\mu X.\, A_2 \vee (A_1 \wedge <Act> tt \wedge [Act]X)$ (actually applying this translation recursively on the subformulas $A_1$ and $A_2$), our proof rules will work in essentially the same manner as those presented in [Lar88, SW89].

Choosing an instance of the model-checking problem to be a pair $(N, A)$ consisting of a labelled 1-safe net and a formula $A$ and defining its size to be the sum of the size of the net and the length of the formula, we obtain the following complexity result.

**Theorem 57** *The model-checking problem is PSPACE-complete.*

**Proof sketch.** The hardness result follows from easy modifications of the results in Chap. 2, while the PSPACE upper bound follows from a modification of the proof of Theorem 41 based on the observations in Sect. 4.4.2 (the bound on the number and length of the $\gamma$'s). ■

## 4.5 Model-Checking by State Labelling

In this section we present a state labelling based algorithm that solves the model-checking problem. The algorithm essentially works as the one presented for CTL in [CES86] except for the $U_\forall$ operator.

**Theorem 58** *Given a net $N$ and a formula $A$. Let $(V, E)_N$ denote the reachability graph of $N = (P, T, F, M_0, l)$. The following state labelling based algorithm solves the model-checking problem for $N$ and $A$ in time $O(|A|(|V| + |E||T|))$.*

**Proof.** Given a formula $A$ and a net $N$, the algorithm proceeds in stages as follows. In the first stage all subformulas of length one are processed. In general, at stage $i$ all subformulas of length $i$ are processed and at the end of stage $i$ a state is labelled with a subformula $A'$ of $A$ (or its negation $\neg A'$) if and only if it is satisfied in that state. Hence, after the $|A|$'th stage all states in $V$ will have been labelled with either $A$ or $\neg A$.

The data structures needed to perform the labelling are essentially those described for the CTL model-checker in [CES86]. The only exception is the $U_\forall$ operator ($U_\exists$ can be handled as the $EU$ operator in CTL, since any finite prefix of a path can be extended to a computation.). The $U_\forall$ operator is handled as follows:

Assume we want to label the states with the subformula $A' = A_1\ U_\forall\ A_2$. All states must already have been labelled with $A_1$ or $\neg A_1$, and $A_2$ or $\neg A_2$. Then, states labelled with $A_2$ are labelled with $A'$, and states labelled with $\neg A_1$ and $\neg A_2$ are labelled with $\neg A'$. The remaining states must all be labelled with $A_1$ and $\neg A_2$. The next step is to compute the maximal strongly connected components of $(V, E)$ restricted to these remaining states.

Let us denote the graph whose nodes are these maximal strongly connected components by $G'$. $G'$ is a directed acyclic graph (DAG) whose nodes are sets of states of $V$. As long as there is a terminal node $n$ in $G'$, repeatedly do the following:

1) If there is a state $p \in n$, a transition $t \in T$, and a state $p' \in V$ such that $p \xrightarrow{t} p'$ and $p'$ is labelled with $\neg A'$, then label all states in $n$ with $\neg A'$. Furthermore, for all nodes $m$ in $G'$, if $n$ can be reached from $m$ then label all states in $m$ with $\neg A'$. Remove all processed nodes from $G'$ and let $G'$ denote the new DAG.

2) Else, if there is a state $p$ in $n$ but no transition $t$ and state $p' \notin n$ such that $p \xrightarrow{t} p'$, then label all states in $n$ (and $m$'s above $n$, as described just above) with $\neg A'$ (there must exist an invalidating computation in $n$ from $p$). Update $G'$ as above.

3) Else, all states of $n$ have successor states not in $n$. Moreover, these successor states are all labelled by $A'$. Assume $T = \{t_1, \ldots, t_k\}$.

   - Initialise a boolean array $B$ of length $k$ such that all its entries are set to *False*. Then, for each edge $\xrightarrow{t_i}$ between any two states in $n$, set all entries $B[j]$ such that $\neg(t_i I t_j)$ to *True*.
   - If there is an entry $B[l]$ which is *False* and $t_l$ is enabled at any state in $n$, then label all states in $n$ with $A'$. Remove $n$ from $G'$ and let $G'$ denote the new DAG.
   - Else, label all states in $n$ (and $m$'s, as described in the first case) with $\neg A'$ and update $G'$ as above.

It should be obvious that case 1) labels the states in $n$ correctly. Case 2) is also correct because we can exhibit a computation in $n$ whose states are labelled with $A_1$ and $\neg A_2$. Case 3) is correct because of the following observation: there exists a computation inside $n$ if and only if there is no transition $t_l$ that is (i) independent of all transition labelling edges between states in $n$, and (ii) enabled at (necessarily all) a state in $n$.

An analysis of the algorithm yields the time complexity $O(|A|(|V| + |E||T|))$. Hence, our algorithm is comparable to the one presented in [CES86]. ■

## 4.6 Extensions of P-CTL and Undecidability Results

In this section we present different extensions of P-CTL with modal operators expressing concurrent or conflicting behaviour. We prove that the satisfiability problem for some of these logics is undecidable for finite as well as infinite labelled nets, if we impose an "injectivety" constraint on their reachability graphs.

*Remark.* Infinite labelled nets are a generalisation of finite labelled nets, where the sets $P$, $T$, and $F$ may be at most countably infinite. Hence, for any state $p$ in the reachability graph of the nets we consider, $comp(p) \neq \emptyset$.

### 4.6.1 The New Modal Operators

Assume a fixed labelled net $N = (P, T, F, M_0, l)$ and let $(V, E)_N$ denote its reachability graph and $I$ the independence relation.

First, we give the syntax of the new modal operators by extending the grammar from Sect. 4.3 with the following rules, where $\alpha_i \in Act$ and $n > 0$:

$$A \quad ::= \quad \ll\!\gg((\alpha_1, A_1), \ldots, (\alpha_n, A_n)) \mid \sharp((\alpha_1, A_1), \ldots, (\alpha_n, A_n)) \mid \langle\alpha_1 \cdots \alpha_n\rangle A \mid \rangle\alpha_1 \cdots \alpha_n\langle A \ .$$

The interpretation is:

- $p \models \ll\!\gg((\alpha_1, A_1), \ldots, (\alpha_n, A_n))$ if and only if
  $(\exists\, t_1, \ldots, t_n \in T, q_1, \ldots, q_n \in V.$
  $(\forall\, 1 \leq i \leq n.\ p \xrightarrow{t_i} q_i \ \wedge\ l(t_i) = \alpha_i \ \wedge\ q_i \models A_i)$
  $\wedge\ (\forall\, 1 \leq i < j \leq n.\ t_i I t_j))$ ,

- $p \models \sharp((\alpha_1, A_1), \ldots, (\alpha_n, A_n))$ if and only if
  $(\exists\, t_1, \ldots, t_n \in T, q_1, \ldots, q_n \in V.$
  $(\forall\, 1 \leq i \leq n.\ p \xrightarrow{t_i} q_i \ \wedge\ l(t_i) = \alpha_i \ \wedge\ q_i \models A_i) \wedge$
  $(\forall\, 1 \leq i < j \leq n.\ (t_i, t_j) \notin I))$ ,

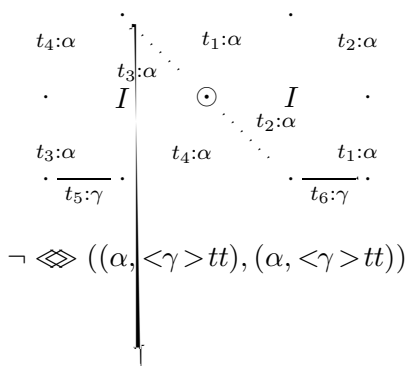
- $p \models \langle\alpha_1 \cdots \alpha_n\rangle A$ if and only if
  $(\exists\, t_1, \ldots, t_n \in T, q \in V.$
  $(\forall\, 1 \leq i \leq n.\ l(t_i) = \alpha_i) \ \wedge\ (\forall\, 1 \leq i < j \leq n.\ t_i I t_j) \wedge$
  $p \xrightarrow{t_1 \cdots t_n} q \ \wedge\ q \models A)$ , and

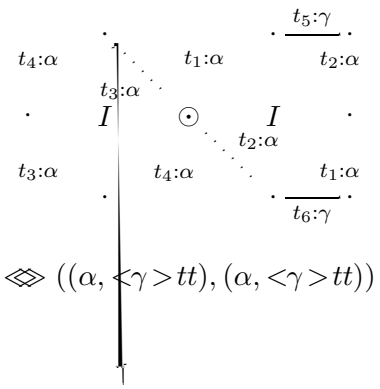
- $p \models \rangle\alpha_1 \cdots \alpha_n\langle A$ if and only if
  $(\exists\, t_1, \ldots, t_n \in T, q \in V, p \xrightarrow{t_1 \cdots t_n} q.$
  $(\forall\, 1 \leq i \leq n.\ l(t_i) = \alpha_i) \wedge$
  $(\forall\, 1 \leq i < n.\ (t_i, t_{i+1}) \notin I) \ \wedge\ q \models A)$ .

The $\ll\!\gg$ and $\langle\rangle$ operators specify concurrent behaviour.[2] The difference between them is that $\langle\rangle$ requires a property $A$ to hold *after* the execution of a set of mutually independent (labelled) transitions, while $\ll\!\gg$ requires properties $A_i$ to hold after the execution of (labelled) transitions $t_i$ from a set of mutually independent transitions. In a similar way, the $\sharp$ and $\rangle\langle$ operators specify conflicting behaviour.

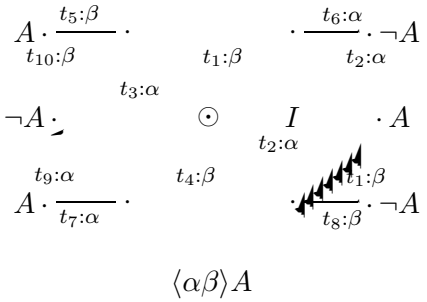
The $\ll\!\gg$, $\langle\rangle$, $\sharp$, and $\rangle\langle$ operators might be replaced by others. We have chosen to present them because they can distinguish the following situations. All the depicted transition systems are reachability graphs of nets. Transitions that are independent are indicated by an $I$ in their "independence square". The initial state is indicated by $\odot$ and states labelled by $A$ or $\neg A$ indicate that one has to extend the reachability graph in a trivial manner such that a property $A$, e.g., $<\gamma>tt$, either holds or doesn't, as indicated by $A$ or $\neg A$:

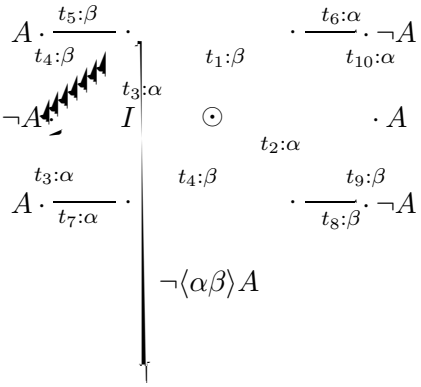
[2] The $\langle\rangle$ operator resembles the one proposed in [LPRT93].



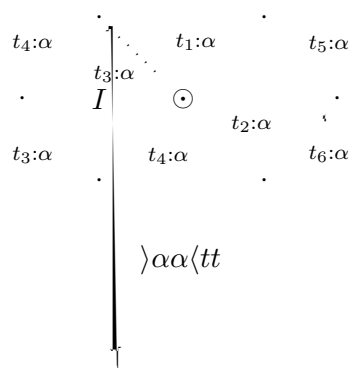



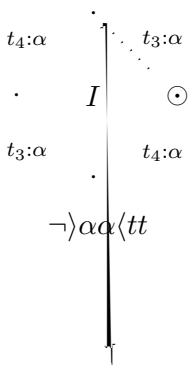

The above two reachability graphs cannot be distinguished by the $\langle\rangle$, $\rangle\langle$, or $\sharp$ operator. To see this, notice that all states having $\alpha$ and $\beta$ labelled transitions satisfy the same formulas, as is the case for states having exactly one $\alpha$ labelled transition and states having no transitions. By induction it can then be shown that the two initial states satisfy the same formulas (not containing the $\ll\!\gg$ operator. The same reasoning applies to the three following examples.



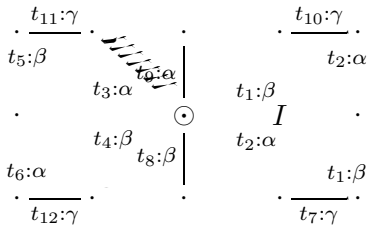




The above two reachability graphs cannot be distinguished by the $\ll\!\gg$, $\rangle\langle$, or $\sharp$ operator.

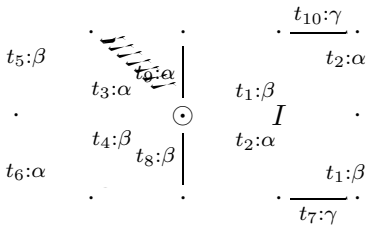



The above two reachability graphs cannot be distinguished by the $\ll\!\gg$, $\langle\rangle$, or $\sharp$ operator.




$\sharp((\alpha, <\gamma>tt), (\beta, <\gamma>tt))$




$\neg\sharp((\alpha, <\gamma>tt), (\beta, <\gamma>tt))$

The above two reachability graphs cannot be distinguished by the $\lll\ggg$, $\langle\rangle$, or $\rangle\langle$ operator.

### 4.6.2 The Undecidability Results

We will concentrate on the extensions of P-CTL which contains the operators $\lll\ggg$. Actually, for any extension of P-CTL with any of the other three presented operators the following undecidability results hold.

**Definition 59** $N$ is said to be *injective* if for all $p \in V$ and $t, t' \in T$ it is the case that $p \xrightarrow{t}, p \xrightarrow{t'}$, and $l(t) = l(t')$ implies $t = t'$. □

**Definition 60** The *(finite) injective satisfiability problem* is the problem of deciding, given a formula $A$, whether there exists a (finite) injective labelled net $N$ such that $M_0 \models A$. If this is the case, $A$ will be said to be *(finitely) injectively satisfiable.* □

The following problems are known to be undecidable [Ber66, Har85, LPRT93].

**Definition 61** The *colouring problem*, $CP$. An *instance* of the problem is a quadruple $\mathcal{C} = (C, R, U, c_0)$, where $C = \{c_0, \ldots, c_k\}$ is a finite nonempty set of colours, and $R, U : C \to \mathcal{P}(C) - \{\emptyset\}$ are the "right" and "up" adjacency functions. A *solution* to $\mathcal{C}$ is a function $Col : \mathbb{N} \times \mathbb{N} \to C$ such that:

- $Col(0,0) = c_0$
- $(\forall (i,j) \in \mathbb{N} \times \mathbb{N}.\ Col(i, j+1) \in U(Col(i,j)) \ \wedge \ Col(i+1, j) \in R(Col(i,j)))$

□

**Definition 62** The *finite colouring problem*, $FCP$. An *instance* of the problem is a quintuple $\mathcal{C}_F = (C, R, U, c_0, c_f)$, where $C = \{c_0, \ldots, c_k\}$ is a finite nonempty set of colours, $c_f \in C$, and $R, U : C \to \mathcal{P}(C) - \{\emptyset\}$ are the "right" and "up" adjacency functions. A *solution* to $\mathcal{C}_F$ is a triple $(Col, M, N)$, where $M, N \in \mathbb{N}$, $Col : \{0, \ldots, M\} \times \{0, \ldots, N\} \to C$ is such that:

- $Col(0,0) = c_0$
- $(\forall 0 \leq i < M, 0 \leq j \leq N.\ Col(i+1, j) \in R(Col(i,j)))$

- $(\forall 0 \leq i \leq M, 0 \leq j < N.\ Col(i, j+1) \in U(Col(i,j)))$
- $Col(M, N) = c_f$

□

For the logics containing the $\ll\gg$ operator we have the following result.

**Theorem 63** *The set of injectively satisfiable formulas non-recursive.*

**Proof.** We reduce the colouring problem to the injective satisfiability problem. Given $\mathcal{C} = (C, R, U, c_0)$, we construct a formula $A_{\mathcal{C}}$, a conjunct of five formulas given below, that is injectively satisfiable if and only if $\mathcal{C}$ has a solution.

Assume that the labels $\alpha_0, \ldots, \alpha_k$, *up*, and *right* are distinct symbols. The five conjuncts are the following:

- $A_1 = <\alpha_0> tt$ , $Col(0,0) = c_0$
- $A_2 = G(\ll\gg ((<right>, tt), (<up>, tt)))$ , coding of grid
- $A_3 = G(\bigwedge_{i=0}^{k}(<\alpha_i> tt \Leftrightarrow \bigwedge_{j \neq i}[\alpha_j] ff))$ , exactly one colour
- $A_4 = G(\bigwedge_{i=0}^{k}(<\alpha_i> tt \Rightarrow [right](\bigvee_{c_j \in R(c_i)} <\alpha_j> tt)))$ , right adjacency
- $A_5 = G(\bigwedge_{i=0}^{k}(<\alpha_i> tt \Rightarrow [up](\bigvee_{c_j \in U(c_i)} <\alpha_j> tt)))$ , up adjacency

We claim that $A_{\mathcal{C}} = \bigwedge_{i=1}^{5} A_i$ is injectively satisfiable if and only if $\mathcal{C}$ has a solution. The "if" direction is easy and therefore omitted. The "only if" direction follows the lines in [LPRT93] and makes essential use of injectivity of the solution to $A_{\mathcal{C}}$ and the following *"diamond"* and *"commutativity"* properties of the reachability graph of a labelled net $N$:

- If $p \in V$, $t, t' \in T$, $p \xrightarrow{t} q$, $p \xrightarrow{t'} q'$, and $tIt'$, then there exists $q'' \in V$ such that $q \xrightarrow{t'} q''$ and $q' \xrightarrow{t} q''$.
- If $p \in V$, $t, t' \in T$, $p \xrightarrow{t} q \xrightarrow{t'} q'$, and $tIt'$, then there exists $q'' \in V$ such that $p \xrightarrow{t'} q'' \xrightarrow{t} q'$.

■

**Theorem 64** *The set of formulas that are finitely injectively satisfiable is non-recursive.*

**Proof.** We reduce the finite colouring problem to the finite injective satisfiability problem. Given $\mathcal{C}_F = (C, R, U, c_0, c_f)$, we construct a formula $A_{\mathcal{C}_F}$, a conjunct of seven formulas given below, that is finitely injectively satisfiable if and only if $\mathcal{C}_F$ has a solution.

Again, assume that the labels $\alpha_0, \ldots, \alpha_k$, *UM*, *RM*, *up*, and *right* are distinct symbols. The seven conjuncts are the following:

- $A_1 = <\alpha_0> tt$ , $Col(0,0) = c_0$
- $A_2 = G((\lll (right, tt), (up, tt)) \wedge [RM]ff \wedge [UM]ff) \vee$
  $(<up> tt \wedge <RM> tt \wedge [right]ff \wedge [UM]ff) \vee$
  $(<right> tt \wedge <UM> tt \wedge [up]ff \wedge [RM]ff) \vee$
  $(<c_f> tt \wedge <RM> tt \wedge <UM> tt \wedge [right]ff \wedge [up]ff))$ , grid structure
- $A_3 = G(\bigwedge_{i=0}^{k}(<\alpha_i> tt \Leftrightarrow \bigwedge_{j \neq i}[\alpha_j]ff))$ , exactly one colour
- $A_4 = G(\bigwedge_{i=0}^{k}(<\alpha_i> tt \Rightarrow [right](\bigvee_{c_j \in R(c_i)} <\alpha_j> tt)))$ , right adjacency
- $A_5 = G(\bigwedge_{i=0}^{k}(<\alpha_i> tt \Rightarrow [up](\bigvee_{c_j \in U(c_i)} <\alpha_j> tt)))$ , up adjacency
- $A_6 = EV(<\alpha_f> tt \wedge <RM> tt \wedge <UM> tt)$ , $c_f$ in upper right corner
- $A_7 = G(<RM> tt \Rightarrow [up] <RM> tt) \wedge$
  $G(<UM> tt \Rightarrow [right] <UM> tt)$ , consistent borders

We claim that $A_{\mathcal{C}_F} = \bigwedge_{i=1}^{7} A_i$ is injectively satisfiable if and only if $\mathcal{C}_F$ has a solution. The "if" direction is easy and therefore omitted. The "only if" direction is nontrivial and follows from the next two lemmas. We use another proof technique than [LPRT93] since they assume a fixed finite alphabet. We only assume finiteness about the set of transitions $T$ of the solution to $A_{\mathcal{C}_F}$. ■

Assume that we have fixed $\mathcal{C}_F$ and a finite injective net $N$ such that $M_0 \models A_{\mathcal{C}_F}$, where $A_{\mathcal{C}_F}$ is defined in the proof of Theorem 64. We will use the notation $p \xrightarrow{\alpha} p'$ to indicate that there is a transition $t \in T$ such that $p \xrightarrow{t} p'$ and $l(t) = \alpha$. This shouldn't lead to any confusion since $N$ is assumed injective.

**Lemma 65** *Assume $N$ is a net such that $M_0 \models A_{\mathcal{C}_F}$. If there exist $p_0, \ldots, p_n \in V$ and $t_1, \ldots, t_n \in T$ such that $p_0 \xrightarrow{t_1} \ldots \xrightarrow{t_n} p_n, p_0 = M_0$, for all $1 \leq j \leq n$ either $l(t_j) = right$ or $l(t_j) = up$, and no state $p_j$ except $p_n$ has an enabled transition labelled $\alpha_f$, then $\mathcal{C}_F$ has a solution.*

**Proof.** Let $\beta_0, \ldots, \beta_n$ denote the unique labels among $\{\alpha_0, \ldots, \alpha_k\}$ that by $A_3$ must label some enabled transition at $p_0, \ldots, p_n$, respectively.

If $\beta_0 = \alpha_f$, then $n = 0$ and $\beta_0 = \alpha_0$, and obviously we have a solution. So assume that we have $n > 0$. By $A_2$ one of the following cases must hold.

- $p_0 \xrightarrow{right} p_1$ and $p_0 \xrightarrow{UM}$: By $A_2$, $A_7$, and our assumptions we conclude that $p_0 \xrightarrow{right} p_1 \xrightarrow{right} \ldots \xrightarrow{right} p_n$ and $\forall 0 \leq i \leq n.\ p_i \xrightarrow{UM}$. Since $p_n \xrightarrow{\alpha_f}$ we easily obtain a solution by $A_4$ and $A_5$.
- $p_0 \xrightarrow{up} p_1$ and $p_0 \xrightarrow{RM}$: Symmetric to the above case.

- $p_0 \xrightarrow{right}$ and $p_0 \xrightarrow{up}$: Without loss of generality we can assume that $l(t_1) = right$, i.e., $p_0 \xrightarrow{right} p_1$. By injectivity and $A_2$ there must exist $p', p'' \in V$ such that $p_0 \xrightarrow{up} p' \xrightarrow{right} p''$ and $p_0 \xrightarrow{right} p_1 \xrightarrow{up} p''$. Continuing this way as long as $l(t_j) = right$ and exploiting injectivety we conclude there must exist $p', p'', \ldots, p^{(m+1)}$ such that

$$\begin{array}{ccccccc} p' & \xrightarrow{right} & p'' & \xrightarrow{right} & \cdots & \xrightarrow{right} & p^{(m+1)} \\ {\scriptstyle up}\downarrow & & {\scriptstyle up}\downarrow & & & & {\scriptstyle up}\downarrow \\ p_0 & \xrightarrow{right} & p_1 & \xrightarrow{right} & \cdots & \xrightarrow{right} & p_m \end{array}$$

  If $m = n$, then we easily obtain a solution, since $p_n \xrightarrow{\alpha_f}$. So assume $m < n$. Then $l(t_{m+1}) = up$ and by injectivety we conclude that $p^{(m+1)} = p_{m+1}$. Again, if $\beta_{m+1} = \alpha_f$ we are done by $A_4$ and $A_5$, so assume this isn't the case. Then $m + 1 < n$. We continue by showing how to expand the above $1 \times (m + 1)$ grid to a $2 \times (m + 1)$ grid if $l(t_{m+2}) = up$ or to a $1 \times (m + 2)$ grid if $l(t_{m+2}) = right$.

  - Assume that $l(t_{m+2}) = up$. By $A_2$, $A_7$, and injectivety we conclude there must exist a $q^{(m)}$ such that $p^{(m)} \xrightarrow{right} p^{(m+1)} \xrightarrow{up} p_{m+2}$ and $p^{(m)} \xrightarrow{up} q^{(m)} \xrightarrow{right} p_{m+2}$. By repeating this we obtain:

$$\begin{array}{ccccccccc} q' & \xrightarrow{right} & q'' & \xrightarrow{right} & \cdots & \xrightarrow{right} & q^{(m)} & \xrightarrow{right} & p_{m+2} \\ {\scriptstyle up}\downarrow & & {\scriptstyle up}\downarrow & & & & {\scriptstyle up}\downarrow & & {\scriptstyle up}\downarrow \\ p' & \xrightarrow{right} & p'' & \xrightarrow{right} & \cdots & \xrightarrow{right} & p^{(m)} & \xrightarrow{right} & p^{(m+1)} \\ {\scriptstyle up}\downarrow & & {\scriptstyle up}\downarrow & & & & {\scriptstyle up}\downarrow & & {\scriptstyle up}\downarrow \\ p_0 & \xrightarrow{right} & p_1 & \xrightarrow{right} & \cdots & \xrightarrow{right} & p_{m-1} & \xrightarrow{right} & p_m \end{array}$$

  - Assume that $l(t_{m+2}) = right$. By similar arguments we get:

$$\begin{array}{ccccccccc} p' & \xrightarrow{right} & p'' & \xrightarrow{right} & \cdots & \xrightarrow{right} & p^{(m+1)} & \xrightarrow{right} & p_{m+2} \\ {\scriptstyle up}\downarrow & & {\scriptstyle up}\downarrow & & & & {\scriptstyle up}\downarrow & & {\scriptstyle up}\downarrow \\ p_0 & \xrightarrow{right} & p_1 & \xrightarrow{right} & \cdots & \xrightarrow{right} & p_m & \xrightarrow{right} & p^{(m+2)} \end{array}$$

    This procedure can be continued for $t_{m+3}, \ldots, t_n$ giving us a grid from which it is easy to obtain a solution to $\mathcal{C}_F$, by $A_1$, $A_3$, $A_4$, and $A_5$.

■

Next, we proof there exist a path of the form mentioned in Lemma 65 if $M_0 \models A_{\mathcal{C}_F}$.

**Lemma 66** *If $N$ is a net such that $M_0 \models A_{\mathcal{C}_F}$, then there exist $p_0, \ldots, p_n \in V$ and $t_1, \ldots, t_n \in T$ such that $p_0 \xrightarrow{t_1} \ldots \xrightarrow{t_n} p_n, p_0 = M_0$, for all $1 \leq j \leq n$ either $l(t_j) = right$ or $l(t_j) = up$, and no state $p_j$ except $p_n$ has an enabled transition labelled $\alpha_f$.*

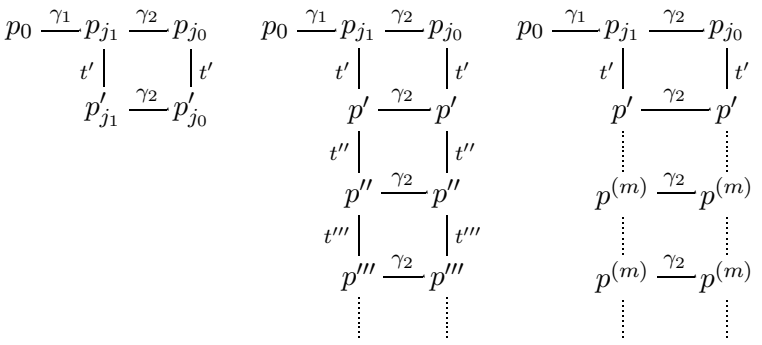



Diagram 1 Diagram 2 Diagram 3

Figure 4.4: Looping diagrams.

**Proof.** Assume by contradiction that there doesn't exist any path of the above form. By $A_2$ either $M_0 \xrightarrow{right}$ or $M_0 \xrightarrow{left}$ Without loss of generality we may assume $M_0 \xrightarrow{right}$. We continue by a case analysis.

- Assume $M_0 \xrightarrow{UM}$: We know that $M_0 \not\xrightarrow{\alpha_f}$, and choosing $p_1$ to be the unique state such that $M_0 = p_0 \xrightarrow{right} p_1$ we also conclude $p_1 \not\xrightarrow{\alpha_f}$. Now by $A_2$, $A_6$, $A_7$, and our assumptions it must be the case that $p_1 \xrightarrow{right}$ and $p_1 \xrightarrow{UM}$. Continuing this way we exhibit an infinite path $p_0 \xrightarrow{right} p_1 \xrightarrow{right} \cdots$ such that $p_j \not\xrightarrow{\alpha_f}$ and $p_j \xrightarrow{UM}$ for all states $p_j$ along the path. Since $K$ is finite there must exist a least $j_0$ such that $p_{j_0}$ is visited twice along $p_0 \xrightarrow{right} \cdots \xrightarrow{right} p_{j_0}$. Let $0 \leq j_1 < j_0$ be the index such that $p_{j_1} = p_{j_0}$. This gives us an infinite path $p_0 \xrightarrow{\gamma_1} p_{j_1} \xrightarrow{\gamma_2} p_{j_0} \xrightarrow{\gamma_2} p_{j_0} \xrightarrow{\gamma_2} \cdots$, where $\gamma_1$ ($\gamma_2$) is the sequence of *right* labelled transitions leading from $p_0$ to $p_{j_1}$ (from $p_{j_1}$ to $p_{j_0}$). But by $A_6$ the above path cannot be a computation from $p_0$. Hence there must exist a transition $t'$ that is cc-enabled along $p_{j_0} \xrightarrow{\gamma_2} p_{j_0} \xrightarrow{\gamma_2} \cdots$. By the diamond and commutativity properties of $(V, E)_N$ we obtain Diagram 1 in Figure 4.4, where $p'_{j_1} = p'_{j_0}$ is a state in $V$.

  Now by $A_2$, $\gamma_2$ being labelled by *right*, and $A_6$, we conclude that there must exist a transition $t''$ which is cc-enabled along the loop obtained by repeating $p'_{j_1} \xrightarrow{\gamma_2} p'_{j_0}$. By repeating this argument we obtain ($p' = p'_{j_1} = p'_{j_0}$) Diagram 2 in Figure 4.4.

  Since $N$ is finite, the set $T$ is finite. All of the reached states have the property that $<c_f>tt \wedge\ <RM>tt \wedge\ <UM>tt$ doesn't hold, since $<right>tt$ and $A_2$ hold. One can now repeat the above argument, observing that finiteness of $V$ implies that some $p^{(m)}$ must occur twice along the leftmost vertical path in Diagram 2. From this observation we can construct Diagram 3 in Figure 4.4.

  Again, we conclude there must exist a transition that is cc-enabled along the looping part of Diagram 3. This transition must be independent of all the transitions on the looping parts between $p^{(m)}$ shown in Diagram 3, especially the transitions labelled *right*. Also, all states along these loops have an enabled transition labelled *right*.

Let $(V', E')$ denote the transition system obtained by restricting $(V, E)_N$ to the states satisfying $\neg(< c_f > tt \ \wedge \ < RM > tt \ \wedge \ < UM > tt)$. It should now be clear that one can produce an infinite computation from $p_0$ which stays in $(V', E')$. One modifies the above infinite path by choosing transitions that are cc-enabled following the above scheme. It is important to notice that because it is always possible to insert loops containing transitions labelled *right* (the $\gamma_2$ loops) the states reached by "taking a cc-enabled transition" also contain a self loop of transitions labelled *right*. Since $(V, E)_N$ is finite there are only a finite number of maximal strongly connected components in $(V', E')$. Hence, by repeating the above procedure one will only be able to proceed towards terminal maximal strongly connected components. This eventually produces a computation along which $\neg(<c_f>tt \wedge \ <RM>tt \wedge \ <UM>tt)$ holds, contradicting $A_6$.

- Assume $M_0 \xrightarrow{up}$: A similar way of reasoning leads to the desired contradiction. To sketch the argument: Choose consecutive *right* labelled transitions as far as possible. If one produces an infinite path labelled *right* the argument is as above. Else one must eventually reach a state with enabled transitions labelled *up* and *RM* (else contradicting our main assumption by $A_2$). From this state choose the infinite path labelled *up* (else contradicting our main assumption by $A_2$ and $A_7$). Apply the above argument in a symmetric way.

■

For the remaining operators we have the following corollary.

**Lemma 67** *For the logics containing at least one of the operators $\sharp$, $\langle\rangle$, or $\rangle\langle$, Theorem 63 and 64 remain true.*

**Proof sketch.** Replace $\ll\gg((<right>, tt), (<up>, tt))$ in $A_2$ in the proof of Theorem 63 and 64 by either

- $(<right>tt \wedge \ <up>tt \ \wedge \ \neg\sharp((right, tt), (up, tt)))$,
- $(\langle right\, up\rangle tt)$, or
- $(<right><up>tt \ \wedge \ \neg\rangle right\, up\langle tt)$

depending on which operator is available. ■

## 4.7 Summary

Partial order semantics for concurrent systems have gained interest because interleaving models of concurrency have failed to provide an acceptable interpretation of what it means for events of a concurrent system to be independent. Much work has been devoted to transfer obtained results and notions from the interleaving models to the

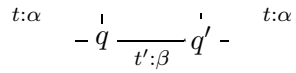
"true concurrency" models [JNW93, JM93, WN94, LPRT93]. Trying to contribute to the "transferring of results" we have provided proof rules for a CTL-like logic interpreted over maximal traces. The work which we have tried to transfer can be found in [Lar88, SW89]. Our work supports automatic verification of distributed systems whose liveness properties are only provable under the assumption of progress.

There is a trade off between the rules and the definition of tableaux. One can obtain simple rules at the cost of a complicated definition of tableaux.[3] At the cost of presenting less simple rules we have kept the definition of tableaux simple.

In [Val90, WG93], it is shown that based on a partial order semantics deadlock detection can sometimes be made efficient. Valmari has also applied stubborn sets to reduce the state space search for performing model-checking of a linear temporal logic not containing the "next" operator [KV92]. Future research could be to investigate to which extend model-checking can benefit from this application of partial order semantics.

Another direction might be to consider how to handle a more expressive logic (perhaps one containing a recursion operator) in a similar way, i.e., define the interpretation of the formulas over maximal traces and proving soundness and completeness of some set of proof rules.

The general satisfiability problem for our logic is still an open problem. Consider the following example




where $t$ is independent of $t'$. Let $a$ be an atomic proposition that holds at $q$ and but not at $q'$ ($a$ can be simulated by having an $a$ labelled enabled transition at all states where $a$ should hold). Then, the formula $(\neg \mathsf{AI}(a)) \wedge (a \wedge \mathsf{Inv}(a \Rightarrow <\alpha> a))$ is satisfied at $q$. But under the usual CTL-interpretation the formula (where $<\alpha>$ is read as the "next" operator) is unsatisfiable (because the set of paths one quantifies over is limit closed). The important observation is that our interpretation of $\mathsf{AI}$ does not quantify over the path $q \xrightarrow{\sigma}$, where $\sigma = t^{\infty}$.

We also investigated extensions of the logic with modal operators expressing concurrent behaviour. It turns out that restricted versions of the satisfiability problem for these logics is undecidable. Axiomatizations of these logics remain to be investigated.

[3] The set of simple rules we have identified requires a global side condition in the definition of tableaux.

# Part II

# On Open Maps

# Chapter 5

# Open Maps (at) Work

**Contents**

## 5.1 Introduction

In the next section we give a short stepwise introduction to open maps as presented in [JNW93, JNW94]. Then, in the subsequent sections, we apply the theory of open maps by instantiating the definitions with different models and notions of (simulation) morphisms and characterise the obtained abstract notion of equivalence operationally. It turns out that our choices of categories, which are guided by our intuitive understanding of what it means for a system to simulate another, yield well known notions of equivalence. More specifically Sect. 5.3.1 to Sect. 5.3.5 are devoted to trace equivalence, weak bisimulation, testing equivalence, barbed bisimulation, and probabilistic bisimulation. In each of these sections we follow the steps presented in the next section. In Sect. 5.4 we consider non-interleaving models and discuss some derived "true concurrency" behavioural equivalences and related decision problems. In Sect. 5.5 we summarise and discuss future directions.

Along, we make several observations clear which are either rather implicit in [JNW93, JNW94] or not mentioned at all.

## 5.2 Open Maps

In this section we briefly recall the basic definitions from [JNW93].

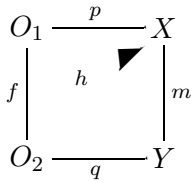
As presented there, the general setting requires several steps. First, a category which represents a *model of computation* has to be identified. We denote this category $\mathcal{M}$. A morphism $m : X \longrightarrow Y$ in $\mathcal{M}$ should intuitively be thought of as a simulation of $X$ in $Y$. Then, within $\mathcal{M}$ we choose a subcategory of "observation objects" and "observation extension" morphisms between these objects. We denote this *category of observations* by $\mathcal{P}$. Given an observation (object) $P$ in $\mathcal{P}$ and a model $X$ in $\mathcal{M}$. $P$ is said to be an *observable behaviour* of $X$ if there exists a morphism $p : P \longrightarrow X$ in $\mathcal{M}$.

Next, we identify morphisms $m : X \longrightarrow Y$ which have the property that whenever an observable behaviour of $X$ can be extended via $f$ in $Y$ then that extension can be matched by an extension of the observable behaviour in $X$.

**Definition 68 Open Maps**
A morphism $m : X \longrightarrow Y$ in $\mathcal{M}$ is said to be $\mathcal{P}$-*open* if whenever $f : O_1 \longrightarrow O_2$ in $\mathcal{P}$, $p : O_1 \longrightarrow X$, $q : O_2 \longrightarrow Y$ in $\mathcal{M}$, and the diagram

$$\begin{array}{ccc} O_1 & \xrightarrow{\;p\;} & X \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle m} \\ O_2 & \xrightarrow[\;q\;]{} & Y \end{array} \tag{5.1}$$

commutes, i.e., $m \circ p = q \circ f$, there exists a morphism $h : O_2 \longrightarrow X$ in $\mathcal{M}$ such that the two triangles in the diagram

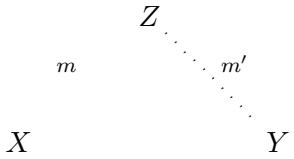



(5.2)

commute, i.e., $p = h \circ f$ and $q = m \circ h$. When no confusion is possible, we refer to $\mathcal{P}$-open morphisms as *open maps*. □

The abstract definition of bisimilarity is as follows.

**Definition 69** **$\mathcal{P}$-bisimilarity**
Two models $X$ and $Y$ in $\mathcal{M}$ are said to be *$\mathcal{P}$-bisimilar*, written $X \sim_{\mathcal{P}} Y$, if there exists a *span of open maps* from a common object $Z$:




(5.3)

□

Notice that if $\mathcal{M}$ has pullbacks, it can be shown that $\sim_{\mathcal{P}}$ is an equivalence relation. The important observation is that pullbacks of open maps are themselves open maps. For more details, the reader is referred to [JNW93].

In the next sections, we proceed by following the above presented steps.

As a preliminary example of a category of models of computation $\mathcal{M}$ we present *labelled transition systems*.

**Definition 70** A *labelled transition system* over $Act$ is a tuple

$$(S, i, Act, \longrightarrow) \ , \tag{5.4}$$

where $S$ is a set of states with initial state $i$, $Act$ is a set of actions ranged over by $\alpha$, $\beta$, ..., and $\longrightarrow \subseteq S \times Act \times S$ is the transition relation. For the sake of readability we introduce the following notation. Whenever $(s_0, \alpha_1, s_1)$, $(s_1, \alpha_2, s_2)$, ..., $(s_{n-1}, \alpha_n, s_n) \in \longrightarrow$ we denote this as $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n$ or $s_0 \xrightarrow{v} s_n$ where $v = \alpha_1\alpha_2 \cdots \alpha_n \in Act^*$. Also, we assume that all states $s \in S$ are reachable from $i$, i.e., there exists a $v \in Act^*$ such that $i \xrightarrow{v} s$. □

Let us briefly remind the reader about Park and Milner's definition of strong bisimulation.

**Definition 71** Let $T_1 = (S_1, i_1, Act, \longrightarrow_1)$ and $T_2 = (S_2, i_2, Act, \longrightarrow_2)$. A *strong bisimulation between $T_1$ and $T_2$* is a relation $R \subseteq S_1 \times S_2$ such that

$$(i_1, i_2) \in R \ , \tag{5.5}$$

$$((r,s) \in R \ \wedge \ r \xrightarrow{\alpha}_1 r') \ \Rightarrow \ \text{for some } s', (s \xrightarrow{\alpha}_2 s' \ \wedge \ (r', s') \in R) \ , \tag{5.6}$$

$$((r,s) \in R \ \wedge \ s \xrightarrow{\alpha}_2 s') \ \Rightarrow \ \text{for some } r', (r \xrightarrow{\alpha}_1 r' \ \wedge \ (r', s') \in R) \ . \tag{5.7}$$

$T_1$ and $T_2$ are said to be *strongly bisimilar* if there exists a strong bisimulation between them. □

Henceforth, whenever no confusion is possible we drop the indexing subscripts on the transition relations and write $\longrightarrow$ instead.

By defining morphisms between labelled transition systems we can obtain a category of models of computation, $\mathcal{LTS}$, labelled transition systems.

**Definition 72** Let $T_1 = (S_1, i_1, Act, \longrightarrow_1)$ and $T_2 = (S_2, i_2, Act, \longrightarrow_2)$. A morphism $m : T_1 \longrightarrow T_2$ is a function $m : S_1 \longrightarrow S_2$ such that

$$m(i_1) = i_2 \ , \tag{5.8}$$

$$s \xrightarrow{\alpha}_1 s' \ \Rightarrow \ m(s) \xrightarrow{\alpha}_2 m(s') \ . \tag{5.9}$$

□

Composition of morphisms is defined as the usual composition of functions. The intuition behind this specific choice of morphism is that an $\alpha$ labelled transition in $T_1$ must be simulated by an $\alpha$ labelled transition in $T_2$.

If, as done in [JNW93], one chooses $\mathcal{P}$ as the full subcategory of $\mathcal{M}$ whose objects are finite synchronisation trees with at most one maximal branch, i.e., labelled transition systems of the form

$$i \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n \ , \tag{5.10}$$

$\mathcal{P}$-bisimilarity corresponds to Park and Milner's strong bisimulation.

**Theorem 73** *Given two labelled transition systems $T_1$ and $T_2$. Then:*

*$T_1$ and $T_2$ are strongly bisimilar if and only if they are $\mathcal{P}$-bisimilar.*

This follows from the following characterisation of $\mathcal{P}$-open maps [JNW93].

**Lemma 74** *A morphism $m : T_1 \longrightarrow T_2$ is $\mathcal{P}$-open if and only if it satisfies the following "zig-zag" property:*

*If $m(r) \xrightarrow{\alpha} s$ then there exists an $r'$ such that $r \xrightarrow{\alpha} r'$ and $m(r') = s$.*

In the following sections we shall "rediscover" well-known behavioural equivalences by varying $\mathcal{M}$ and $\mathcal{P}$. In the following, whenever we write, e.g., $\mathcal{M}$, $\mathcal{P}$, or $\mathcal{P}$-bisimilarity they refer to the specific choices of categories made in the section they appear.

## 5.3 Behavioural Equivalences

In this section we show how various well-known, and quite different, behavioural equivalences can be presented using the theory of open maps.

### 5.3.1 Trace Equivalence

In this section we show how trace equivalence between two labelled transition systems can be captured by open maps. Trace equivalence is perhaps the first and simplest equivalence between labelled transition systems that one can think of. The result is based on the following important fact: Two labelled transition systems are trace equivalent if and only if their underlying deterministic transition systems are bisimilar.

First, we present the category $\mathcal{LTS}_1$ of *labelled transition systems*, which will corresponds to $\mathcal{M}$. Then, we identify a subcategory, $\mathcal{P}$, of observations. Finally, we show that $\mathcal{P}$-bisimilarity corresponds to trace equivalence.

The object of $\mathcal{LTS}_1$ are the labelled transition systems (lts) from Definition 70. The following definition is needed in the definition of the morphisms in $\mathcal{LTS}_1$.

**Definition 75** Given an lts $T = (S, i, Act, \longrightarrow)$. For nonempty sets $X, Y \subseteq S$, we define $X \xrightarrow{\alpha} Y$ if $Y = \{r' \in S \mid \exists\, r \in X.\, r \xrightarrow{\alpha} r'\}$. Notice that this transition relation is deterministic. As before, the transition relation can be generalised to a relation $X \xrightarrow{v} Y$, where $v \in Act^*$. Furthermore, we define $RS(T)$, *the reachability set of* $T$, to be the least subset of $2^S/\{\emptyset\}$, such that

$$\{i\} \in RS(T)\ , \tag{5.11}$$

$$X \in RS(T) \text{ and } X \xrightarrow{\alpha} Y \text{ implies } Y \in RS(T)\ . \tag{5.12}$$

□

Next, we define morphisms between two ltss.

**Definition 76** Given two ltss, $T_j = (S_j, i_j, Act, \longrightarrow_j), j = 1, 2$. A morphism $m$ between $T_1$ and $T_2$ is a function $m$ from $RS(T_1)$ to $RS(T_2)$, such that

$$m(\{i_1\}) = \{i_2\}\ , \tag{5.13}$$

$$X \xrightarrow{\alpha} Y \text{ implies } m(X) \xrightarrow{\alpha} m(Y)\ . \tag{5.14}$$

□

Composition of morphisms is defined as the usual composition of functions. This defines the category $\mathcal{LTS}_1$.

The intuition behind this definition of (simulating) morphism is that one is only interested in what action sequences an lts can perform. After performing a sequence $\sigma = \alpha_1 \cdots \alpha_n$ of actions from the initial state $i$ one may in general end up in several different states of $T$, i.e., a set $X$ of states of $T$. These sets of states are exactly the elements of $RS(T)$. Extending the sequence $\sigma$ by performing another action $\alpha$ then corresponds to performing an $\alpha$ transition from $X$.

Next step is to define $\mathcal{P}$.

**Definition 77** Let $\mathcal{P}$ be the full subcategory of $\mathcal{LTS}_1$ whose objects are of the form

$$i \xrightarrow{\alpha_1} r_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} r_n \ , \tag{5.15}$$

where all states are distinct. □

Apart from showing that $\mathcal{LTS}_1$ has pullbacks, the construction in the following lemma will be referred to in the main theorem of this section. Also, it follows at once from the remarks in Sect. 5.2 that $\mathcal{P}$-bisimilarity is an equivalence relation. We will also be able to conclude this after proving the main theorem of this section; it states that $\mathcal{P}$-bisimilarity coincides with trace equivalence, which is know to be an equivalence relation. However, in general one cannot expect that $\mathcal{P}$-bisimilarity coincides with a known equivalence relation. Lemmas as the following are sufficient for $\mathcal{P}$-bisimilarity to be an equivalence relation.

**Lemma 78** *$\mathcal{LTS}_1$ has pullbacks.*

**Proof.** Given a diagram

$$\begin{array}{ccc} & T_1 & \qquad\qquad T_2 \\ & {\scriptstyle m_1} \searrow \qquad \swarrow {\scriptstyle m_2} & \\ & T & \end{array}$$

Define an lts $T' = (S', i', Act, \longrightarrow')$ as follows. $S' \subseteq RS(T_1) \times RS(T_2)$ and $\longrightarrow' \subseteq (RS(T_1) \times RS(T_2)) \times Act \times (RS(T_1) \times RS(T_2))$ are the least sets such that

- $i' = (\{i_1\}, \{i_2\}) \in S'$
- If $(X, Y) \in S'$, $X \xrightarrow{\alpha} X'$, $Y \xrightarrow{\alpha} Y'$
  then $(X', Y') \in S'$ and $((X, Y), \alpha, (X', Y')) \in \longrightarrow'$.

Notice that because the transition relations on the reachability sets are deterministic it is the case that $m_1(X) = m_2(Y)$ for any $(X, Y) \in S'$ and $RS(T')$ contains only singletons. Let $\pi_1 : T' \longrightarrow T_1$ be defined as $\pi_1(\{(X, Y)\}) = X$. It can be shown that $\pi_1$ is well defined and is a morphism from $T'$ to $T_1$. We can define $\pi_2 : T' \longrightarrow T_2$ in a similar way.

Given an lts $T''$ and two morphisms $f_1 : T'' \longrightarrow T_1$ and $f_2 : T'' \longrightarrow T_2$ such that $m_1 \circ f_1 = m_2 \circ f_2$. Define $h : T'' \longrightarrow T'$ by $h(Z) = (f_1(Z), f_2(Z))$ for $Z \in RS(T'')$. From the definition of $T'$ is should be easy to see that $h$ is a morphism; the initial state of $T''$ is mapped to that of $T'$ and transitions are preserved. Furthermore we also have $f_1 = \pi_1 \circ h$ and $f_2 = \pi_2 \circ h$. This is trivial, since there is at most one morphism between any two objects in $\mathcal{LTS}_1$. Hence, $h$ is also unique. This gives us the desired pullback. ■

The next lemma characterises the open maps in $\mathcal{LTS}_1$.

**Lemma 79** *A morphism $m : T_1 \longrightarrow T_2$ is $\mathcal{P}$-open if and only if $m : RS(T_1) \longrightarrow RS(T_2)$ has the following "zig-zag" property*

*If $m(X) \xrightarrow{\alpha} Y'$, then there exists an $X'$ such that $X \xrightarrow{\alpha} X'$ and $m(X') = Y'$.*

**Proof.** Assume $m$ is $\mathcal{P}$-open and $m(X) \xrightarrow{\alpha} Y'$. Then it must be the case that $X_0 \xrightarrow{\alpha_1} X_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} X_n$ for some $X_0, \ldots, X_n \in RS(T_1)$, where $X_0 = \{i_1\}$ and $X_n = X$. Also, we have $Y_0 \xrightarrow{\alpha_1} Y_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} Y_n \xrightarrow{\alpha} Y'$, where $Y_j = m(X_j)$ for $0 \leq j \leq n$. Let $O_1$ be the observation

$$i \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n \ ,$$

and $O_2$ be the observation

$$i' \xrightarrow{\alpha_1} s'_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s'_n \xrightarrow{\alpha} s'_{n+1} \ .$$

Now let $f$ denote the unique morphism from $O_1$ to $O_2$, $p$ denote the morphism that maps $\{i\}$ to $\{i_1\}$ and $\{s_j\}$ to $X_j$ for $1 \leq j \leq n$, and $q$ denote the morphism that maps $\{i'\}$ to $\{i_2\}$, $\{s'_j\}$ to $Y_j$ for $1 \leq j \leq n$, and $\{s'_{n+1}\}$ to $Y'$. We then have $m \circ p = q \circ f$. From our assumptions it then follows that there exists a morphism $h : O_2 \longrightarrow T_1$ such that $p = h \circ f$ and $q = m \circ h$. We now conclude $h(\{s'_n\}) = h(f(\{s_n\})) = p(\{s_n\}) = X$, $h(\{s'_n\}) \xrightarrow{\alpha} h(\{s'_{n+1}\})$, and $m(h(\{s'_{n+1}\})) = q(\{s'_{n+1}\}) = Y'$. Now choose $X'$ as $h(\{s'_{n+1}\})$.

Conversely, assume $m$ has the "zig-zag" property and we are given a commuting diagram

$$\begin{array}{ccc} O_1 & \xrightarrow{\ p\ } & X \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle m} \\ O_2 & \xrightarrow[\ q\ ]{} & Y \end{array}$$

where $O_1$ is an observations of the form

$$i \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n \ ,$$

and $O_2$ an observation of the form

$$i' \xrightarrow{\alpha_1} s'_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_m} s'_m \ ,$$

and $n \leq m$. Notice $f$ is uniquely determined (maps $\{s_j\}$ to $\{s'_j\}$ for $1 \leq j \leq n$). We will show how to define a morphism $h : O_2 \longrightarrow T_1$ such that $p = h \circ f$ and $q = m \circ h$. We start by defining $h(\{i'\}) = \{i_1\}$ and $h(\{s'_j\}) = p(\{s_j\})$ for $1 \leq j \leq n$. Notice that we now already have $p = h \circ f$ for the partially defined $h$. Consequently, $q = m \circ h$ on $\{i'\}$, $\{s'_1\}, \ldots, \{s'_n\}$ because of the way $f$ is defined and $m \circ p = q \circ f$. Now assume $n < m$. Since $m(p(\{s_n\})) = q(f(\{s_n\})) = q(\{s'_n\}) \xrightarrow{\alpha_{n+1}} q(\{s'_{n+1}\})$ we know there must exist an $X'$ such that $p(\{s_n\}) \xrightarrow{\alpha_{n+1}} X'$ and $m(X') = q(\{s'_{n+1}\})$. Now define $h(\{s'_{n+1}\}) = X'$. Then $m(h(\{s'_{n+1}\})) = q(\{s'_{n+1}\})$. Continuing this way for the remaining $\{s'_{n+2}\}, \ldots, \{s'_m\}$ we obtain the desired morphism. ■

**Definition 80** Given an lts $T = (S, i, Act, \longrightarrow)$. The *traces/language of* $T$, denoted $L(T)$, is defined as

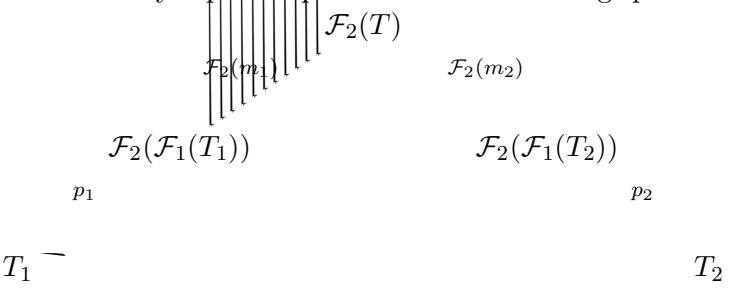
$$L(T) = \{v \in Act^* \mid i \xrightarrow{v} r \text{ for some } r \in S\} \ . \tag{5.16}$$

Two ltss, $T_1$ and $T_2$, are said to be *trace equivalent* if $L(T_1) = L(T_2)$. □

**Theorem 81** *Given two* ltss $T_1$ *and* $T_2$. *Then:*

*$T_1$ and $T_2$ are trace equivalent if and only if they are $\mathcal{P}$-bisimilar.*

**Proof.** The "if" direction follows from Lemma 79. For the "only if" direction, let $\mathcal{F}_1$ be the functor from $\mathcal{LTS}_1$ to $\mathcal{LTS}$ which sends an object $T$ to $(RS(T), \{i\}, Act, \longrightarrow)$, where $\longrightarrow$ was defined in Definition 75, and an morphism $m : T \longrightarrow T'$ to the obvious morphism between $\mathcal{F}_1(T)$ and $\mathcal{F}_1(T')$ defined by $m : RS(T) \longrightarrow RS(T')$. Let $\mathcal{F}_2$ be the functor from $\mathcal{LTS}$ to $\mathcal{LTS}_1$ which maps an object to itself and a morphism $m : T \longrightarrow T'$ to the morphism determined uniquely by the induced function $m : RS(T) \longrightarrow RS(T')$. Since all observations of $\mathcal{LTS}_1$ are isomorphic to their image under $\mathcal{F}_1$ and there is at most one morphism between any two objects in $\mathcal{LTS}_1$, we conclude, using Lemma 74, that $\mathcal{F}_2$ preserves open maps. So assume that $T_1$ and $T_2$ are trace equivalent. We then know that $\mathcal{F}_1(T_1)$ and $\mathcal{F}_1(T_2)$ are strong bisimilar. From Sect. 5.2 this implies that there exists a span of open maps in $\mathcal{LTS}$, $m_1 : T \longrightarrow \mathcal{F}_1(T_1)$ and $m_2 : T \longrightarrow \mathcal{F}_1(T_2)$. Also, there clearly exist isomorphisms $p_1 : \mathcal{F}_2(\mathcal{F}_1(T_1)) \longrightarrow T_1$ and $p_2 : \mathcal{F}_2(\mathcal{F}_1(T_2)) \longrightarrow T_2$. Since isomorphisms are always open maps we have the following span of open maps:




We conclude that $T_1$ and $T_2$ are $\mathcal{P}$-bisimilar. Observe that a construction similar to the one used in Lemma 78 would also have provided a span of open maps. ■

*Remark.* Edmund Robinson has later observed that trace equivalence can also be captured by choosing $\mathcal{LTS}$ as the category $\mathcal{M}$, and $\mathcal{P}$ as the subcategory consisting of objects of the form (5.10), and morphisms which are either identity morphisms or morphisms whose domains are observations having only one state. $\mathcal{P}$-open maps are then characterised by

**Lemma 82** *A morphism $m : T_1 \longrightarrow T_2$ is $\mathcal{P}$-open if and only if it has the following "zig-zag" property for $v \in Act^*$*

*If $i_2 \xrightarrow{v} s$, then there exists an $r$ such that $i_1 \xrightarrow{v} r$ and $m(r) = s$.*

Intuitively, any observation in $T_2$ can be "lifted" to $T_1$. From this lemma the following theorem easily follows.

**Theorem 83** *Given two* ltss $T_1$ *and* $T_2$*. Then:*

$$T_1 \text{ and } T_2 \text{ are trace equivalent if and only if they are } \mathcal{P}\text{-bisimilar.}$$

We have chosen to present trace equivalence using reachability sets, partly because they will be used to present Hennessy's testing equivalence.

Having identified trace equivalence we now continue by exploring other possibilities. In the next section we try to take "invisible" or "silent" actions into account.

### 5.3.2 Weak Bisimulation

In this section we show that Milner's *weak bisimulation* [Mil89] can be characterised using the general setting of Sect. 5.2.

Weak bisimulation differs from strong bisimulation in at least two respects. First, a special "invisible" action, usually denoted $\tau$, is required to be a member of the set of labels. Second, an $\alpha$ labelled transition in one labelled transition system is no longer required to be simulated exactly by an $\alpha$ labelled transition in the other system. It may be preceded and succeeded by several $\tau$ transition. We write $r \stackrel{t}{\Longrightarrow} r'$ if $t = \alpha_1 \cdots \alpha_n$ and $r \stackrel{\tau^*}{\longrightarrow} r_1 \stackrel{\alpha_1}{\longrightarrow} r_1' \stackrel{\tau^*}{\longrightarrow} \cdots \stackrel{\tau^*}{\longrightarrow} r_n \stackrel{\alpha_n}{\longrightarrow} r_n' \stackrel{\tau^*}{\longrightarrow} r'$ for some $r_1, \ldots, r_n'$. Furthermore, a $\tau$ transition needn't be simulated by any transitions at all.

We start by defining a category $\mathcal{LTS}_2$, *labelled transition systems*, and a subcategory of observations, $\mathcal{P}$, in $\mathcal{LTS}_2$. Then, we show that $\mathcal{P}$-bisimilarity corresponds to Milner's weak bisimulation.

The objects of $\mathcal{LTS}_2$ are the same as those from $\mathcal{LTS}$. However, we assume that the set of actions *Act* contains a special "invisible" action $\tau$. Guided by our intuitive understanding of how an action may be simulated, we define the morphisms between two ltss as follows.

**Definition 84** Given two ltss, $T_j = (S_j, i_j, \mathrm{Act}, \longrightarrow_j), j = 1, 2$. A morphism between $T_1$ and $T_2$, $m : T_1 \longrightarrow T_2$, is a function $m$ from $S_1$ to $S_2$, such that

$$m(i_1) = i_2 \ , \tag{5.17}$$

$$r \stackrel{\alpha}{\longrightarrow} r' \text{ implies } m(r) \stackrel{\widehat{\alpha}}{\Longrightarrow} m(r') \ . \tag{5.18}$$

The function $\widehat{\ }$: $Act^* \longrightarrow Act^*$ removes all $\tau$'s from its argument [Mil90]. □

Composition of morphisms is defined as the usual composition of functions. This defines the category $\mathcal{LTS}_2$. $\mathcal{P}$, the category of observations, is defined as follows.

**Definition 85** Let $\mathcal{P}$ be the subcategory of $\mathcal{LTS}_2$ whose objects are of the form

$$i \xrightarrow{\alpha_1} r_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} r_n \ , \tag{5.19}$$

where all the states are distinct. Moreover, there will be a morphism $f$ from an observation

$$i \xrightarrow{\alpha_1} r_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} r_n \tag{5.20}$$

to another observation

$$i' \xrightarrow{\alpha_1} r'_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} r'_n \xrightarrow{\alpha_{n+1}} \cdots \xrightarrow{\alpha_{n+k}} r'_{n+k} \ , \tag{5.21}$$

if $f(i) = i'$, $f(r_j) = r'_j$ for $1 \leq j \leq n$, and $k \geq 0$.

□

Notice that morphisms between observations are required to simulate action in the "strong" sense—e.g., no additional $\tau$'s may be added. In the summary, Sect. 5.5, we will comment on this interesting choice. Allowing any morphism between two observations will in fact make $\mathcal{P}$-bisimilarity stronger than weak bisimulation; the reader should have no major difficulties in going through the proofs. Lemma 87 will no longer be true, neither will the "only if" in Theorem 89.

Having defined $\mathcal{M}$ as $\mathcal{LTS}_2$ and $\mathcal{P}$ we now show that $\mathcal{LTS}_2$ has pullbacks.

**Lemma 86** *The category $\mathcal{LTS}_2$ has pullbacks.*

**Proof.** Given a diagram

$$\begin{array}{ccc} T_1 & & T_2 \\ & {\scriptstyle m_1} \searrow \quad \swarrow {\scriptstyle m_2} & \\ & T & \end{array}$$

Define $T' = (S', i', Act, \longrightarrow')$ as follows. $S' \subseteq S_1 \times S_2$ and $\longrightarrow' \subseteq (S_1 \times S_2) \times Act \times (S_1 \times S_2)$ are the least sets such that

- $i' = (i_1, i_2) \in S'$
- If $(r, s) \in S'$, $r \stackrel{\widehat{\alpha}}{\Longrightarrow} r'$, $s \stackrel{\widehat{\alpha}}{\Longrightarrow} s'$, and $m_1(r') = m_2(s')$ then $(r', s') \in S'$ and $((r, s), \alpha, (r', s')) \in \longrightarrow'$.

Define $\pi_1 : T' \longrightarrow T_1$ and $\pi_2 : T' \longrightarrow T_2$ as $\pi_1((r, s)) = r$ and $\pi_2((r, s)) = s$.

We now show that this defines a pullback. Clearly, $\pi_1$ and $\pi_2$ are morphisms. Assume we have a commuting diagram

$$\begin{array}{ccc} T'' & \xrightarrow{g} & T_2 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle m_2} \\ T_1 & \xrightarrow[m_1]{} & T \end{array}$$

Define $h : S'' \longrightarrow S_1 \times S_2$ as $h(v) = (f(v), g(v))$ for $v \in S''$. It should be easy to see, using the commutativity of the diagram and the definition of $T'$, that $h(v) \in S'$, since $v$ is reachable from $i''$, and that $v \xrightarrow{\alpha} v'$ in $T''$ implies $h(v) \xrightarrow{\alpha} h(v')$ in $T'$. $h$ is then a well defined morphism from $T''$ to $T'$. Also, $f = \pi_1 \circ h$ and $g = \pi_2 \circ h$ and moreover these equations determine $h$ uniquely. Hence, $T'$, $\pi_1$, and $\pi_2$ constitute a pullback. ■

Next, we characterise the open maps.

**Lemma 87** *A morphism $m : T_1 \longrightarrow T_2$ is $\mathcal{P}$-open if and only if it satisfies the following "zig-zag" property:*

$$\textit{If } m(r) \xrightarrow{\alpha} s \textit{ then there exists an } r' \textit{ such that } r \stackrel{\widehat{\alpha}}{\Longrightarrow} r' \textit{ and } m(r') = s.$$

**Proof.** Assume $m$ is open and $i_1 \xrightarrow{\alpha_1} r_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} r_n = r$. Let $O_1$ be the observation

$$i \xrightarrow{\alpha_1} r'_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} r'_n \ ,$$

$O_2$ be the observation

$$i' \xrightarrow{\alpha_1} r''_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} r''_n \xrightarrow{\alpha} s'' \ ,$$

and $f$ the unique morphism from $O_1$ to $O_2$. Let $p : O_1 \longrightarrow T_1$ be the morphism which sends $r'_j$ to $r_j$ for $1 \leq j \leq n$, and $q : O_2 \longrightarrow T_2$ the morphism that sends $r''_j$ to $m(r_j)$ for $1 \leq j \leq n$ and $s''$ to s. Then $m \circ p = q \circ f$ and since $m$ is an open map there exists a morphism $h : O_2 \longrightarrow T_1$ such that the two triangles in the diagram

$$\begin{array}{ccc} O_1 & \xrightarrow{\ p\ } & T_1 \\ {\scriptstyle f}\downarrow & \swarrow_{h} & \downarrow{\scriptstyle m} \\ O_2 & \xrightarrow[\ q\ ]{} & T_2 \end{array}$$

commutes. Since $h(r''_n) \stackrel{\widehat{\alpha}}{\Longrightarrow} h(s'')$, $h(r''_n) = h(f(r'_n)) = p(r'_n) = r_n$, and $s = q(s'') = m(h(s''))$, we conclude that $h(r''_n) = r_n = r \stackrel{\widehat{\alpha}}{\Longrightarrow} h(s'')$ and $m(h(s'')) = s$. Hence, there exists a $r' = h(s'')$ such that $r \stackrel{\widehat{\alpha}}{\Longrightarrow} r'$ and $m(r') = s$.

For the other direction, assume $m$ has the "zig-zag" property and we are given a commuting diagram of the form

$$\begin{array}{ccc} O_1 & \xrightarrow{\ p\ } & T_1 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle m} \\ O_2 & \xrightarrow[\ q\ ]{} & T_2 \end{array}$$

where $O_1$ is an observation of the form

$$i \xrightarrow{\alpha_1} r_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} r_n \ ,$$

$O_2$ is an observation of the form

$$i' \xrightarrow{\alpha_1} r'_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} r'_n \xrightarrow{\alpha_{n+1}} \cdots \xrightarrow{\alpha_{n+k}} r'_{n+k} \ ,$$

and $f : O_1 \longrightarrow O_2$ is the uniquely determined morphism which sends $r_j$ to $r'_j$ for $1 \leq j \leq n$.

We show that there exists a morphism $h : O_2 \longrightarrow T_1$ such that $p = h \circ f$ and $q = m \circ h$. Apart from defining $h(r'_j) = p(r_j)$ for $1 \leq j \leq n$, and of course $h(i') = i_1$, let us consider which value we should give $h(r'_{n+1})$. Assume $q(r'_n) \stackrel{\widehat{\alpha_{n+1}}}{\Longrightarrow} q(r'_{n+1})$ because $q(r'_n) \xrightarrow{\beta_1} v_1 \xrightarrow{\beta_2} \cdots \xrightarrow{\beta_{l-1}} v_{l-1} \xrightarrow{\beta_l} q(r'_{n+1})$, where $l \geq 0$ and $\widehat{\alpha_{n+1}} = \beta_1 \widehat{\cdots} \beta_l$. By commutativity we have $m(p(r_n)) = q(r'_n)$ and by repeated use of the "zig-zag" property we conclude that there exist states $w_j$ such that $p(r_n) \stackrel{\widehat{\beta_1}}{\Longrightarrow} w_1 \stackrel{\widehat{\beta_2}}{\Longrightarrow} \cdots \stackrel{\widehat{\beta_l}}{\Longrightarrow} w_l$, $m(w_j) = v_j$ for $1 \leq j < l$, and $m(w_l) = q(r'_{n+1})$. Define $h(r'_{n+1}) = w_l$. Then $h(r'_n) \stackrel{\widehat{\alpha_{n+1}}}{\Longrightarrow} h(r'_{n+1})$ and $m(h(r'_{n+1})) = m(w_l) = q(r'_{n+1})$. Continuing this process for the remaining $r'_{n+2}, \ldots, r'_{n+k}$ it is easy to see that we obtain a morphism $h : O_2 \longrightarrow T_1$ such that $p = h \circ f$ and $q = m \circ h$. Hence, $m$ is an open map. ■

For the sake of completeness we give Milner's definition of weak bisimulation [Mil89], here adapted to the case where we consider initial states of lts's.

**Definition 88** Given two lts's $T_1$ and $T_2$. A relation $R \subseteq S_1 \times S_2$ is said to be a *weak bisimulation* over $T_1$ and $T_2$ if

$$(i_1, i_2) \in R \ , \tag{5.22}$$

$$((r,s) \in R \ \wedge \ r \xrightarrow{\alpha} r') \ \Rightarrow \ \text{for some } s', (s \stackrel{\widehat{\alpha}}{\Longrightarrow} s' \ \wedge \ (r', s') \in R) \ , \tag{5.23}$$

$$((r,s) \in R \ \wedge \ s \xrightarrow{\alpha} s') \ \Rightarrow \ \text{for some } r', (r \stackrel{\widehat{\alpha}}{\Longrightarrow} r' \ \wedge \ (r', s') \in R) \ . \tag{5.24}$$

$T_1$ and $T_2$ are said to be *weakly bisimilar* if there exists a weak bisimulation as defined above. □

We now show that $\mathcal{P}$-bisimilarity coincides with weak bisimulation.

**Theorem 89** *Given two lts's $T_1$ and $T_2$. Then,*

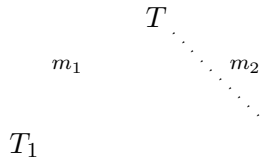
*$T_1$ and $T_2$ are weakly bisimilar if and only if $T_1$ and $T_2$ are $\mathcal{P}$-bisimilar.*

**Proof.** Assume $R$ is a weak bisimulation over $T_1$ and $T_2$. Define $T' = (S', i', Act, \longrightarrow')$, where $S' \subseteq S_1 \times S_2$, as follows. Let $S'$ and $\longrightarrow'$ be the least sets such that

- $i' = (i_1, i_2) \in S'$
- If $(r,s) \in S'$, $r \stackrel{\widehat{\alpha}}{\Longrightarrow} r'$, $s \stackrel{\widehat{\alpha}}{\Longrightarrow} s'$, and $(r', s') \in R$, then $(r', s') \in S'$ and $((r,s), \alpha, (r', s')) \in \longrightarrow'$.

Notice that $S' \subseteq R$. Now define $p : T' \longrightarrow T_1$ as $p((r,s)) = r$ and $q : T' \longrightarrow T_2$ as $q((r,s)) = s$. From the definitions and the above observation it should be easy to see that $p$ and $q$ are open maps, i.e., $T_1$ and $T_2$ are $\mathcal{P}$-bisimilar.

Assume $T_1$ and $T_2$ are $\mathcal{P}$-bisimilar, i.e., there exist a span of open maps




It is enough to show that $T$ and $T_1$ are weakly bisimilar since weak bisimulation is an equivalence relation. Define $R$ to be the least relation in $S \times S_1$ such that

- $(i, i_1) \in R$,
- If $(r,s) \in R$ and $r \xrightarrow{\alpha} r'$ then $(r', m_1(r')) \in R$, and
- If $(r,s) \in R$ and $s \xrightarrow{\alpha} s'$ then $(r', s') \in R$ where $r'$ is any state such that $r \stackrel{\widehat{\alpha}}{\Longrightarrow} r'$ and $m_1(r') = s'$. Such a state exists by Lemma 87.

Notice that $(r,s) \in R$ implies $m_1(r) = s$. Hence, in the last item $s = m_1(r) \stackrel{\widehat{\alpha}}{\Longrightarrow} m_1(r')$. It is now easy to show that $R$ is a weak bisimulation over $T$ and $T_1$. ■

*Remark.* It can be shown formally, that weak bisimulation cannot be captured as $\mathcal{P}$-bisimilarity for any choice of $\mathcal{P}$ in the category $\mathcal{LTS}$ from Sect. 5.2. It suffices to consider the two (weakly bisimilar) transition systems

$$1 \xrightarrow{\tau} \cdot \xrightarrow{a} \cdot \quad \text{and} \quad i' \xrightarrow{a} \cdot$$

### 5.3.3 Testing Equivalence

In this section we modify the category from Sect. 5.3.1 (basically) only with respect to the morphisms. We then choose a new subcategory $\mathcal{P}$ of observations. This time the elements of $\mathcal{P}$ will reflect a special type of branching structure. Then we show that the obtained $\mathcal{P}$-bisimilarity coincides with Hennessy's *testing equivalence* [Hen88]. Testing equivalence is slightly stronger than trace equivalence, due to an extra requirement on the set of possible actions, so-called *acceptance sets*, from states reached by performing a sequence of actions/labels.

We continue by defining a new category $\mathcal{LTS}_3$ of transition systems. The objects are those from $\mathcal{LTS}_1$ which are finitely branching, i.e., from every state only finitely many actions can be taken. Before defining the morphisms we need some definitions, inspired by [Hen88].

**Definition 90** Let $T = (S, i, Act, \longrightarrow)$ be an lts. Let $RS(T)$ denote the reachability set of $T$. For $r \in S$, $X \in RS(T)$, and $s \in Act^*$ let the *successors* of $r$ and $X$, respectively, be

$$S_T(r) = \{\alpha \in Act \mid \exists r'.\, r \xrightarrow{\alpha} r'\} \ , \tag{5.25}$$

$$S_T(X) = \{S_T(r) \mid r \in X\} \ , \tag{5.26}$$

let the *language* of $r$ be

$$L(r) = \{s \mid \exists r'.\, r \xrightarrow{s} r'\} \ , \tag{5.27}$$

and let the *acceptance sets of $r$ after $s$* be

$$\mathcal{A}_T(r,s) = \{S_T(r') \mid r \xrightarrow{s} r'\} \ . \tag{5.28}$$

Notice that if $\{i\} \xrightarrow{s} X$ for $s \in Act^*$ then $\{S_T(X)\} = \mathcal{A}_T(i,s)$. $S_T(X)$ is the *acceptance set of $X$*. □

Let $\mathcal{A}$ and $\mathcal{B}$ denote acceptance sets. We write $\mathcal{A} \subset\subset \mathcal{B}$ if for every $A \in \mathcal{A}$ there exists some $B \in \mathcal{B}$ such that $B \subseteq A$.

**Definition 91** Let $T = (S, i, Act, \longrightarrow)$ be an lts and $r, r' \in S$. Then,

$$r \ll_{MAY} r' \qquad \text{if} \qquad L(r) \subseteq L(r') \tag{5.29}$$

$$r \ll_{MUST} r' \qquad \text{if} \qquad \mathcal{A}_T(r,s) \subset\subset \mathcal{A}_T(r',s) \text{ for every } s \in Act^* \tag{5.30}$$

$$r \ll r' \quad \text{if both} \quad r \ll_{MAY} r' \text{ and } r \ll_{MUST} r' \tag{5.31}$$

□

The morphisms are now defined as follows.

**Definition 92** Given two ltss, $T_j = (S_j, i_j, Act, \longrightarrow_j)$, $j = 1, 2$. A morphism $m$ between $T_1$ and $T_2$ is a function $m$ from $RS(T_1)$ to $RS(T_2)$, such that

$$m(\{i_1\}) = \{i_2\} \ , \tag{5.32}$$

$$X \xrightarrow{\alpha} X' \text{ implies } m(X) \xrightarrow{\alpha} m(X') \ , \tag{5.33}$$

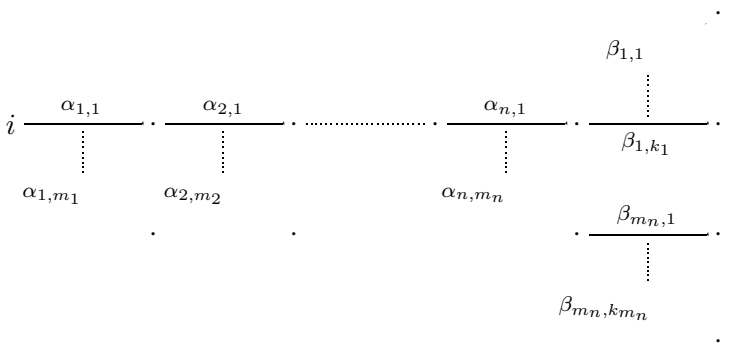
$$m(X) = Y \ \Rightarrow \ \forall A' \in S_{T_2}(Y).\, \exists A \in S_{T_1}(X).\, A \subseteq A' \ . \tag{5.34}$$

□

Notice how the definition of morphisms intuitively simulates Hennessy's $\ll_{MAY}$ and $\ll_{MUST}$ pre-orders. Being guided by the definitions in [Hen88] and our results from Sect. 5.3.1, (5.32) and (5.33) reflect that we want traces to be simulated, and (5.34) reflects how acceptance sets are to be matched. Composition of morphisms is defined as the usual composition of functions. This defines the category $\mathcal{LTS}_3$.

The subcategory $\mathcal{P}$ of observations will not consist of finite paths, but of trees consisting of a "trunk" and "branches" of length one, except for the "top" of the tree, where a more general branching structure is allowed.

**Definition 93** Let $\mathcal{P}$ be the full subcategory of $\mathcal{LTS}_3$ whose objects are of the form




where $0 \leq m_1, \ldots, m_n, k_1, \ldots, k_{m_n}$ and all states are distinct. □

Intuitively, the "trunk" corresponds to the observations in Sect. 5.3.1, i.e., it will ensure the existence of certain traces. The "top" of the the tree will ensure the existence of acceptance sets. The branches along the trunk are merely there for technical reasons. Think of a tree that has a trunk and only branches (of length one) at the top. Then allow branches of length one ("acceptance sets") to "grow" at any node. This will produce an observation in $\mathcal{P}$.

**Lemma 94** *The category $\mathcal{LTS}_3$ has pullbacks.*

**Proof.** Assume we are given a diagram

$$\begin{array}{ccc} & T_1 & \qquad\qquad T_2 \\ & {\scriptstyle m_1} \searrow \qquad \swarrow {\scriptstyle m_2} & \\ & T_0 & \end{array}$$

where $T_j = (S_j, i_j, Act, \longrightarrow_j)$, $j = 0, 1, 2$.

We start be defining an lts $T = (S, i, Act, \longrightarrow)$ as follows. $S$ will consist of triples whose first, second, and third components are elements from $RS(T_1)$, $RS(T_2)$, and subsets of $Act$, respectively. $S$ and $\longrightarrow$ are defined to be the least set such that

- $i = (\{i_1\}, \{i_2\}, S_{T_1}(i_1) \cap S_{T_2}(i_2)) \in S$
- If $(X, Y, C) \in S$, $\alpha \in C$, $X \xrightarrow{\alpha} X'$, and $Y \xrightarrow{\alpha} Y'$ then $(X', Y', C') \in S$ for all $C' \in M(X', Y')$, where $M(X', Y') = \{A' \cap (\bigcup S_{T_2}(Y')) \mid A' \in S_{T_1}(X')\} \cup \{B' \cap (\bigcup S_{T_1}(X')) \mid B' \in S_{T_2}(Y')\}$. Furthermore, $(X, Y, C) \xrightarrow{\alpha} (X', Y', C')$ for all $C' \in M(X', Y')$.

It should be easy to see that $T$ is an lts. For later use we notice the following facts.

- For $(X, Y, C) \in S$ it is the case that $S_T((X, Y, C)) = C$. This follows from the definition of $C$.

- It is the case that $L(T) = L(T_1) \cap L(T_2)$. To see this note that $L(T) \subseteq L(T_1) \cap L(T_2)$ clearly holds. So choose any $v \in L(T_1) \cap L(T_2)$, where $v = \alpha_1 \cdots \alpha_n$, $n \geq 0$. Then there exists $X_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} X_n$ in $T_1$ and $Y_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} Y_n$ in $T_2$, where $X_0 = \{i_1\}$ and $Y_0 = \{i_2\}$. Also, there must exists $A_j \in S_{T_1}(X_j)$ such that $\alpha_{j+1} \in A_j$ for $0 \leq j < n$ and clearly we have $\alpha_{j+1} \in \bigcup S_{T_2}(Y_j)$ for $0 \leq j < n$. So $(X_0, Y_0, A_0 \cap \bigcup S_{T_2}(Y_0)) \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} (X_n, Y_n, A_n \cap \bigcup S_{T_2}(Y_n))$. Hence, $v \in L(T)$.

- Given $Z \in RS(T)$. Then there exist an $X \in RS(T_1)$ and a $Y \in RS(T_2)$ such that $Z = \{(X, Y, C) \mid C \in M(X, Y)\}$. This follows from that fact that the transitions are deterministic on $RS(T_1)$ and $RS(T_2)$.

Let us define $\pi_1 : T \longrightarrow T_1$ as follows ($\pi_2$ is defined in a similar fashion). Given $Z \in RS(T)$ let $\pi_1(Z) = X$, where $X$ is the unique first component of the elements of $Z$. We now show that $\pi_1$ is a morphism.

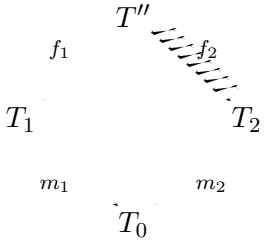
- Clearly $\pi_1(\{i\}) = \{i_1\}$.

- Assume $Z \xrightarrow{\alpha} Z'$, for $Z, Z' \in RS(T)$. Then by definition $\pi_1(Z) \xrightarrow{\alpha} \pi_1(Z')$.

- Assume $A \in S_{T_1}(X)$, where $X = \pi_1(Z)$. Then, $(X, Y, A \cap \bigcup S_{T_2}(Y)) \in Z$, where $\pi_2(Z) = Y$, by definition. Also, $S_T((X, Y, A \cap \bigcup S_{T_2}(Y)) = A \cap \bigcup S_{T_2}(Y) \subseteq A$. Hence, there exists an $C \in S_T(Z)$ such that $C \subseteq A$.

Having argued for that $\pi_1$ and $\pi_2$ are morphisms it trivially follows that $m_1 \circ \pi_1 = m_2 \circ \pi_2$.

Now assume that we are given a commuting square of the form




We will show that there exists a morphism $h : T'' \longrightarrow T$. We then necessarily have $f_1 = \pi_1 \circ f_1$ and $f_2 = \pi_2 \circ f_2$. Hence, we will have the desired pullback. Define $h : T'' \longrightarrow T$ by $h(V) = \{(X, Y, C) \mid C \in M(X, Y)\}$, where $X = f_1(V)$ and $Y = f_2(V)$.

- Clearly $h(\{i''\}) = i$.

- If $V \xrightarrow{\alpha} V'$ in $T''$ then $f_1(V) \xrightarrow{\alpha} f_1(V')$ and $f_2(V) \xrightarrow{\alpha} f_2(V')$, and hence, by previous facts we conclude that $h(V) \xrightarrow{\alpha} h(V')$.

- Assume $C \in S_T(Z)$, where $Z = h(V)$. By the previous facts we conclude that $(X, Y, C) \in Z$, where $f_1(V) = X$ and $f_2(V) = Y$. Without loss of generality we may assume $C$ is of the form $A \cap \bigcup S_{T_2}(Y)$ for $A \in S_{T_1}(X)$. Since $f_1$ is a morphism we know there exists a $D \in S_{T''}(V)$ such that $D \subseteq A$. Also, $D \subseteq \bigcup S_{T_2}(Y)$ because $f_2$ simulates the transitions from $V$ by transitions from $Y$. Hence, $D \subseteq A \cap \bigcup S_{T_2}(Y)$.

This shows that $h$ is a morphism and completes the proof of the lemma. ∎

Having defined $\mathcal{M}$ and $\mathcal{P}$ we now characterise the open maps.

**Lemma 95** *A morphism $m : T_1 \longrightarrow T_2$ is $\mathcal{P}$-open if and only if $m : RS(T_1) \longrightarrow RS(T_2)$ has the following "zig-zag" property*

$$\textit{If } m(X) \xrightarrow{\alpha} Y', \textit{ then there exists an } X' \textit{ such that } X \xrightarrow{\alpha} X' \textit{ and } m(X') = Y',$$

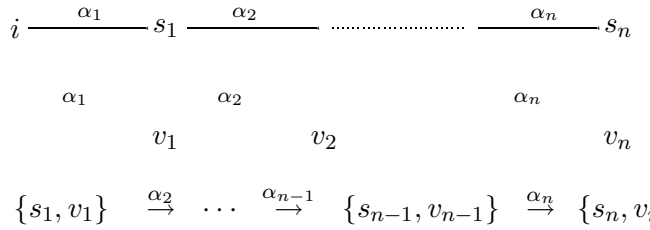
$$\textit{and } \forall A \in S_{T_1}(X).\, \exists A' \in S_{T_2}(m(X)).\, A' \subseteq A\ .$$

**Proof.** Assume $m$ is $\mathcal{P}$-open.

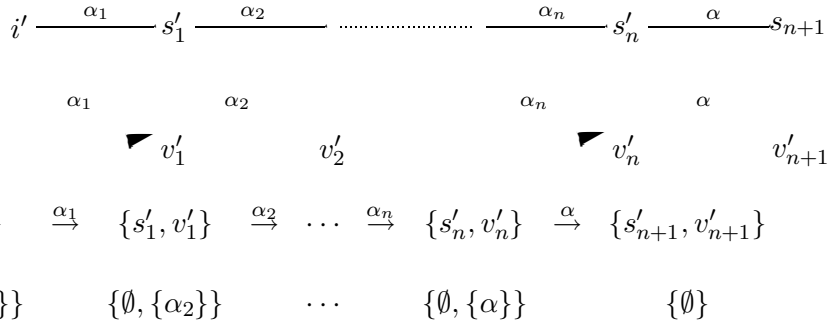
- If $m(X) \xrightarrow{\alpha} Y'$ then there exists

$$X_0 \xrightarrow{\alpha_1} X_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} X_n\ ,$$

with $\{i_1\} = X_0$ and $X = X_n$ in $T_1$. Let $Y_j = m(X_j)$ for $0 \leq j \leq n$ and $Y_{n+1} = Y'$. Let $O_1$ denote the following observation. For the sake of completeness we show the underlying deterministic transition system and right below the associated acceptance sets.




Let $O_2$ denote the following observation.



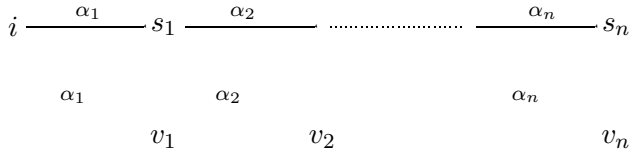

It should be easy to see that there exists a unique morphism from $O_1$ to $O_2$. Denote it $f : O_1 \longrightarrow O_2$. Also, defining $p : O_1 \longrightarrow T_1$ by $p(\{i\}) = \{i_1\}$ and $p(\{s_j, v_j\}) = X_j$ for $1 \leq j \leq n$ yields a morphism. Similarly, $q : O_2 \longrightarrow T_2$ defined by $q(\{i'\}) = \{i_2\}$ and $q(\{s'_j, v'_j\})$ for $1 \leq j \leq n+1$ is a morphism. Since there is at most one morphism between any two objects in $\mathcal{LTS}_3$ we conclude that $m \circ p = q \circ f$. Since $m$ is $\mathcal{P}$-open there exists a morphism $h : O_2 \longrightarrow T_1$ such that $p = h \circ f$ and $q = m \circ h$. Now $h(\{s'_n, v'_n\}) = h(f(\{s_n, v_n\})) =$

$p(\{s_n, v_n\}) = X_n$ and since $\{s'_n, v'_n\} \xrightarrow{\alpha} \{s'_{n+1}, v'_{n+1}\}$ we conclude $h(\{s'_n, v'_n\}) \xrightarrow{\alpha} h(\{s'_{n+1}, v'_{n+1}\})$. *Also*, $m(h(\{s'_{n+1}, v'_{n+1}\})) = q(\{s'_{n+1}, v'_{n+1}\}) = Y_{n+1}$. So we conclude that there exists an $X'$ such that $X \xrightarrow{\alpha} X'$ and $m(X') = Y'$.

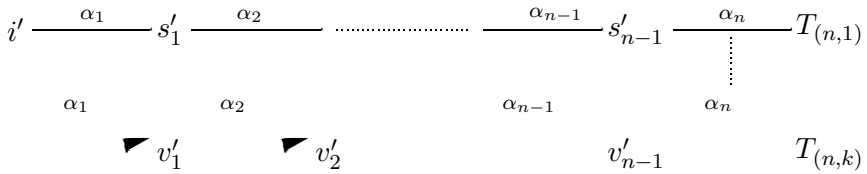
- Let $A \in S_{T_1}(X)$. We have to show that there exists an $A' \in S_{T_2}(m(X))$ such that $A' \subseteq A$. As before, assume

$$X_0 \xrightarrow{\alpha_1} X_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} X_n \ ,$$

where $\{i_1\} = X_0$ and $X = X_n$ in $T_1$, and let $Y_0 = m(X_0), \ldots, Y_n = m(X_n)$. Let $O_1$ denote the following observation.




Let $O_2$ denote




where $k = |\, S_{T_2}(Y)\,|$, and if $S_{T_2}(Y) = \{A_1, \ldots, A_k\}$ then $T_{(n,j)}$ denotes a tree of depth one, whose branches are labelled by $A_j$, for $1 \leq j \leq k$. For completeness have show the reachability sets and the associated acceptance sets.

$$\begin{array}{lllll} \{i'\} \xrightarrow{\alpha_1} \{s'_1, v'_1\} & \cdots & \{s'_{n-1}, v'_{n-1}\} \xrightarrow{\alpha_n} & \{i_{(n,1)}, \ldots, i_{(n,k)}\} & \cdots \quad X'_1 \\ & & & & \ddots \quad X'_k \\ \{\{\alpha_1\}\} & \{\emptyset, \{\alpha_2\}\} & \cdots \quad \{\emptyset, \{\alpha\}\} & S_{T_2}(Y) & \emptyset \end{array}$$

For the sake of clarity we have used $X'_1, \ldots, X'_k$ to denote the remaining reachability sets.

As before, it should be easy to see that there exists a unique morphism $f : O_1 \longrightarrow O_2$. Define $p : O_1 \longrightarrow T_1$ as before. Finally, define $q(\{i'\}) = Y_1$, $q(\{s'_j, v'_j\}) = Y_j$ for $1 \leq j \leq n-1$, $q(\{i_{(n,1)}, \ldots, i_{(n,k)}\}) = Y_n$, and since the transitions from $\{i_{(n,1)}, \ldots, i_{(n,k)}\}$ can all be matched by those of $Y_n$ ($S_{T_2}(Y) = S_{O_2}(\{i_{(n,1)}, \ldots, i_{(n,k)}\})$) it is possible to extend $q$ to a morphism $q : O_2 \longrightarrow T_2$. Notice how the requirements on the acceptance sets are fulfilled. As noted before, we then necessarily have $m \circ p = q \circ f$ and therefore the existence of a morphism $h : O_2 \longrightarrow T_1$ such that $p = h \circ f$ and $q = m \circ h$. Now the first equation tells us that $h(\{i_{(n,1)}, \ldots, i_{(n,k)}\}) = X_n$. Since $A \in S_{T_1}(X_n)$ there must exist an $A' \in S_{O_2}(\{i_{(n,1)}, \ldots, i_{(n,k)}\})$ such that $A' \subseteq A$. But since $S_{O_2}(\{i_{(n,1)}, \ldots, i_{(n,k)}\}) = S_{T_2}(Y_n) = S_{T_2}(m(X))$, we are done.

Assume $m$ has the "zig-zag" property. Given a commuting diagram

$$\begin{array}{ccc} O_1 & \xrightarrow{p} & T_1 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle m} \\ O_2 & \xrightarrow[q]{} & T_2 \end{array}$$

Since $m$ has the property that $m(X) \xrightarrow{\alpha} Y'$ implies there exists $X \xrightarrow{\alpha} X'$ such that $m(X') = Y'$, transitions on the reachability sets are deterministic, and $RS(O_1)$ together with the transitions has a tree structure, we can define $h : O_2 \longrightarrow T_1$ inductively as follows.

- $h(\{i'\}) = \{i_1\}$
- If $h(Y)$ has been defined, $Y \xrightarrow{\alpha} Y'$, and $h$ has not been defined on $Y'$ then define $h(Y')$ as follows. By induction we may assume $q(Y) = m(h(Y))$ and we know $q(Y) \xrightarrow{\alpha} q(Y')$. Then there must exist a (necessarily unique) $X'$ such that $h(Y) \xrightarrow{\alpha} X'$ and $m(X') = q(Y')$. Define $h(Y') = X'$. Notice also that for all $A' \in S_{T_1}(h(Y'))$ there exists an $A \in S_{O_2}(Y')$ such that $A \subseteq A'$. This follows from $m$ being $\mathcal{P}$-open and $q$ being a morphism; There must exist an $A'' \in S_{T_2}(m(h(Y'))) = S_{T_2}(q(Y'))$ such that $A'' \subseteq A'$ and also an $A \in S_{O_2}(Y')$ such that $A \subseteq A''$.

Checking that $h : O_2 \longrightarrow T_1$ is a morphism is now routine work. The equations $p = h \circ f$ and $q = m \circ h$ now follows trivially. ■

We continue by defining Hennessy's testing equivalence.

**Definition 96** Given two ltss, $T_j = (S_j, i_j, Act, \longrightarrow_j), j = 1, 2$. $i_1$ is said to be *testing equivalent* to $i_2$ if

$$L(T_1) = L(T_2) \ , \tag{5.35}$$

and for any $s \in L(T_1)$

$$\forall A' \in \mathcal{A}(i_2, s).\, \exists A \in \mathcal{A}(i_1, s).\, A \subseteq A' \ , \tag{5.36}$$

$$\forall A \in \mathcal{A}(i_1, s).\, \exists A' \in \mathcal{A}(i_2, s).\, A' \subseteq A \ . \tag{5.37}$$

□

The above definition follows from Definition 2.8.8 in [Hen88]. Using the notation from [Hen88] the above definition can be rewritten to $i_1 <<_{MAY} i_2$, $i_2 <<_{MAY} i_1$, $i_1 <<_{MUST} i_2$, and $i_2 <<_{MUST} i_1$.

With the given choice of $\mathcal{M}$ and $\mathcal{P}$ it turns out that testing equivalence corresponds to $\mathcal{P}$-bisimilarity.

**Theorem 97** *Given two* lts*s $T_1$ and $T_2$. Then,*

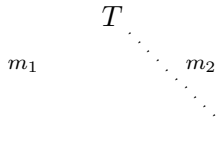
*$T_1$ and $T_2$ are testing equivalent if and only if $T_1$ and $T_2$ are $\mathcal{P}$-bisimilar.*

**Proof.** Assume $T_1$ and $T_2$ are testing equivalent. Define $T = (S, i, Act, \longrightarrow)$, $\pi_1$ and $\pi_2$ as in the proof of Lemma 94. By symmetry it is enough to show that $\pi_1$ is $\mathcal{P}$-open.

Let $X = \pi_1(Z)$ for $Z \in RS(T)$ and assume $X \xrightarrow{\alpha} X'$. Assume $X_0 \xrightarrow{\alpha_1} X_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} X_n \xrightarrow{\alpha} X'$, where $X_0 = \{i_1\}$ and $X_n = X$. Since $L(T_1) = L(T_2)$ we know that there exist $Y_0 \xrightarrow{\alpha_1} Y_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} Y_n \xrightarrow{\alpha} Y'$ in $T_2$, where $Y_0 = \{i_2\}$. Then necessarily $Z \xrightarrow{\alpha} Z'$, where $Z' = \{(X', Y', C) \mid C \in M(X', Y')\}$ and $\pi_1(Z') = X'$.

Let $C \in S_T(Z)$, where $Z \in RS(T)$. We conclude that $(X, Y, C) \in Z$, where $Y = \pi_2(Z)$. There exists a $v \in L(T)$ such that $\{i_1\} \xrightarrow{v} X$ and $\{i_2\} \xrightarrow{v} Y$. Without loss of generality assume $C$ is of the form $A \cap \bigcup S_{T_2}(Y)$, where $A \in S_{T_1}(X)$, Consider any minimal (with respect to set inclusion) $A_{min} \in S_{T_1}(X)$ such that $A_{min} \subseteq A$. Since $S_{T_1}(X) = \mathcal{A}(i_1, v) = \mathcal{A}(i_2, v) = S_{T_2}(Y)$, we conclude $A_{min} \in S_{T_2}(Y)$ and hence, $A_{min} \subseteq \bigcup S_{T_2}(U)$. Hence, $A_{min} \subseteq C$, i.e., there exists an $A' \in S_{T_1}(X)$ such that $A' \subseteq C$.

Next, assume $T_1$ and $T_2$ are $\mathcal{P}$-bisimilar, i.e., there exists a span of open maps




It is enough to show that $T$ and $T_1$ are testing equivalent. By Lemma 95 it follows easily that $L(T) = L(T_1)$. Given an $s \in L(T)$. Assume $A \in \mathcal{A}(i, s)$. We have to show that there exists an $A_1 \in \mathcal{A}(i_1, s)$ such that $A_1 \subseteq A$. Let $X$ be the unique element in $RS(T)$ such $\{i\} \xrightarrow{s} X$. Then $A \in S_T(X)$. Since $m_1$ is an open map there exists an $A_1 \in S_{T_1}(m(X))$ such that $A_1 \subseteq A$. Since $\mathcal{A}(i_1, s) = S_{T_1}(m(X))$ we are done.

Now assume $A_1 \in \mathcal{A}(i_1, s)$. We have to show that there exists an $A \in \mathcal{A}(i, s)$ such that $A \subseteq A_1$. Let $X_1$ be the unique element from $RS(T_1)$ such that $\{i_1\} \xrightarrow{s} X_1$. Then $A_1 \in S_{T_1}(X_1)$. Since $m_1$ is open there exists an $X \in RS(T)$ such that $\{i\} \xrightarrow{s} X$ and $m_1(X) = X_1$. By the definition of morphisms there exists an $A \in S_T(X)$ such that $A \subseteq A_1$. But since $S_T(X) = \mathcal{A}(i, s)$ we are done. Hence, $T_1$ and $T_2$ are testing equivalent. ∎

### 5.3.4 Barbed Bisimulation

In this section we show how we can obtain Milner and Sangiorgi's *barbed bisimulation* [MS92].

Barbed bisimulation differs from strong bisimulation in three obvious ways. First, as in the case of weak bisimulation, we distinguish between "visible" and "invisible" actions. Second, only $\tau$ transitions are required to be (bi)simulated. And third, only the *existence* of a "visible" transition has to be matched.

We start by defining the category of models $\mathcal{LTS}_4$, then the subcategory of observations $\mathcal{P}$, and finally we characterise the $\mathcal{P}$-open maps and prove that $\mathcal{P}$-bisimilarity coincides with barbed bisimulation.

Let $\mathcal{LTS}_4$ be the category of *labelled transition systems* (lts) whose objects are those from $\mathcal{LTS}$. Again we assume that *Act* contains a special "invisible" action denoted $\tau$. Before defining the morphisms of $\mathcal{LTS}_4$ we need the following definition.

**Definition 98** Given an lts $T = (S, i, Act, \longrightarrow)$. $R_\tau(T)$, the set of *$\tau$-reachable states of* $T$, is defined to be the set $\{s \in S \mid i \xrightarrow{\tau} \cdots \xrightarrow{\tau} s\}$. We use the notation $s\downarrow$ if there exist an $\alpha \in Act - \{\tau\}$ and an $s' \in S$ such that $s \xrightarrow{\alpha} s'$.

□

Morphisms between two lts in $\mathcal{LTS}_4$ are defined as follows:

**Definition 99** Given two ltss, $T_1 = (S_1, i_1, Act, \longrightarrow_1)$ and $T_2 = (S_2, i_2, Act, \longrightarrow_2)$. A morphism $m$ from $T_1$ to $T_2$ is a function $m : R_\tau(T_1) \longrightarrow R_\tau(T_2)$ such that

$$m(i_1) = i_2 \ , \tag{5.38}$$

$$r \xrightarrow{\tau} r' \text{ implies } m(r) \xrightarrow{\tau} m(r') \ , \tag{5.39}$$

$$r\downarrow \text{ implies } m(r)\downarrow \ . \tag{5.40}$$

This definition reflects that we only want to simulate $\tau$'s and preserve the $\downarrow$ predicate. This corresponds well to our intuitive understanding of barbed (bi)simulation. Composition of morphisms is defined as the composition of the functions between the underlying $\tau$-reachable states. □

Next we define the category of observations.

**Definition 100** Let $\mathcal{P}$ be the subcategory of $\mathcal{LTS}_4$ whose objects are of the form

$$i \xrightarrow{\tau} r_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} r_{n-1} \xrightarrow{\alpha} r_n \ , \tag{5.41}$$

where $\alpha \in Act$ and all states are distinct. □

Notice how the observations correspond to the $\tau$-reachability of a state ($r_n$, when $\alpha = \tau$) and the $\downarrow$ predicate holding at it ($r_{n-1}$, when $\alpha \neq \tau$).

Having defined $\mathcal{M}$ as $\mathcal{LTS}_4$ and $\mathcal{P}$ we show that $\mathcal{LTS}_4$ has pullbacks.

**Lemma 101** *The category $\mathcal{LTS}_4$ has pullbacks.*

**Proof.** Given a diagram

$$\begin{array}{ccc} T_1 & & T_2 \\ & {\scriptstyle m_1} \searrow \quad \swarrow {\scriptstyle m_2} & \\ & T & \end{array}$$

Define $S' \subseteq R_\tau(T_1) \times R_\tau(T_2)$ and $\longrightarrow'$ as the least sets such that

- $(i_1, i_2) \in S'$
- If $(r,s) \in S'$, $r \xrightarrow{\tau} r'$, $s \xrightarrow{\tau} s'$, and $m_1(r') = m_2(s')$ then $(r',s') \in S'$ and $((r,s),\tau,(r',s')) \in \longrightarrow'$.

Intuitively, $S'$ is the $\tau$-reachable states of the lts that will constitute the pullback. Now choose any $\alpha \in Act - \{\tau\}$ and define

- $S'' = \{(r',s') | \exists (r,s) \in S'.\, r \xrightarrow{\beta} r' \ \wedge\ s \xrightarrow{\gamma} s' \ \wedge\ \beta \neq \tau \ \wedge\ \gamma \neq \tau\}$
- $\longrightarrow'' = \{((r,s),\alpha,(r',s')) \mid (r,s) \in S' \ \wedge\ (r',s') \in S'' \wedge$ $\exists \beta \neq \tau, \gamma \neq \tau.\, r \xrightarrow{\beta} r' \ \wedge s \xrightarrow{\gamma} s'\}$

$S''$ and $\longrightarrow''$ ensure that the $\downarrow$ predicate has the desired value at the states in $S'$. Now define $T' = (S' \cup S'', (i_1,i_2), \longrightarrow' \cup \longrightarrow'', Act)$, $\pi_1 : T' \longrightarrow T_1$ as $\pi_1((r,s)) = r$, and $\pi_2 : T' \longrightarrow T_2$ as $\pi_2((r,s)) = s$. It can be shown that $T'$, $\pi_1$, and $\pi_2$ constitute a pullback of the above diagram. ■

Next, we characterise the open maps.

**Lemma 102** *A morphism $m : T_1 \longrightarrow T_2$ is $\mathcal{P}$-open if and only if it satisfies the following "zig-zag" property:*

$$\textit{If } m(r) \xrightarrow{\tau} s \textit{ then there exists an } r' \textit{ such that } r \xrightarrow{\tau} r' \textit{ and } m(r') = s.$$

$$\textit{Also, } m(r)\downarrow \textit{ implies } r\downarrow.$$

**Proof.** Assume $m$ is open, $r \in R_\tau(T_1)$, and $m(r) \xrightarrow{\tau} s$. We first prove the existence of the above mentioned $r'$. Assuming $i_1 \xrightarrow{\tau} r_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} r_n = r$, let $O_1$ be the observation

$$i \xrightarrow{\tau} s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_n\ ,$$

and $O_2$ the observation

$$i' \xrightarrow{\tau} s'_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s'_n \xrightarrow{\tau} s'\ .$$

Let $f : O_1 \longrightarrow O_2$ be the unique function from $O_1$ to $O_2$ which sends $s_j$ to $s'_j$, $p : O_1 \longrightarrow T_1$ the morphism which sends $s_j$ to $r_j$ for $1 \leq j \leq n$, and $q : O_2 \longrightarrow T_2$ the morphism which sends $s'_j$ to $m(r_j)$, for $1 \leq j \leq n$, and $s'$ to $s$. Then the diagram

$$\begin{array}{ccc} O_1 & \xrightarrow{\ p\ } & T_1 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle m} \\ O_2 & \xrightarrow[\ q\ ]{} & T_2 \end{array}$$

commutes, and there exists a morphism $h : O_2 \longrightarrow T_1$ such that $p = h \circ f$ and $q = m \circ h$. Observe that $h(s'_n) = h(f(s_n)) = p(s_n) = r_n$. Hence, $r = r_n = h(s'_n) \xrightarrow{\tau} h(s')$. Also, $m(h(s')) = q(s') = s$. Now choose $r' = h(s')$. Finally, assume $m(r)\downarrow$ because

$m(r) \xrightarrow{\alpha} s$ for $\alpha \neq \tau$. Choosing $O_2$ as $i' \xrightarrow{\tau} s'_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s'_n \xrightarrow{\alpha} s'$ and the other components as before (adjusting the morphisms in the obvious way) we obtain another commuting diagram and a morphism $h' : O_2 \longrightarrow T_1$ such that the triangles commute. So $h'(s'_n) = h'(f(s_n)) = p(s_n) = r_n = r$ and since $s'_n\downarrow$ we have $r\downarrow$.

Assume $m$ has the "zig-zag" property and the diagram

$$\begin{array}{ccc} O_1 & \xrightarrow{\;p\;} & T_1 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle m} \\ O_2 & \xrightarrow[\;q\;]{} & T_2 \end{array}$$

commutes. We proceed by a case analysis on the shape of the observations $O_1$ and $O_2$.

- Assume $O_1$ is of the form $i \xrightarrow{\tau} s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_n$ and $O_2$ of the form $i' \xrightarrow{\tau} s'_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s'_n \xrightarrow{\tau} \cdots \xrightarrow{\tau} s'_{n+k}$, where $k \geq 0$.

  Define $h(s'_j) = p(s_j)$ for $1 \leq j \leq n$. Next, we define $h$ on $s'_{n+1}$. Since $q(s'_n) = q(f(s_n)) = m(p(s_n))$, $p(s_n) \in R_\tau(T_1)$, and $m(p(s_n)) \xrightarrow{\tau} q(s'_{n+1})$ there exists a $r'$ such that $p(s_n) \xrightarrow{\tau} r'$ and $m(r') = q(s'_{n+1})$. Define $h(s'_{n+1}) = r'$. Continuing this way for the remaining state $s'_{n+2}, \ldots, s'_{n+k}$ it can be shown that the produced $h$ is a morphism from $O_2$ to $T_1$ such that $p = h \circ f$ and $q = m \circ h$.

- Assume $O_1$ is of the form $i \xrightarrow{\tau} s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_{n-1} \xrightarrow{\alpha} s_n$ and $O_2$ of the form $i' \xrightarrow{\tau} s'_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s'_{m-1} \xrightarrow{\beta} s'_m$, where $\alpha \neq \tau$ and $\beta \neq \tau$.

  Since $f$ is a morphism we conclude $n = m$ and $f(s_j) = s'_j$ for $1 \leq j \leq n$. It is then easy to see that $h(s'_j) = p(s_j)$ for $1 \leq j \leq n$ defines a morphism from $O_2$ to $T_1$ such that $p = h \circ f$ and $q = m \circ h$.

- Finally assume $O_1$ is of the form $i \xrightarrow{\tau} s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_n$ and $O_2$ of the form $i' \xrightarrow{\tau} s'_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s'_{m-1} \xrightarrow{\alpha} s'_m$, where $\alpha \neq \tau$.

  As before, we conclude $m > n$. Using the same procedure as in case 1, only this time observing that $m(h(s'_{m-1})) = q(s'_{m-1})\downarrow$ implies $h(s'_{m-1})\downarrow$, we obtain a morphism $h : O_2 \longrightarrow T_1$ such that the two triangles in the diagram commute.

We conclude that $m$ is $\mathcal{P}$-open. ∎

We now give Milner and Sangiorgi's definition of barbed bisimulation adapted to the case where the labelled transition systems have initial states.

**Definition 103** Given $T_1$ and $T_2$. A *barbed bisimulation over $T_1$ and $T_2$* is a relation $R \subseteq R_\tau(T_1) \times R_\tau(T_2)$ such that

$$(i_1, i_2) \in R \;, \tag{5.42}$$

$$((r, s) \in R \;\wedge\; r \xrightarrow{\tau} r') \;\Rightarrow\; \text{for some } s' \; (s \xrightarrow{\tau} s' \;\wedge\; (r', s') \in R) \;, \tag{5.43}$$

$$((r,s) \in R \ \wedge \ s \xrightarrow{\tau} s') \ \Rightarrow \ \text{for some } r' \ (r \xrightarrow{\tau} r' \ \wedge \ (r',s') \in R) \ , \tag{5.44}$$

$$(r,s) \in R \ \Rightarrow \ (r\downarrow \Leftrightarrow \ s\downarrow) \ . \tag{5.45}$$

$T_1$ and $T_2$ are said to be *barbed bisimilar* if there exists such an $R$. □

And now to the theorem relating $\mathcal{P}$-bisimilarity to barbed bisimilarity.

**Theorem 104** *Given two* lts*s, $T_1$ and $T_2$. Then:*

*$T_1$ is barbed bisimilar to $T_2$ if and only if $T_1$ and $T_2$ are $\mathcal{P}$-bisimilar.*

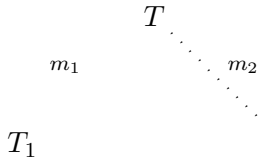
**Proof.** Assume $R$ is a barbed bisimulation over $T_1$ and $T_2$ and $\gamma \in Act - \{\tau\}$. Now define

- $S' = \{(r,s) \in R \,|\, \exists (r_1,s_1),\ldots,(r_{n-1},s_{n-1}) \in R.$
  $i_1 \xrightarrow{\tau} r_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} r_{n-1} \xrightarrow{\tau} r \ \wedge$
  $i_2 \xrightarrow{\tau} s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_{n-1} \xrightarrow{\tau} s\}$,
- $\longrightarrow' = \{((r,s),\tau,(r',s')) \,|\, (r,s),(r',s') \in S' \ \wedge \ r \xrightarrow{\tau} r' \ \wedge \ s \xrightarrow{\tau} s'\}$,
- $S'' = \{(r',s') \in S_1 \times S_2 \,|\, \exists (r,s) \in S', \alpha,\beta \in Act - \{\tau\}. r \xrightarrow{\alpha} r' \ \wedge \ s \xrightarrow{\beta} s'\}$,
- $\longrightarrow'' = \{((r,s),\gamma,(r',s')) \,|\, (r,s) \in S' \ \wedge \ (r',s') \in S'' \wedge$
  $\exists \alpha,\beta \in Act - \{\tau\}. r \xrightarrow{\alpha} r' \ \wedge \ s \xrightarrow{\beta} s'\}$,
- $T = (S' \cup S'', (i_1,i_2), Act, \longrightarrow' \cup \longrightarrow'')$,
- $\pi_1 : T \longrightarrow T_1$ by $\pi_1((r,s)) = r$, and
- $\pi_2 : T \longrightarrow T_2$ by $\pi_1((r,s)) = s$.

It can be shown that $T$ is a lts and that $\pi_1$ and $\pi_2$ are morphisms. Without loss of generality we show that $\pi_1$ is open. Choose any $(r,s) \in R_\tau(T)$.

- Assume $\pi_1((r,s)) = r \xrightarrow{\tau} r'$. Since $R_\tau(T) = S' \subseteq R$ and $R$ was a barbed bisimulation over $T_1$ and $T_2$, we know that there exists a $s'$ such that $s \xrightarrow{\tau} s'$ and $(r',s') \in R$. But by definition of $S'$ we conclude $(r',s') \in S'$ and also $(r,s) \xrightarrow{\tau} (r',s')$ in $T$, where $\pi_1((r',s')) = r'$.
- Assume that $\pi_1((r,s)) = r\downarrow$, i.e., there exists an $r'$ such that $r \xrightarrow{\alpha} r'$ and $\alpha \neq \tau$. Since $(r,s) \in R_\tau(T)$ we know that $r\downarrow \Rightarrow s\downarrow$, i.e., there exists $s'$ such that $s \xrightarrow{\beta} s'$ and $\beta \neq \tau$. But then $(r',s') \in S''$ and $((r,s),\gamma,(r',s')) \in \longrightarrow''$, i.e., $(r,s)\downarrow$.

Assume $T_1$ and $T_2$ are $\mathcal{P}$-bisimilar. Then there must exist a span of open maps:



It is sufficient to show that if $m_1 : T \longrightarrow T_1$ ($m_2 : T \longrightarrow T_2$) is open then $T$ and $T_1$ ($T_2$) are barbed bisimilar, since being barbed bisimilar is an equivalence relation. So define $R_1 = \{(r, m_1(r)) \,|\, r \in R_\tau(T)\}$. We claim that $R_1$ is a barbed bisimulation over $T$ and $T_1$, namely;

- $(i, i_1) \in R_1$,
- $(r, s) \in R_1$ and $r \xrightarrow{\tau} r'$ implies $s = m_1(r) \xrightarrow{\tau} m_1(r')$ and $(r', m_1(r')) \in R_1$,
- if $(r, s) = (r, m_1(r)) \in R_1$ and $m_1(r) \xrightarrow{\tau} s'$ then there exists an $r'$ such that $r \xrightarrow{\tau} r'$ and $m_1(r') = s'$, since $m_1$ is open, and $(r', s') = (r', m_1(r')) \in R_1$, and
- if $(r, s) \in R_1$ and $r\downarrow$ then $s = m_1(r)\downarrow$ since $m_1$ is a morphism and if $s\downarrow$ then $r\downarrow$ since $m_1(r) = s$, $r \in R_\tau(T)$, and $m_1$ is open.

This concludes the proof. ■

### 5.3.5 Probabilistic Transition Systems

In this section we show that the *probabilistic bisimulation* of Larsen and Skou [LS91] can be characterised using the general setting in Sect. 5.2. We will however apply the theory in a slightly different way. Until now, we have tried to characterise $\mathcal{P}$-bisimilarity between objects of $\mathcal{M}$, for the specific choices of $\mathcal{P}$ and $\mathcal{M}$. In this section we will focus on $\mathcal{P}$-bisimilarity between objects of a subcategory of $\mathcal{M}$. This application of the theory of open maps still turns out "successful".

Intuitively, Larsen and Skou's *probabilistic bisimulation* differs from strong bisimulation in at least two respects. First, to each labelled transition there is associated a real number from the interval $[0; 1]$ which is to be understood as the probability with which the transition can be performed. Second, it is no longer single labelled transitions between two states that have to be matched but a set of identically labelled transitions into an equivalence class of probabilistic bisimilar states.

Based on [LS91] we start by defining a category $\mathcal{PPTS}$, *partial probabilistic transition systems*, corresponding to $\mathcal{M}$, and a subcategory of observations, $\mathcal{P}$, in $\mathcal{PPTS}$. Then, we show that $\mathcal{P}$-bisimilarity in the full subcategory of *probabilistic transition systems*, $\mathcal{PTS}$, in $\mathcal{PPTS}$, corresponds to Larsen and Skou's probabilistic bisimulation. Contrary to Larsen and Skou we do not assume lower limit on the probability of transitions. Because we wish to allow arbitrary small probabilities and for technical reasons, we consider $\mathbb{R}^*$, the field of hyper-real numbers, instead of $\mathbb{R}$, the field of real numbers. $\mathbb{R}^*$ is the proper ordered extension of $\mathbb{R}$ containing infinitesimals. An element $\epsilon \in \mathbb{R}^*$ is infinitesimal if $0 < |\epsilon| < r$ for all positive real numbers $r$. We reserve the symbol $\epsilon$ to denote infinitesimals. For a thorough presentation the reader is referred to [Kei76].

**Definition 105** A *partial probabilistic transition system* (ppts) is a tuple

$$T = (Pr, i, Act, Can, \mu) \ , \tag{5.46}$$

where $Pr$ is a set of *processes* (or states), $i$ is the *initial state*, $Act$ is the set of *observable actions* that processes may perform, $Can$ is an $Act$-indexed family of sets of processes, and $\mu$ is a family of *partial probability distributions* indexed by states and actions. For any action $a \in Act$ and any process $r \in Pr$, $r \in Can_a$ indicates that the process $r$ can perform an $a$-action, in which case $\mu_{r,a} : Pr \to [0;1] \subseteq \mathbb{R}^*$ is a function such that $\sum_{r'} \mu_{r,a}(r') \leq 1$. □

In general, we do not require the sum to be equal to 1; hence the name *partial* probability distribution. If all $\mu_{r,a}$ are probability distributions, i.e., $\mu_{r,a}$ maps into the real numbers and $\sum_{r'} \mu_{r,a}(r') = 1$, we leave out the term *partial* and refer to $T$ as a *probabilistic transition system* (pts). $\mu_{r,a}(r') = \mu$ can intuitively be read as "$r$ can perform the action $a$ and with probability $\mu$ become $r'$".

Given a ppts T. We shall use the following notations:

| | | |
|---|---|---|
| $r \xrightarrow{a}_{\mu} r'$ | whenever | $r \in Can_a$ and $\mu_{r,a}(r') = \mu$ |
| $r \xrightarrow{a} r'$ | whenever | $r \xrightarrow{a}_{\mu} r'$ for some $\mu > 0$ |
| $r \xrightarrow{a}$ | whenever | $r \in Can_a$ |
| $r \not\xrightarrow{a}$ | whenever | $r \notin Can_a$ |
| $r \xrightarrow{a}_{\mu} S$ | whenever | $S$ is *any* set of processes, $r \in Can_a$ and $\mu = \sum_{r' \in S} \mu_{r,a}(r')$. |

We assume the set $Act$ to be fixed and that all processes in $Pr$ are reachable from the initial state via transitions having non-zero probabilities. Finally, two ppts will be said to be *distinct* if their sets of processes are disjoint.

Next, we define morphisms between pptss.

**Definition 106** A ppts-*morphism* between two pptss, $T_j = (Pr_j, i_j, Act, Can_j, \mu_j)$, $j = 1, 2$, is a function $f$ between $Pr_1$ and $Pr_2$ such that

$$f(i_1) = i_2 \ , \tag{5.47}$$

$$f(r) \xrightarrow{a} f(r') \text{ whenever } r \xrightarrow{a} r' \ , \tag{5.48}$$

$$\text{If } r \xrightarrow{a} r' \text{ and } f(r) \xrightarrow{a}_{\mu'} f(r') \text{ then } \sum_{r \xrightarrow{a}_{\mu} r'' \, f(r'') = f(r')} \mu \leq \mu' \ . \tag{5.49}$$

□

The intuition behind (5.49) is that all transitions from $r$ in $T_1$ which are simulated by a transition from $f(r)$ can occur with a probability which is no higher than the probability of the simulating transition from $f(r)$.

Let $\mathcal{PPTS}$ denote the category of partial probabilistic transition systems, whose objects are ppts's and morphisms are ppts-*morphisms*, with composition of morphisms defined as the usual composition of functions. Let $\mathcal{PTS}$ denote the full subcategory of $\mathcal{PPTS}$ whose objects are ptss.

In our model of computation, $\mathcal{PPTS}$, we identify the following subcategory $\mathcal{P}$ of observations.

**Definition 107** Let $\mathcal{P}$ be the full subcategory of $\mathcal{PPTS}$ whose objects are pptss of the following form

$$i \xrightarrow{a_1}_{\epsilon_1} r_1 \xrightarrow{a_2}_{\epsilon_2} \cdots \xrightarrow{a_n}_{\epsilon_n} r_n \ , \tag{5.50}$$

for some natural number $n$, distinct states, and actions $a_1, \ldots, a_n \in Act$. Notice that all the probabilities are infinitesimals. □

Since we will choose only to observe behaviours in ptss, using only infinitesimals on the transitions of the observations will intuitively enable us to observe whether or not a transition can occur, rather than the specific probability with which is occurs.

This time, we postpone the investigation of the existence of pullbacks in $\mathcal{PPTS}$. Instead, we now try to characterise the $\mathcal{P}$-open maps in $\mathcal{PPTS}$ between any two ptss.

**Lemma 108** *A morphism $m : T_1 \longrightarrow T_2$ between two ptss is $\mathcal{P}$-open if and only if it is "zig-zag" in the following sense:*

$$\textit{If } m(r) \xrightarrow{a} s \textit{ then there exists an } r' \textit{ such that } r \xrightarrow{a} r' \textit{ and } m(r') = s.$$

**Proof.** Assume $m$ is $\mathcal{P}$-open and $m(r) \xrightarrow{a} s$. Since $r$ is reachable from $i_1$ there exists

$$i_1 = r_0 \xrightarrow{a_1} r_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} r_n = r$$

in $T_1$. Let $O_1$ be any observation of the form

$$i \xrightarrow{a_1}_{\epsilon_1} s_1 \xrightarrow{a_2}_{\epsilon_2} \cdots \xrightarrow{a_n}_{\epsilon_n} s_n \ ,$$

and $O_2$ any observation of the form

$$i' \xrightarrow{a_1}_{\epsilon'_1} s'_1 \xrightarrow{a_2}_{\epsilon'_2} \cdots \xrightarrow{a_n}_{\epsilon'_n} s'_n \xrightarrow{a}_{\epsilon'} s' \ ,$$

where $\epsilon_j \leq \epsilon'_j$ for $1 \leq j \leq n$. Let $h$ denote the unique morphism between $O_1$ and $O_2$, $f$ denote the morphism from $O_1$ to $T_1$ which maps $s_j$ into $r_j$ for $1 \leq j \leq n$, and $g$ denote the morphism from $O_2$ to $T_2$ which maps $O_2$ into

$$i_2 = m(r_0) \xrightarrow{a_1} m(r_1) \xrightarrow{a_2} \cdots \xrightarrow{a_n} m(r_n) = m(r) \xrightarrow{a} s \ .$$

It is easy to see, that the square

$$\begin{array}{ccc} O_1 & \xrightarrow{f} & T_1 \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle m} \\ O_2 & \xrightarrow[g]{} & T_2 \end{array}$$

commutes. But since $m$ is $\mathcal{P}$-open there exists a morphism $m' : O_2 \longrightarrow T_1$ such that $m' \circ h = f$ and $m \circ m' = g$. This implies the existence of an $r'$ such that $r \xrightarrow{a} r'$ and $m(r') = s$.

Conversely, assume that $m$ is "zig-zag". Let $O_1$ and $O_2$ be any two observations and $f$, $g$, and $h$ any morphisms such that the square

$$\begin{array}{ccc} O_1 & \xrightarrow{\quad f \quad} & T_1 \\ {\scriptstyle h}\big| & & \big|{\scriptstyle m} \\ O_2 & \xrightarrow[\quad g \quad]{} & T_2 \end{array}$$

commutes. We now define a morphism $m'$ from $O_2$ to $T_1$ such that $m' \circ h = f$ and $m \circ m' = g$.

Assume $O_1$ has the form

$$i \xrightarrow{a_1}_{\epsilon_1} r_1 \xrightarrow{a_2}_{\epsilon_2} \cdots \xrightarrow{a_n}_{\epsilon_n} r_n \ ,$$

and $O_2$ has the form

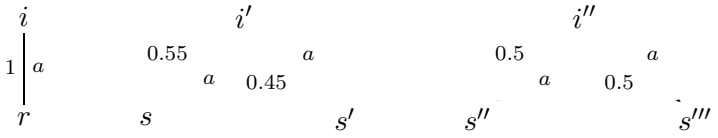
$$i' \xrightarrow{a_1}_{\epsilon'_1} r'_1 \xrightarrow{a_2}_{\epsilon'_2} \cdots \xrightarrow{a_n}_{\epsilon'_n} r'_n \xrightarrow{a_{n+1}}_{\epsilon'_{n+1}} \cdots \xrightarrow{a_{n+k}}_{\epsilon'_{n+k}} r'_{n+k} \ ,$$

where $\epsilon_j \leq \epsilon'_j$ for $1 \leq j \leq n$. Apart from $m'(i') = i_1$, we must define $m'(r'_j) = f(r_j)$ for $1 \leq j \leq n$. Since $g(r'_n) = m(f(r_n)) \xrightarrow{a_{n+1}} g(r'_{n+1})$ and $m$ is "zig-zag", there exists an $s$ such that $f(r_n) \xrightarrow{a_{n+1}} s$ and $m(f(s)) = g(r'_{n+1})$. Define $m'(r_{n+1}) = s$. Continuing this process for the remaining $r'_{n+2}, \ldots, r'_{n+k}$ we obtain the map $m' : O_2 \longrightarrow T_1$. It is now easy to show that $m'$ is indeed a morphism, since the transitions in the observations have infinitesimal probabilities, and that the "triangles" in the diagram commute. Hence, $m$ must be $\mathcal{P}$-open ■

Going through the proof the reader should be able to realize why only infinitesimal probabilities are allowed on the observations from $\mathcal{P}$. Allowing arbitrary probabilities would imply that two ptss which are related by an open map $m$ would be locally isomorphic in the following sense: if $m(r) \xrightarrow{a}_{\mu} s'$, then there exists an $r \xrightarrow{a}_{\mu} r'$ such that $m(r') = s'$.

From the definition of the morphisms in $\mathcal{PPTS}$ one observes the following facts:

- If $m : T_1 \longrightarrow T_2$ and $T_1$ is a pts, then $T_2$ must also be a pts.
- $\mathcal{PPTS}$ does not have pullbacks, neither does $\mathcal{PTS}$. Consider the following example which illustrates three ptss.




  Let $T$, $T_1$, and $T_2$ denote the ptss from left to right. Clearly there are uniquely determined morphisms from $T_1$ to $T$ and from $T_2$ to $T$. Together, they form a diagram which does not have a pullback.

However, for $\mathcal{P}$-bisimilarity to be an equivalence relation (a transitive relation, to be more precise) it is in general not necessary for the category $\mathcal{M}$ to have pullbacks. The following weaker result suffices.

**Theorem 109** *Given two $\mathcal{P}$-open morphisms between* pts*s, $m_1 : T_1 \longrightarrow T_0$ and $m_2 : T_2 \longrightarrow T_0$. There exists a* pts *$T$ and $\mathcal{P}$-open morphisms $\pi_1$ and $\pi_2$ such that*

$$\begin{array}{ccc} & T & \\ {\scriptstyle \pi_1}\swarrow & & \searrow{\scriptstyle \pi_2} \\ T_1 & & T_2 \\ {\scriptstyle m_1}\searrow & & \swarrow{\scriptstyle m_2} \\ & T_0 & \end{array}$$

*is a commuting square.*

**Proof.** We define a pts, $T = (Pr, i, Act, Can, \mu)$, and two maps $\pi_1 : T \longrightarrow T_1$, $\pi_2 : T \longrightarrow T_2$ with the desired properties.

- Define $S = \{m_1^{-1}(r) \times m_2^{-1}(r) \mid r \in Pr_0\}$.
- Let $Pr \subseteq S$ and $\rightsquigarrow \subseteq S \times Act \times [0;1] \times S$ be the least sets such that
  - $i = (i_1, i_2) \in Pr$
  - If $(r,s) \in Pr$, $r \xrightarrow{a}_{\mu_1} r'$, $\mu_1 > 0$, $s \xrightarrow{a}_{\mu_2} s'$, $\mu_2 > 0$, $m_1(r') = m_2(s')$, and $\mu' = \frac{\mu_1 \mu_2}{\mu}$, where $\mu > 0$ is uniquely determined by $m_1(r) = m_2(s) \xrightarrow{a}_{\mu} m_2(s') = m_1(r')$, then $((r,s), a, \mu', (r',s')) \in \rightsquigarrow$, written $(r,s) \overset{a}{\rightsquigarrow}_{\mu'} (r',s')$ and $(r',s') \in Pr$.
- Now for $(r,s) \in Pr$, $a \in Act$ define $(r,s) \in Can_a$ if $(r,s) \overset{a}{\rightsquigarrow}_{\mu} (r',s')$ for some $(r',s') \in Pr$. Also, define $\mu_{(r,s),a}((r',s')) = \mu' > 0$ if $(r,s) \overset{a}{\rightsquigarrow}_{\mu'} (r',s')$ and otherwise 0.
- Define $\pi_1 : T \longrightarrow T_1$ by $\pi_1((r,s)) = r$ and $\pi_2 : T \longrightarrow T_2$ by $\pi_2((r,s)) = s$.

It can be shown that $\pi_1$ and $\pi_2$ are morphisms. Here we merely show that $T$ is a pts.

Clearly, all states in $Pr$ are reachable from the initial state. Since $m_1$ and $m_2$ form a co-span of open maps between ptss we know, e.g., that if $m_1(r) \xrightarrow{a}_{\mu} v$ then

$$\mu = \sum_{r' m_1(r') = v} \mu_{r,a}(r') = \sum_{r \xrightarrow{a}_{\mu} r' m_1(r') = v} \mu \ .$$

Also, if $(r,s) \in Can_a$ then $(r,s) \overset{a}{\rightsquigarrow}_{\mu'} (r',s')$ for some $\mu' > 0$ and we have to show

$$\sum_{(r',s')} \mu_{(r,s),a}((r's')) = 1 \ .$$

It follows from the definition that $r \in Can_a$ and $s \in Can_a$. We then have

$$\sum_{(r',s')} \mu_{(r,s),a}((r's'))$$

$$
\begin{aligned}
&= \sum_{I_1}\sum_{I_2}\sum_{I_3} \frac{\mu_1\mu_2}{\mu} \\
&= \sum_{I_1}\sum_{I_2} \frac{\mu_1}{\mu} \sum_{I_3} \mu_2 \\
&= \sum_{I_1}\sum_{I_2} \frac{\mu_1}{\mu}\mu \\
&= \sum_{I_1}\sum_{I_2} \mu_1 \\
&= \sum_{I_1} \mu \\
&= 1 \ .
\end{aligned}
$$

where the index sets are

$$I_1 = \{v \,|\, m_1(r) \xrightarrow{a}_{\mu} v \ \wedge \ \mu > 0\} \ ,$$

$$I_2 = \{r' \,|\, r \xrightarrow{a}_{\mu_1} r' \ \wedge \ m_1(r') = v\} \ ,$$

and

$$I_3 = \{s' \,|\, s \xrightarrow{a}_{\mu_2} s' \ \wedge \ m_2(s') = v\} \ .$$

■

Noticing that the composition of two open maps is itself an open map [JNW93], we obtain the following corollary.

**Corollary 110** *$\mathcal{P}$-bisimilarity between* pts*s is an equivalence relation.*

If, as done by Larsen and Skou in [LS91], we had assumed a lower limit $\gamma$ on the probability of transitions (*minimal probability assumption*) and only considered the field of real numbers it would have been hard to obtain a result as the above, at least for us. The problem is that "pullback like" constructions of $T$ involves expressions of the type $\frac{\mu_1\mu_2}{\mu}$ which may denote values smaller than $\gamma$.

Next, we recall the definition of probabilistic bisimulation from [LS91]. We have adapted it to the case where the probabilistic transition systems have initial states.

**Definition 111** Let $T_j = (Pr_j, i_j, Act, Can_j, \mu_j)$, where $j = 1, 2$, be two distinct pts*s*. A *probabilistic bisimulation* between $T_1$ and $T_2$ is an equivalence $\equiv$ on $Pr = Pr_1 \cup Pr_2$ such that $i_1 \equiv i_2$ and whenever $r \equiv s$, then the following holds:

$$\forall a \in Act. \forall S \in Pr/_{\equiv}. \, r \xrightarrow{a}_{\mu} S \Leftrightarrow s \xrightarrow{a}_{\mu} S \ , \tag{5.51}$$

where the notation $r \xrightarrow{a}_{\mu} S$ was defined after Definition 105. □

Now to the main result of this section.

**Theorem 112** *Given two* pts*s,* $T_1$ *and* $T_2$*. Then :*

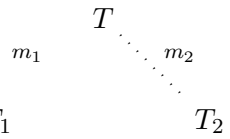
$T_1$ *is probabilistic bisimilar to* $T_2$ *if and only if* $T_1$ *is* $\mathcal{P}$*-bisimilar to* $T_2$*.*

**Proof.** Assume $\equiv$ is a probabilistic bisimulation between $T_1$ and $T_2$. We define a pts $T = (Pr, i, Act, Can, \mu)$ and two $\mathcal{P}$-open morphisms, $m_1 : T \longrightarrow T_1$ and $m_2 : T \longrightarrow T_2$, which constitute a span of open maps, showing that $T_1$ and $T_2$ are $\mathcal{P}$-bisimilar.

- Let $S = \{(r, s) \in Pr_1 \times Pr_2 \mid r \equiv s\}$.
- Let $Pr \subseteq S$ and $\leadsto \subseteq S \times Act \times [0; 1] \times S$ be the least sets such that
  - $i = (i_1, i_2) \in Pr$
  - If $(r, s) \in Pr$, $r \xrightarrow{a}_{\mu_1} r'$, $\mu_1 > 0$, $s \xrightarrow{a}_{\mu_2} s'$, $\mu_2 > 0$, $r' \equiv s'$, and $\mu' = \frac{\mu_1\mu_2}{\mu}$, where $\mu > 0$ is uniquely determined by $r \xrightarrow{a}_{\mu} [r']$ and $[r']$ is the equivalence class of $r'$ under $\equiv$, then $(r', s') \in Pr$ and $((r, s), a, \mu', (r', s')) \in \leadsto$, written $(r, s) \overset{a}{\leadsto}_{\mu'} (r', s')$.
- Now for $(r, s) \in Pr$, $a \in Act$ define $(r, s) \in Can_a$ if $(r, s) \overset{a}{\leadsto}_{\mu} (r', s')$ for some $(r', s') \in Pr$. Also, define $\mu_{(r,s),a}((r', s')) = \mu' > 0$ if $(r, s) \overset{a}{\leadsto}_{\mu'} (r', s')$ and otherwise 0.
- Define $m_1 : T \longrightarrow T_1$ by $m_1((r, s)) = r$ and $m_2 : T \longrightarrow T_2$ by $m_2((r, s)) = s$.

It can be shown that $T$ is a pts, that $m_1$ and $m_2$ are open maps, and that together they constitute the desired span of open maps.

Now assume there is a span of open maps:




Define the relation $\sim \subseteq Pr \times Pr$ as

$$r \sim r' \text{ if and only if } m_1(r) = m_1(r') \text{ or } m_2(r) = m_2(r') \ ,$$

and let $\approx = \sim^*$. Now define an equivalence relation $\equiv$ on $Pr_1 \cup Pr_2$ whose set of equivalence classes are

$$\{m_1(S') \cup m_2(S') \mid S' \in Pr/_{\approx}\} \ .$$

That $\equiv$ is indeed an equivalence relation on $Pr_1 \cup Pr_2$ follows from the definition of $\approx$ and that $m_1$ and $m_2$ are $\mathcal{P}$-open morphisms, which implies that they are surjective functions.

We now claim that $\equiv$ is a probabilistic bisimulation between $T_1$ and $T_2$. This follows from the following observations (without loss of generality stated for $T_1$).

- $i_1 \equiv i_2$.

- If $s \in Pr_1$, $a \in Act$, and $S$ is an equivalence class of $\equiv$ then $s \xrightarrow{a}_\mu S$ if and only if $s \xrightarrow{a}_\mu (S \cap Pr_1)$. Also, for any $r \in m_1^{-1}(s)$, $s \xrightarrow{a}_\mu S$ implies $r \xrightarrow{a}_\mu S'$ for the unique equivalence class $S' = m_1^{-1}(S \cap Pr_1) \in Pr/_\approx$, and $r \xrightarrow{a}_\mu S'$ for $S' \in Pr/_\approx$ implies $m_1(r) \xrightarrow{a}_\mu m_1(S')$.
- For $r_1 \approx r_2$ in T and any $S' \in Pr/_\approx$ we have $r_1 \xrightarrow{a}_\mu S'$ if and only if $r_2 \xrightarrow{a}_\mu S'$.

■

## 5.4 Non-interleaving Models

We start by recalling a well-known model of "true concurrency", event structures [Win80, Win86], and the characterisation of the non-interleaving bisimulation from [JNW93] and [NW95]. We then proceed to look for a non-interleaving generalisation of our characterisation of trace equivalence.

**Definition 113** A *(labelled prime) event structure* is a structure $(E, \leq, \#, l)$ consisting of a set of *events* $E$,which are partially ordered by $\leq$, the *causal dependency relation*, such that for $e' \in E$

$$|\{e' \,|\, e' \leq e\}| < \infty \ , \tag{5.52}$$

a binary, symmetric, irreflexive relation $\# \subseteq E \times E$, the *conflict relation*, such that for $e, e' \in E$

$$(e \# e' \ \wedge \ e' \leq e'') \ \Rightarrow \ e \# e'' \tag{5.53}$$

and a labelling function $l : E \longrightarrow Act$ assigning an action-label from $Act$ to each event in $E$.

Two events $e, e' \in E$ are said to be *concurrent*, denoted $e$ co $e'$, if and only if $\neg(e \leq e')$, $\neg(e' \leq e)$, and $\neg(e \# e')$. □

We will use the notation $e < e'$ whenever $e \leq e'$ and $e \neq e'$.

The finiteness assumption restricts attention to discrete processes where an event occurrence depends only on finitely many previous occurrences. The axiom on the conflict relation expresses that if two events causally depend on events in conflict then they too are in conflict.

Guided by our interpretation we can formulate a notion of computation state of an event structure $(E, \leq, \#, l)$. Taking a computation state of a process to be represented by the set $x$ of events which have occurred in the computation, we expect that

$$(e' \in x \ \wedge \ e \leq e') \ \Rightarrow \ e \in x \ ,$$

which expresses that if an event has occurred, then all events on which it causally depends have occurred too, and also that

$$\forall e, e' \in x . \neg(e \# e') \ ,$$

which expresses that no two events in conflict can occur together in the same computation.

**Definition 114** Let $(E, \leq, \#, l)$ be an event structure. Define its *configurations*, $\mathcal{D}(E, \leq, \#, l)$, to consist of those finite subsets $x \subseteq E$ such that

$$\forall e, e' \in x. \neg(e \# e') \ , \quad \text{conflict-free} \tag{5.54}$$

$$\forall e, e'. e' \leq e \in x \ \Rightarrow \ e' \in x \ , \quad \text{downwards-closed} \tag{5.55}$$

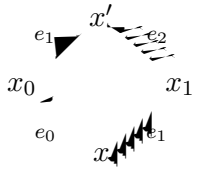
□

Notice that $e \downarrow = \{e' \in E \mid e' \leq e\}$ is a configuration for any $e \in E$.

Events manifest themselves as atomic jumps from one configuration to another, in a way which allows one to regard event structures as transition systems with a notion of concurrency. Notice that the event structures presented here are not just an arbitrary choice from the world of so-called non-interleaving models.It is well known, see e.g. [WN94], that these structures bear a strong relationship to many other models like net systems [Thi87] and asynchronous transition systems [Shi85, Bed88].

**Definition 115** Let $(E, \leq, \#, l)$ be an event structure. Let $x, x'$ be configurations. We write $x \xrightarrow{e} x'$ if and only if $e \notin x$ and $x' = x \cup \{e\}$. □

**Proposition 116** *Two events $e_0, e_1$ of an event structure are concurrent if and only if there exist configurations $x, x_0, x_1, x'$ such that*




Morphisms on event structures are defined as follows:

**Definition 117** Let $ES = (E, \leq, \#, l)$ and $ES' = (E', \leq', \#', l')$ be event structures over the same action set $Act$. A *morphism* from $ES$ to $ES'$ consists of a total function $\eta : E \longrightarrow E'$ on events, which satisfies $l' \circ \eta = l$ and

$$\begin{aligned} x \in \mathcal{D}(ES) \quad \Rightarrow \quad & \eta x \in \mathcal{D}(ES') \ \wedge \\ & \forall e_0, e_1 \in x. \eta(e_0) = \eta(e_1) \ \Rightarrow \ e_0 = e_1. \end{aligned} \tag{5.56}$$

□

A morphism $\eta : ES \longrightarrow ES'$ between event structures expresses how behaviour in $ES$ determines behaviour in $ES'$. The function $\eta$ expresses how the occurrence of events in $ES$ are simulated by events in $ES'$.

We write $\mathcal{E}$ for the category of event structures; composition is the usual componentwise composition of functions.

### 5.4.1 Strong History-preserving Bisimulation

Choosing $\mathcal{M}$ as $\mathcal{E}$ leaves open the choice of an observation category $\mathcal{P}$. However, Pratt's *pomsets* are a natural "concurrent" counterpart of the sequential observations from Sec. 5.2, as suggested in [JNW93]. Formally, we may identify pomsets with the full subcategory, $\mathcal{PO}$, of $\mathcal{E}$ whose objects consist of event structures, in which $\#$ is empty (no conflict), i.e., structures of the form $P = (E, \leq, \emptyset, l)$, representing individual non-sequential "runs" of systems. Notice that we may "dually" identify Milners's synchronisation trees with event structures with empty concurrency relation.

It turns out that with this choice of pomsets $\mathcal{PO}$-bisimilarity is a strengthening of the equivalence *history-preserving bisimulation*, previously studied by Rabinovitch and Trakhtenbrot [RT88], and van Glabeek and Goltz [GG89].

**Definition 118** A *history-preserving bisimulation* between two labelled event structures $ES_1$ and $ES_2$ over a common labelling set $Act$ consists of a set $H$ of triples $(x_1, f, x_2)$ where $x_1$ is a configuration of $ES_1$, $x_2$ a configuration of $ES_2$, and $f \subseteq E_1 \times E_2$ is an isomorphism (with domain $x_1$ and co-domain $x_2$) between them (regarded as pomsets), such that $(\emptyset, \emptyset, \emptyset) \in H$ and, whenever $(x_1, f, x_2) \in H$

(i) if $x_1 \xrightarrow{e} x_1'$ in $ES_1$ then there exists an $e'$ such that $x_2 \xrightarrow{e'} x_2'$ in $ES_2$ and $l_1(e) = l_2(e')$ and $(x_1', f', x_2') \in H$ with $f \subseteq f'$, for some $x_2'$ and $f'$.

(ii) if $x_2 \xrightarrow{e'} x_2'$ in $ES_2$ then there exists an $e$ such that $x_1 \xrightarrow{e} x_1'$ in $ES_1$ and $l_1(e) = l_2(e')$ and $(x_1', f', x_2') \in H$ with $f \subseteq f'$, for some $x_1'$ and $f'$

A history-preserving bisimulation $H$ is *strong* when it further satisfies

(I) $(x, f, y) \in H$ & $x' \subseteq x$, for a configuration $x'$ of $ES_1$ implies $(x', f', y') \in H$, for some $f' \subseteq f$ and $y' \subseteq y$.

(II) $(x, f, y) \in H$ & $y' \subseteq y$, for a configuration $y'$ of $ES_2$, implies $(x', f', y') \in H$, for some $f' \subseteq f$ and $x' \subseteq x$.

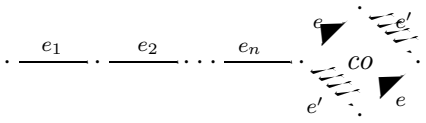
□

In [JNW94, NW95] the following theorem was shown.

**Theorem 119** *$\mathcal{PO}$-bisimilarity between labelled event structures coincides with strong history-preserving bisimilarity.*

One interesting observation is that the characterisation of strong history-preserving bisimilarity is very robust with respect to the choice of observations.

**Theorem 120** *The characterisation of strong history-preserving bisimulation as $\mathcal{P}$-bisimilarity also holds for a very restricted full subcategory of $\mathcal{PO}$ containing only objects corresponding to sequential words, "sticks", and "lollipops" like*

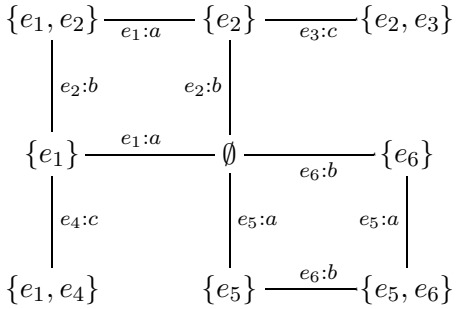



**Proof.** Follows from the proof of Theorem 119. ∎

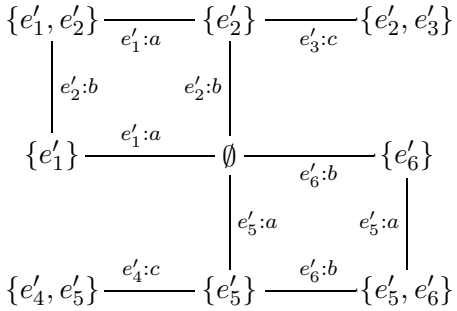
The above observation also suggests that history-preserving bisimulation might be hard, or even impossible, to characterise in a similar intuitive way. In fact, we have the following observation.

**Theorem 121** *Let $\mathcal{M}$ be any category whose objects are event structures and whose morphisms $\eta : ES \longrightarrow ES'$ simulates events from $ES$ by events in $ES'$ in the following sense: $\eta$ induces a simulation morphism in the sense of Definition 72 between the* lts*s induced by Definition 114 and 115. Let $\mathcal{P}$ be any subcategory of $\mathcal{M}$ containing the subcategory corresponding to $\mathcal{P}$ as defined in Definition 5.10, i.e., words closed under prefix. Then, $\mathcal{P}$-bisimilarity does not coincide with history-preserving bisimulation.*

**Proof.** Assume by contradiction $\mathcal{P}$-bisimilarity does coincide with history-preserving bisimulation. Consider the two event structures $ES_1$ and $ES_2$ whose induced lts*s* are




and




respectively. For clarity, the transitions are labelled by events and their labels. For the proof, we only require that the transitions are labelled by the labels of the events. Since $ES_1$ and $ES_2$ are history-preserving bisimilar, there must exist, by our assumptions, a span of open maps from a common event structure $ES$

Since $ES$ is $\mathcal{P}$-bisimilar to both $ES_1$ and $ES_2$, they must be history-preserving bisimilar, by our assumptions. Hence, $ES$'s induced $\mathsf{lts}$ must contain

$$\begin{array}{ccc} \{e,e'\} & \xrightarrow{\quad e{:}a \quad} & \{e'\} \\ \Big| {\scriptstyle e'{:}b} & & {\scriptstyle e'{:}b} \Big| \\ \{e\} & \xrightarrow{\quad e{:}a \quad} & \emptyset \end{array}$$

By our assumptions and Theorem 73, the induced morphisms between the $\mathsf{lts}$s must be $\mathcal{P}$-open in the sense of Lemma 74. However, it is easy to see that either the $\mathsf{lts}$ of $ES_1$ or $ES_2$ cannot be strongly bisimilar in the sense of Definition 71 to the $\mathsf{lts}$ of $ES$. Hence $ES$ cannot be history-preserving bisimilar with either $ES_1$ or $ES_2$, a contradiction. ■

This observation tells us, that if history-preserving bisimulation is to be captured as $\mathcal{P}$-bisimulation for some $\mathcal{P}$ and $\mathcal{M}$, the setting will be somewhat different and probably less intuitive than the setting which captures strong history-preserving bisimulation. We could also interpret this observation as saying that history-preserving bisimulation is perhaps a less natural choice for a true concurrency equivalence than strong history-preserving bisimulation. For instance, a history-preserving bisimulation between $ES_1$ and $ES_2$ will match

$$\emptyset \xrightarrow{e_1} \{e_1\}$$

by

$$\emptyset \xrightarrow{e'_5} \{e'_5\}$$

and subsequently

$$\{e_1\} \xrightarrow{e_2} \{e_1, e_2\}$$

by

$$\{e'_5\} \xrightarrow{e'_6} \{e'_5, e'_6\} \ .$$

However, the matching of $e_2$ by $e'_6$ is dependent of the previous matching of $e_1$ by $e'_5$, although these events are independent of $e_2$ and $e'_6$, respectively, since

$$\emptyset \xrightarrow{e_2} \{e_2\}$$

cannot be matched by

$$\emptyset \xrightarrow{e'_6} \{e'_6\} \ .$$

On the other hand, (I) and (II) in Definition 118 says that strong history-preserving bisimulation does have the property that if two events $e$ and $e'$ are matched, than this matching is consistent irrespect of previous matchings between events independent of $e$ and $e'$, respectively.

Nielsen and Winskel have also defined the notion of a *strong coherent history-preserving bisimulation* between two labelled event structures.

**Definition 122** A history-preserving bisimulation $H$ between event structures $ES_1$ and $ES_2$ over a common labelling set $Act$ is said to be *strongly coherent* if it satisfies the following property

(I) whenever $(x, f, y) = \bigcup_{i \in I}(x_i, f_i, y_i)$ for some index set $I$, then $(x, f, y) \in H$ if and only if $\forall i \in I.\, (x_i, f_i, y_i) \in H$.

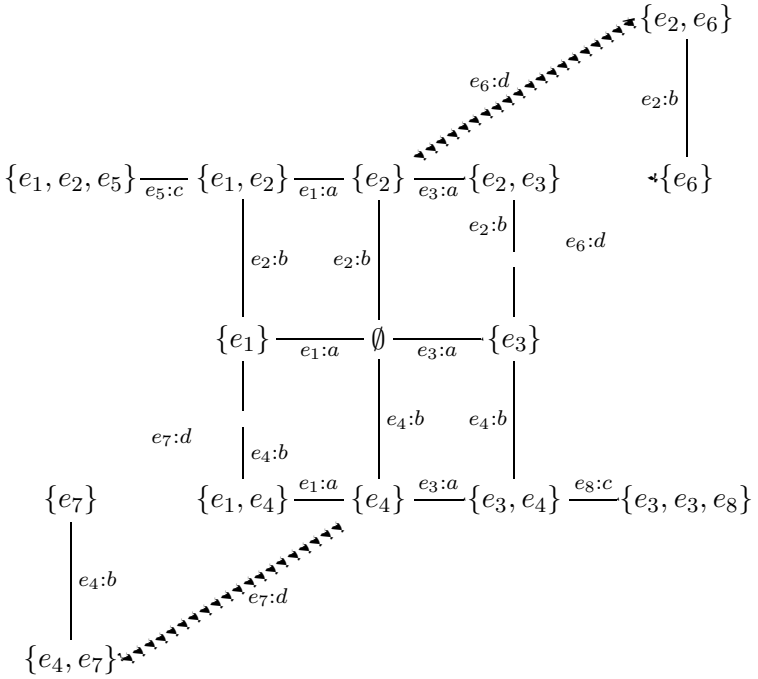
□

Recall that the notation $(x', f', y')$ implicitly means that $x'$ is a configuration of $ES_1$, $y'$ is a configuration of $ES_2$, and $f'$ is an isomorphism between them (regarded as pomsets).

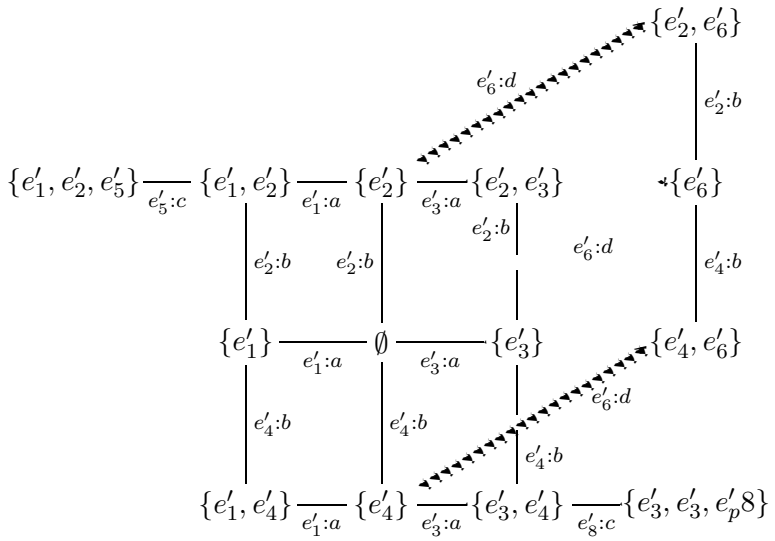
It is easy to see that strong coherent history-preserving bisimilarity implies strong history-preserving bisimilarity. However, the opposite is not the case:

**Theorem 123** *Strong coherent history-preserving bisimilarity is strictly finer than strong history-preserving bisimilarity.*

**Proof.** Consider the following two event structures, $ES_1$,




and $ES_2$




We claim that $ES_1$ and $ES_2$ are strong history-preserving bisimilar but not strong coherent history-preserving bisimilar. The former is easily established following Definition 118. To see the latter, observe that a strong coherent history-preserving bisimulation $H$ between $ES_1$ and $ES_2$ must contain the triples

$$
\begin{gathered}
(\{e_6\}, \{(e_6, e'_6)\}, \{e'_6\}) \\
(\{e_2, e_6\}, \{(e_2, e'_2), (e_6, e'_6)\}, \{e'_2, e'_6\}) \\
(\{e_2, e_6\}, \{(e_2, e'_4), (e_6, e'_6)\}, \{e'_2, e'_4\})
\end{gathered}
$$

and hence, by strongness, also

$$
\begin{gathered}
(\{e_2\}, \{(e_2, e'_2)\}, \{e'_2\}) \\
(\{e_2\}, \{(e_2, e'_4)\}, \{e'_4\}) \quad .
\end{gathered}
$$

Since $H$ must contain either

$$
\begin{gathered}
(\{e_1\}, \{(e_1, e'_1)\}, \{e'_1\}) \quad \text{or} \\
(\{e_1\}, \{(e_1, e'_3)\}, \{e'_3\})
\end{gathered}
$$

and is strongly coherent, we conclude that $H$ contains either

$$
\begin{gathered}
(\{e_1, e_2\}, \{(e_1, e'_1), (e_2, e'_4)\}, \{e'_1, e'_4\}) \quad \text{or} \\
(\{e_1, e_2\}, \{(e_1, e'_3), (e_2, e'_2)\}, \{e'_2, e'_3\}) \ .
\end{gathered}
$$

However, $\{e_1, e_2\} \xrightarrow{e_5:c} \{e_1, e_2, e_5\}$ while neither $\{e'_1, e'_4\}$ nor $\{e'_2, e'_3\}$ has an enabled $c$-labelled event. ■

### 5.4.2 Pomset Equivalences

We continue by considering the concurrent counterpart of trace equivalence. We recast, in the terminology of Pratt, the choice of observation extension defined formally in terms of event structure morphisms on pomsets in Sec. 5.4.1. The following terminology is from [Pra86].

**Definition 124** Given a pomset $P = (E, \leq, \emptyset, l)$, any $(E, \leq', \emptyset, l)$ for which $\leq \subseteq \leq'$ is called an *augment* of $P$. Any restriction of $(E, \leq, \emptyset, l)$ to a downwards closed subset of $E$ is called a *prefix* of $P$. □

On pomsets, a morphism $\eta$ from $P$ to $P'$ amounts to "$P$ being an augment of a prefix of $P'$"—a very generous notion of extension. An alternative would be the restricted notion of extension corresponding to the one from the characterisation of trace equivalence for transition systems.

**Definition 125** Let $\mathcal{PO}_1$ denote the subcategory of $\mathcal{PO}$ whose morphisms are the identities and morphisms with the empty pomset as domain. □

With this choice, we would expect to get some kind of equivalence in terms of "pomset-languages" of event structures.

**Definition 126** Let $ES = (E, \leq, \#, l)$ be an event structure. The *pomset language* of $ES$, $\mathcal{P}(ES)$, is defined as the set of restrictions of $ES$ to $\mathcal{D}(E)$. Following Pratt, $\alpha\mathcal{P}(E)$ denotes the augmentation closure of $\mathcal{P}(E)$. □

And the $\mathcal{PO}_1$-bisimulation may now be characterised as follows.

**Theorem 127** *Two event structures $ES$ and $ES'$ are $\mathcal{PO}_1$-bisimilar if and only if $\alpha\mathcal{P}(ES) = \alpha\mathcal{P}(ES')$.*

**Proof sketch.** The proof is a direct analog of the proof of Theorem 83 based on a lemma similar to Lemma 82. ■

### 5.4.3 Remarks on Decidability Issues

The theory of open maps has helped identifying new behavioural equivalences such as strong (coherent) history-preserving bisimilarity. Associated to each behavioural equivalence one may consider the decision problem: given two (finite descriptions of) systems $T_1$ and $T_2$, are the $\mathcal{P}$-bisimilar? In particular, Nielsen and Winskel [NW95] observe that using the categorical relationships (adjunctions) between the category of (labelled) 1-safe nets and event structures one may derive a notion of behavioural equivalence between two finite 1-safe nets as follows: $N_1$ and $N_2$ are said to be strong (coherent) history-preserving bisimilar if and only if the event structures associated with their unfoldings

are strong (coherent) history-preserving bisimilar. To the best of the authors knowledge, these problems are still open. [1]

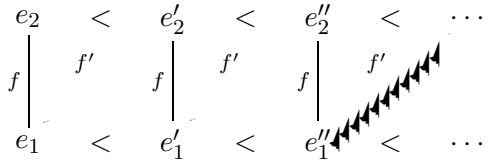
We continue by giving an alternative definition of strong coherent history-preserving bisimulation which might shed some light on the "strongness condition".

In the following we shall restrict attention to event structures $E$ that do not exhibit "auto-concurrency", i.e., $E$ has no configuration $x$ and concurrent events $e_1$ and $e_2$ such that $x \xrightarrow{e_1}$, $x \xrightarrow{e_2}$, and $l(e_1) = l(e_2)$.

**Lemma 128** *Given two event structures $ES_1$ and $ES_2$ that do not exhibit auto-concurrency. Let $x_1$ be a configuration of $E_1$, $x_2$ a configuration of $E_2$, and $f$ and $f'$ isomorphisms between $x_1$ and $x_2$. Then $f = f'$.*

**Proof.** Assume $f \neq f'$. Choose a least—with respect to subset inclusion—configuration $x_1' \subseteq x_1$ such that $f|_{x_1'} \neq f'|_{x_1'}$. The notation $g|_c$ denotes the restriction of $g$ to $c$. Then, $x_1' \neq \emptyset$. Let $e_1 \in x_1'$ be a maximal event, i.e., there exists no $e' \in x_1'$ such that $e < e'$. By minimality of $x_1'$ we conclude $f(e_1) \neq f'(e_1)$. Let $e_2 = f(e_1)$ and $e_2' = f'(e_1)$. Since $f$ is an isomorphism between $x_1$ and $x_2$, $f^{-1}(e_2') \neq e_1$. Let $e_1' = f^{-1}(e_2')$. By our assumptions, $e_1$ and $e_1'$ cannot be concurrent since they must be identically labelled. Also, since $e_1$ and $e_1'$ belong to $x_1$ they cannot be in conflict. Hence there are two possibilities:

- If $e_1 < e_1'$, then $e_2 = f(e_1) < f(e_1') = e_2' = f'(e_1) < f'(e_1')$. Let $e_2'' = f'(e_1')$ and $e_1'' = f^{-1}(e_2'')$. Then, $e_1 < e_1' < e_1''$ are events in $x_1$. By repeating this argument we obtain an infinite increasing chain $e_1 < e_1' < e_1'' < \cdots$ of events in $x_1$, as illustrated below.




  However, this contradicts the finiteness of $x_1$. So we conclude $\neg(e_1 \leq e_1')$.

- If $e_1 > e_1'$, we obtain—by a similar argument—an infinite decreasing chain $e_1 > e_1' > e_1'' > \cdots$ of events in $x_1$. We conclude $\neg(e_1 \geq e_1')$.

Hence $e_1$ and $e_1'$ must be concurrent (and identically labelled), which contradicts our assumptions. We finally conclude $f = f'$. ∎

**Definition 129** Given two event structures $ES_1$ and $ES_2$. Let $R \subseteq E_1 \times E_2$. $\sim_R$ is the least set of triples $(x_1, f, x_2)$, as defined in Definition 118, such that

$$(\emptyset, \emptyset, \emptyset) \in \sim_R \tag{5.57}$$

$$(\,(x_1, f, x_2) \in \sim_R \;\wedge\; (e_1, e_2) \in R \;\wedge\; x_1 \xrightarrow{e_1} x_1' \;\wedge\; x_2 \xrightarrow{e_2} x_2' \wedge \tag{5.58}$$

[1] Notice that the complexity of history-preserving bisimulation for 1-safe nets was determined to be DEXPTIME-complete by Jategaonkar [Jat93].

$$f \cup \{(e_1, e_2)\} \text{ is an isomorphism between } x_1' \text{ and } x_2' \,) \tag{5.59}$$

$$\Rightarrow \quad (x_1', f \cup \{(e_1, e_2)\}, x_2') \in \sim_R \tag{5.60}$$

The relation $\sim_R$ is called a bisimulation if whenever

$$(x_1, f, x_2) \in \sim_R \ \wedge\, x_1 \xrightarrow{e_1} x_1' \tag{5.61}$$

then there exists a $e_2$ such that

$$(e_1, e_2) \in R \ \wedge\ (x_1', f \cup \{(e_1, e_2)\}, x_2') \in \sim_R \tag{5.62}$$

and vice versa. □

Assume $H$ is a strong coherent history-preserving bisimulation between $ES_1$ and $ES_2$. Let $R_H$ be the set

$$\{(e_1, e_2) \mid \exists (x_1, f, x_2) \in H.\, e_1 \in x_1 \ \wedge\ e_2 \in x_2 \ \wedge\ f(e_1) = e_2\}\ .$$

**Lemma 130** *If $(x_1, f, x_2) \in \sim_{R_H}$, then $(x_1, f, x_2) \in H$.*

**Proof.** The proof is by induction in $|x_1|$. The base case $|x_1| = 0$ is trivial. So assume $|x_1| = n + 1$. Then by definition there exist $(x_1', f', x_2') \in \sim_{R_H}$ and $(e_1, e_2) \in R_H$ such that $x_1' \xrightarrow{e_1} x_1$, $x_2' \xrightarrow{e_2} x_2$, and $f = f' \cup \{(e_1, e_2)\}$. Since $(e_1, e_2) \in R_H$ there exists $(x_1'', g, x_2'') \in H$ such that $e_1 \in x_1''$, $e_2 \in x_2''$, and $g(e_1) = e_2$. By strongness of $H$, $(e_1 \downarrow, g|_{e_1\downarrow}, e_2 \downarrow) \in H$. Let $c_1 = e_1 \downarrow \setminus \{e_1\}$ and $c_2 = e_2 \downarrow \setminus \{e_2\}$. Notice $c_1 \subseteq x_1'$ and $c_2 \subseteq x_2'$. By induction $(x_1', f', x_2') \in H$ and by strongness of $H$ $(c_1, f'|_{c_1}, f'(c_1)) \in H$. Since $f$ is an isomorphism between $x_1$ and $x_2$, we conclude $f'(c_1) = c_2$. Similarly, from $(x_1'', g, x_2'') \in H$ we conclude $(c_1, g|_{c_1}, c_2) \in H$. By Lemma 128, we have $f'|_{c_1} = g|_{c_1}$. Since $(x_1', f', x_2') \in H$, $(e_1 \downarrow, g|_{e_1\downarrow}, e_2 \downarrow) \in H$, and $(x_1' \cup e_1 \downarrow, f' \cup g|_{e_1\downarrow}, x_2' \cup e_2 \downarrow) = (x_1, f, x_2)$, we conclude by coherence of $H$ that $(x_1, f, x_2) \in H$. ■

**Theorem 131** *Given $H$, a strong coherent history-preserving bisimulation between $ES_1$ and $ES_2$. Then $\sim_{R_H}$ is a bisimulation.*

**Proof.** Given $(x_1, f, x_2) \in \sim_{R_H}$ and assume $x_1 \xrightarrow{e_1} x_1'$. By Lemma 130 $(x_1, f, x_2) \in H$. Since $H$ is a strong coherent history-preserving bisimulation there exists a $e_2$ such that $x_2 \xrightarrow{e_2} x_2'$ and $(x_1', f \cup \{(e_1, e_2)\}, x_2') \in H$. But then $(e_1, e_2) \in R_H$, and hence $(x_1', f \cup \{(e_1, e_2)\}, x_2') \in \sim_{R_H}$. The case $x_2 \xrightarrow{e_2} x_2'$ is handled similarly. ■

**Theorem 132** *Given event structures $ES_1$ and $ES_2$. If $R \subseteq E_1 \times E_2$ is such that $\sim_R$ is a bisimulation, then $\sim_R$ is a strong coherent history-preserving bisimulation between $ES_1$ and $ES_2$.*

**Proof.** We prove that all relevant conditions stated in Definition 118 and 122 hold.

- Clearly, $(\emptyset, \emptyset, \emptyset) \in \sim_R$.

- Assume $(x_1, f, x_2) \in \sim_R$ and $x_1 \xrightarrow{e_1} x_1'$. Since $\sim_R$ is a bisimulation there exists a $e_2$ such that $x_2 \xrightarrow{e_2} x_2'$ and $(x_1', f \cup \{(e_1, e_2)\}, x_2') \in \sim_R$. The dual case $x_2 \xrightarrow{e_2} x_2'$ is handled similarly.

- Assume $(x_1, f, x_2)$, $(x_1', f', x_2')$, and $(x_1 \cup x_1', f \cup f', x_2 \cup x_2')$ all denote triples of configurations and isomorphisms as defined in Definition 118. Since configurations are finite, one can show that establishing the property in Definition 122 is equivalent to showing that $(x_1, f, x_2), (x_1', f', x_2') \in \sim_R$ if and only if $(x_1 \cup x_1', f \cup f', x_2 \cup x_2') \in \sim_H$.

  Assume $(x_1, f, x_2), (x_1', f', x_2') \in \sim_R$. Since $(x_1 \cup x_1', f \cup f', x_2 \cup x_2')$ is a triple of configurations related by the isomorphism $f \cup f'$, we conclude that events in $x_1 \backslash x_1'$ are concurrent with events in $x_1' \backslash x_1$. Since $f \cup f'$ is an isomorphism, the same holds for $x_2 \backslash x_2'$ and $x_2' \backslash x_2$. Let $e_{j_1}, e_{j_2}, \ldots, e_{j_m}$ be an enumeration of the events in $x_1 \backslash x_1'$, such that

$$x_1 \cap x_1' \xrightarrow{e_{j_1}} \cdots \xrightarrow{e_{j_m}} x_1 \ .$$

  In fact, we also have

$$\begin{aligned} x_2 \cap x_2' &\xrightarrow{f(e_{j_1})} \cdots \xrightarrow{f(e_{j_m})} x_2 \ , \\ x_1' &\xrightarrow{e_{j_1}} \cdots \xrightarrow{e_{j_m}} x_1 \cup x_1' \ , \quad \text{and} \\ x_2' &\xrightarrow{f(e_{j_1})} \cdots \xrightarrow{f(e_{j_m})} x_2 \cup x_2' \ . \end{aligned}$$

  By definition of $\sim_R$ we know $(e_{j_i}, f(e_{j_i})) \in R$, for $1 \leq i \leq m$. So we conclude

$$\begin{gathered} (x_1' \cup \{e_{j_1}\}, f' \cup \{(e_{j_1}, f(e_{j_1}))\}, x_2' \cup \{f(e_{j_1})\} \in \sim_R \\ \cdots \\ (x_1' \cup \{e_{j_1}, \ldots, e_{j_m}\}, f' \cup \{(e_{j_1}, f(e_{j_1})), \ldots, (e_{j_m}, f(e_{j_m}))\}, \\ x_2' \cup \{f(e_{j_1}), \ldots, f(e_{j_m})\} \in \sim_R \ . \end{gathered}$$

  Since the last triple is $(x_1 \cup x_1', f \cup f', x_2 \cup x_2')$ we have shown the "only if" direction.

  Conversely, assume $(x_1 \cup x_1', f \cup f', x_2 \cup x_2') \in \sim_H$. Then we know there exists an enumeration $e_1, e_2, \ldots, e_m$ of all events in $x_1 \cup x_1'$ such that

$$\begin{gathered} \emptyset \xrightarrow{e_1} \cdots \xrightarrow{e_m} x_1 \cup x_1' \ , \\ \emptyset \xrightarrow{g(e_1)} \cdots \xrightarrow{g(e_m)} x_2 \cup x_2' \ , \text{ and} \\ (\{e_1, \ldots, e_i\}, \{(e_1, g(e_1)), \ldots, (e_i, g(e_i))\}, \{g(e_1), \ldots, g(e_i)\}) \ , \end{gathered}$$

  where $g = f \cup f'$ and $1 \leq i \leq m$. But since $(e_i, g(e_i)) \in R$, for $1 \leq i \leq m$, and all events in $x_1 \backslash x_1'$ are independent of events $x_1' \backslash x_1$, we know

$$\emptyset \xrightarrow{e_{j_1}} \cdots \xrightarrow{e_{j_n}} x_1 \ ,$$

where $1 \leq j_1 < \cdots < j_n \leq m$ is the subsequence of indexes corresponding to the events from $x_1$. But then

$$\begin{gathered}
(\{e_{j_1}\}, \{(e_{j_1}, g(e_{j_1})\}, \{g(e_{j_1})\}) \in \sim_R \text{ , hence} \\
(\{e_{j_1}, e_{j_2}\}, \{(e_{j_1}, g(e_{j_1}), (e_{j_2}, g(e_{j_2}))\}, \{g(e_{j_1}), g(e_{j_2})\}) \in \sim_R \text{ , hence} \\
\cdots \\
(\{e_{j_1}, \ldots, e_{j_n}\}, \{(e_{j_1}, g(e_{j_1}), \ldots, (e_{j_n}, g(e_{j_n})\}, \\
\{g(e_{j_1}), \ldots, g(e_{j_n})\}) \in \sim_R \ .
\end{gathered}$$

Since the last triple is just $(x_1, f, x_2)$ and the case $(x_1', f', x_2') \in \sim_R$ can be shown analogously, we have shown the "if" direction.

We conclude $\sim_R$ is a strong coherent history-preserving bisimulation. ■

## 5.5 Summary

We have examined Joyal, Nielsen, and Winskel's notion of $\mathcal{P}$-bisimilarity as an abstract category theoretic definition of bisimulation.

Guided by our intuitive understanding of what it means for a system $X$ to be simulated by a system $Y$ we defined different categories of models of computation. Our choices of (sub)categories of observations were also guided by which behaviours ought to be observable.

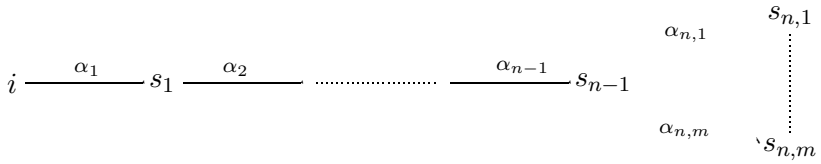
It turns out that we could identify well-know notions of behavioural equivalences. We started by the most fundamental (or coarsest) namely, trace equivalence. Then, we considered "invisible" actions and identified weak bisimulation, testing equivalence, and barbed bisimulation. We showed how the theory of open maps could be relaxed and we identified Larsen and Skou's probabilistic bisimulation. For technical reasons we applied the theory to a category in which the subcategory of observations was disjoint from the subcategory of models we were interested in. Finally, we examined non-interleaving models and behavioural equivalences.

Our results show that the theory of open maps does give meaningful equivalences when applied to well know models of computation.

In fact, some of the behavioural equivalences we captured are not "bisimulation like" in nature. So it seems that (spans of) open maps are more than a abstract way of capturing bisimulations. If one would be interested in only capturing bisimulation-like behavioural equivalences, one could try to define an abstract notion of "atomic observations" together with some abstract notion of prefixing as a basis for (i.e., inducing) the category of observations $\mathcal{P}$.

We also noticed that the equivalences we identified could be captured by several different choices of observations. E.g., from the proofs in Sect. 5.3.3 it is clear that we could choose a smaller subcategory of observations; binary branching along the "trunks" of the trees is sufficient. Theorem 120 also showed that strong history-preserving bisimulation could be characterised using a simpler category of observations.

On the other hand, consider the characterisation of strong bisimulation from [JNW93] (see also Sect. 5.2). Had $\mathcal{P}$ been chosen to be the subcategory induced by objects of the form




where all states are distinct, then an $\mathcal{P}$-open map $m$ would have been characterised by the usual "zig-zag" property

$$\text{if } m(r) \xrightarrow{\alpha} s', \text{ then there exists an } r' \text{ such that } r \xrightarrow{\alpha} r' \text{ and } m(r') = s',$$

and the additional "local bijection" property

$$\text{for any } \alpha \in \mathit{Act},\ \{r' \mid r \xrightarrow{\alpha} r'\} \text{ is in bijective correspondence with}$$

$$\{s' \mid m(r) \xrightarrow{\alpha} s'\} \text{ under } m.$$

So apart from helping us identify "characterising observations" for behavioural equivalences, the theory of open maps also allows us to test how "robust" an equivalence is against different choices of observations.

The choice of simulating morphisms turned out to be important. In the category of labelled transition systems $\mathcal{LTS}$ we didn't expect to be able to capture weak bisimulation just by changing the choice of observations. We defined new morphisms which intuitively corresponded to a "weak simulation".

The relationship exhibited in Theorem 131 and 132 shows that having a strong coherent history-preserving bisimulation corresponds to having a matching of events from both systems in such a way that this matching is consistent as discussed after Theorem 121. From this observation, we have attempted to decide strong coherent history-preserving bisimulation between labelled 1-safe nets using an approach based on Jategaonkar's "growth-site correspondences (gsc) [Jat93] and a matching table. More specifically, our idea has been to show that nets $N_1$ and $N_2$ are strong coherent history-preserving bisimilar if and only if there exists a certain type of finite automaton. Among other things, we believe the automaton should be a generalisation of that used in [Jat93, Theorem 6.3.4]. The states $(M, M', \text{gsc}, \text{match})$ contain a marking $M$ of $N_1$, a marking $M'$ of $N_2$, a growth-site correspondence gsc recording the how the latest occurrences of transitions in $N_1$ have been matched by latest occurrences of transitions in $N_2$, and a matching match corresponding to how future occurrences of transitions from $N_1$ and $N_2$ can or must

be matched. This last component is inspired by Theorem 131 and 132. However, although we believe this observation is an promising way to solve this decidability problem, we haven't been able to successfully apply this idea. Game theoretic characterisations of bisimulations exist, both for interleaving models [Sti93] and non-interleaving models [NC94]. It then seems that the decidability problem could be formulated as having a finite game strategy, corresponding to the "regular" finite automata-like structure we described above.

Another conjecture we have is that for the class of labelled 1-safe free choice nets strong (coherent) history-preserving bisimulation coincides with history-preserving bisimulation. In fact, all the 1-safe net examples we have been able to find which show that strong (coherent) history-preserving bisimulation is strictly finer than history-preserving bisimulation have not been free-choice.

# Chapter 6

# Open Maps and Congruences

**Contents**

## 6.1 Introduction

Based on the view that endofunctors on $\mathcal{M}$ may be seen as abstract operators we define a simple, natural, and general notion of a functor being $\mathcal{P}$-*factorisable*. We then show that a $\mathcal{P}$-factorisable functor must preserve $\mathcal{P}$-bisimilarity.

The presentation of $\mathcal{P}$-factorisability focusses, especially, on certain closure properties of the category $\mathcal{P}$. Based on this observation, we show how one can parametrise the proofs of functors being $\mathcal{P}$-factorisable with respect to the choice of the observation category $\mathcal{P}$, i.e., the choice of a behavioural equivalence. Intuitively, we fix the operators, but allow the behavioural equivalence to vary. Then we identify conditions on $\mathcal{P}$ which ensure that the varying equivalences are congruences with respect to the operators. Our results can be seen as "orthogonal" to that of Bloom, Istrail, and Meyer, in that we can parametrise with respect to the behavioural equivalences, as opposed to operators, [BIM88].

In the next section we generalise slightly Joyal, Nielsen, and Winskel's theory of open maps. In Sec. 6.3 we present our notion of $\mathcal{P}$-factorisability. Then, in Sec. 6.4 we apply our theory to a variant of Winskel and Nielsen's labelled transition systems [WN95]. We consider the universal constructions from [WN95] and provide general "congruence" results parametrised by the category of observations $\mathcal{P}$. We then continue to examine the trickier recursion operator in Sec. 6.5. Finally we summarise and give suggestions for further research in Sec. 6.6.

## 6.2 Open Maps, Generalised

In this section we briefly recall the basic definitions from [JNW93]. We present a slightly more general definition since it turns out more beneficial, more specifically for Theorem 162 and the discussion in Sect. 6.4.8.

Let $\mathcal{U}$ denote a category, the *universe*. A morphism $m : X \longrightarrow Y$ in $\mathcal{U}$ should intuitively be thought of as a simulation of $X$ in $Y$. Then, a subcategory of $\mathcal{U}$ which represents a *model of computation* has to be identified. We denote this category $\mathcal{M}$. Also, within $\mathcal{U}$, we choose a subcategory of "observation objects" and "observation extension" morphisms between these objects. We denote this *category of observations* by $\mathcal{P}$. If nothing else is mentioned, we assume that $\mathcal{U} = \mathcal{M}$, corresponding to the definitions in [JNW93].

Given an observation (object) $O$ in $\mathcal{P}$ and a model $X$ in $\mathcal{M}$, then $O$ is said to be an *observable behaviour* of $X$ if there exists a morphism $p : O \longrightarrow X$ in $\mathcal{M}$. We think of $p$ as representing a "run" of $O$ in $X$. We shall use $O, O', \ldots$ to denote observations and $T, T', X, Y, \ldots$ to denote objects from $\mathcal{M}$. A morphism $O \stackrel{q}{\longrightarrow} O'$ is implicitly assumed to belong to $\mathcal{P}$.

Next, we identify morphisms $m : X \longrightarrow Y$ in $\mathcal{M}$ which have the property that whenever an observable behaviour of $X$ can be extended via $f$ in $Y$ then that extension can be matched by an extension of the observable behaviour in $X$.

**Definition 133 Open Maps**
A morphism $m : X \longrightarrow Y$ in $\mathcal{M}$ is said to be $\mathcal{P}$-*open* (or just an *open map*) if whenever $f : O_1 \longrightarrow O_2$ in $\mathcal{P}$, $p : O_1 \longrightarrow X$, $q : O_2 \longrightarrow Y$ in $\mathcal{M}$, and the diagram

$$\begin{array}{ccc} O_1 & \xrightarrow{p} & X \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle m} \\ O_2 & \xrightarrow[q]{} & Y \end{array} \tag{6.1}$$

commutes, i.e., $m \circ p = q \circ f$, there exists a morphism $h : O_2 \longrightarrow X$ in $\mathcal{M}$ (a *mediating* morphism) such that the two triangles in the diagram

$$\begin{array}{ccc} O_1 & \xrightarrow{p} & X \\ {\scriptstyle f}\downarrow & \overset{h}{\nearrow} & \downarrow{\scriptstyle m} \\ O_2 & \xrightarrow[q]{} & Y \end{array} \tag{6.2}$$

commute, i.e., $p = h \circ f$ and $q = m \circ h$. When no confusion is possible, we refer to $\mathcal{P}$-open morphisms as *open maps*. □

The abstract definition of bisimilarity is as follows.

**Definition 134 $\mathcal{P}$-bisimilarity**
Two models $X$ and $Y$ in $\mathcal{M}$ are said to be $\mathcal{P}$-*bisimilar* (in $\mathcal{M}$), written $X \sim_{\mathcal{P}} Y$, if there exists a *span of open maps* from a common object $Z$:

$$\begin{array}{ccccc} & & Z & & \\ & {\scriptstyle m}\swarrow & & \searrow{\scriptstyle m'} & \\ X & & & & Y \end{array} \tag{6.3}$$

□

*Remark.* Notice that if $\mathcal{M}$ has pullbacks, it can be shown that $\sim_{\mathcal{P}}$ is an equivalence relation. The important observation is that pullbacks of open maps are themselves open maps. For more details, the reader is referred to [JNW93].

As a preliminary example of a category of models of computation $\mathcal{M}$ we present *labelled transition systems*.

**Definition 135** A *labelled transition system* over *Act* is a tuple

$$(S, i, Act, \longrightarrow) \ , \tag{6.4}$$

where $S$ is a set of states with *initial state* $i$, *Act* is a set of actions ranged over by $\alpha, \beta, \ldots$, and $\longrightarrow \subseteq S \times Act \times S$ is the transition relation. For the sake of readability we introduce the following notation. Whenever $(s_0, \alpha_1, s_1)$, $(s_1, \alpha_2, s_2)$, ..., $(s_{n-1}, \alpha_n, s_n) \in \longrightarrow$ we denote this as $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n$ or $s_0 \xrightarrow{v} s_n$, where $v = \alpha_1\alpha_2\cdots\alpha_n \in Act^*$. Also, we assume that all states $s \in S$ are reachable from $i$, i.e., there exists a $v \in Act^*$ such that $i \xrightarrow{v} s$. □

Let us briefly remind the reader of Park and Milner's definition of strong bisimulation [Mil89].

**Definition 136** Let $T_1 = (S_1, i_1, Act, \longrightarrow_1)$ and $T_2 = (S_2, i_2, Act, \longrightarrow_2)$. A *strong bisimulation between* $T_1$ *and* $T_2$ is a relation $R \subseteq S_1 \times S_2$ such that

$$(i_1, i_2) \in R \ , \tag{6.5}$$

$$((r,s) \in R \ \wedge \ r \xrightarrow{\alpha}_1 r') \ \Rightarrow \ \text{for some } s', (s \xrightarrow{\alpha}_2 s' \ \wedge \ (r', s') \in R) \ , \tag{6.6}$$

$$((r,s) \in R \ \wedge \ s \xrightarrow{\alpha}_2 s') \ \Rightarrow \ \text{for some } r', (r \xrightarrow{\alpha}_1 r' \ \wedge \ (r', s') \in R) \ . \tag{6.7}$$

$T_1$ and $T_2$ are said to be *strongly bisimilar* if there exists a strong bisimulation between them. □

Henceforth, whenever no confusion is possible we drop the indexing subscripts on the transition relations and write $\longrightarrow$, instead.

By defining morphisms between labelled transition systems we can obtain a category of models of computation, $\mathcal{TS}_{Act}$, labelled transition systems.

**Definition 137** Let $T_1 = (S_1, i_1, Act, \longrightarrow_1)$ and $T_2 = (S_2, i_2, Act, \longrightarrow_2)$. A morphism $m : T_1 \longrightarrow T_2$ is a function $m : S_1 \longrightarrow S_2$ such that

$$m(i_1) = i_2 \ , \tag{6.8}$$

$$s \xrightarrow{\alpha}_1 s' \ \Rightarrow \ m(s) \xrightarrow{\alpha}_2 m(s') \ . \tag{6.9}$$

□

The intuition behind this specific choice of morphism is that an $\alpha$-labelled transition in $T_1$ must be simulated by an $\alpha$-labelled transition in $T_2$. Composition of morphisms is defined as the usual composition of functions.

By varying the choice of $\mathcal{P}$ we can obtain different behavioural equivalences, corresponding to $\mathcal{P}$-bisimilarity. E.g., if, as done in [JNW93], we choose $\mathcal{P}_M$ as the full subcategory of $\mathcal{TS}_{Act}$ whose objects are finite synchronisation trees with at most one maximal branch, i.e., labelled transition systems of the form

$$i \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n \ , \tag{6.10}$$

where all states are distinct, we get:

**Theorem 138** [JNW93] *$\mathcal{P}_M$-bisimilarity coincides with Park and Milner's strong bisimulation.*

By slightly restricting our choice of observation extension so that $\mathcal{P}_H$ is the subcategory of $\mathcal{TS}_{Act}$ whose objects (observations) are of the form (6.10), and whose morphisms are the identity morphisms and morphisms whose domains are observations having only one state (the empty word), we get:

**Theorem 139** [NC95] *$\mathcal{P}_H$-bisimilarity coincides with Hoare trace equivalence.*

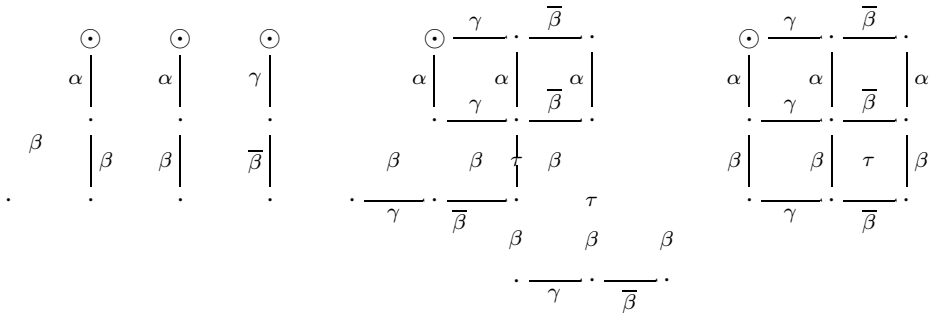
In Chap. 5 other behavioural equivalences were considered, e.g., weak bisimulation and probabilistic bisimulation.

## 6.3 $\mathcal{P}$-Factorisability

In this section we propose the notion of *$\mathcal{P}$-factorisability*. We start by a motivating example and continue with some category theoretical preliminaries, which notationally eases the presentation of $\mathcal{P}$-factorisability.

### 6.3.1 An Example

Consider $\mathcal{M} = \mathcal{TS}_{Act}$ and $\mathcal{P} = \mathcal{P}_M$ from Sec. 6.2 and the transition systems below, which we denote—left to right—$T_1, \ldots, T_5$. The initial states are depicted as $\odot$.



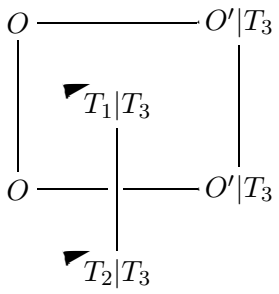
$T_1$ is strongly bisimilar ($\mathcal{P}$-bisimilar) to $T_2$. In fact, there is an obvious open map $k$ from $T_1$ to $T_2$. Considering $T_3$ to be fixed, we can define a functor $\_|T_3 : \mathcal{M} \longrightarrow \mathcal{M}$, where $|$ acts as a CCS-like parallel composition. $T_4 = T_1|T_3$ and $T_5 = T_2|T_3$ serve as an informal illustration of $\_|T_3$, when applied to $T_1$ and $T_2$, respectively. In much the same way as Milner [Mil89] shows that $P \sim P'$ implies $P \,|\, Q \sim P' \,|\, Q$, we would like to conclude that if $k : T_1 \longrightarrow T_2$ is open, then so is $T_1|T_3 \xrightarrow{k|T_3} T_2|T_3$. [1]

Recall that $\mathcal{P}$-bisimilarity is based on open maps, which again are based on observations from $\mathcal{P}$. E.g., we can observe $O$, the behaviour $\odot \xrightarrow{\alpha} \cdot \xrightarrow{\gamma} \cdot$, in $T_4$ and—via

[1] In fact, just as Milner uses a bisimulation $P \sim P'$ to exhibit a bisimulation $P \,|\, Q \sim P' \,|\, Q$, we will "factor" the observation $\odot \xrightarrow{\alpha} \cdot \xrightarrow{\gamma} \cdot$ into transitions from $T_3$ and from $T_1$ and $T_2$, respectively. This will guide us to the mediating morphism required in (6.2).

$k|T_3 : T_4 \longrightarrow T_5$—in $T_5$. Some of these transitions in $T_4$, here only the $\alpha$ transition, are due to transitions "from" $T_1$. Using $k$, we conclude that the $\alpha$ transition in $O$ must also be observable in $T_2$. In fact, we have a commuting diagram as in (6.1) with $X = T_4$, $Y = T_5$, $O_1 = O_2 = O$, $m = k|T_3$, and $f = 1_O$, and by the above we have extracted a second commuting diagram of the form (6.1) with $X = T_1$, $Y = T_2$, $O_1 = O_2 = O' = \odot \xrightarrow{\alpha} \cdot$, and $m = k$.

The way we have "factored" $O$ into $O'$ is consistent with $\_|T_3$ in the following sense: there exists a commuting diagram of the form




In the next section, we formalise this by defining the notion of $\mathcal{P}$-factorisability, and, as a consequence, we will be able to conclude that $k|T_3$ is an open map.

### 6.3.2 Categorical Preliminaries

Given a category $\mathcal{C}$ with objects $\mathcal{C}_0$ and morphisms (arrows) $\mathcal{C}_1$, let $\widehat{\mathcal{C}}$ be the category whose objects are $\mathcal{C}_1$ and whose morphisms represent commuting diagrams, i.e., there is a morphism $(h_1, h_2)$ from $f$ to $g$ if

$$\begin{array}{ccc} \cdot & \xrightarrow{h_1} & \cdot \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ \cdot & \xrightarrow[h_2]{} & \cdot \end{array} \tag{6.11}$$

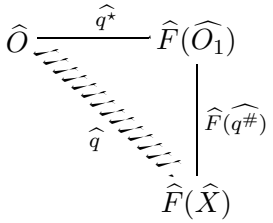
is a commuting diagram in $\mathcal{C}$. Composition of morphisms is defined component-wise. For notational convenience we may "hat" objects and morphisms from $\widehat{\mathcal{C}}$, e.g., $\widehat{X}$ and $\widehat{m}$. When convenient, we will denote objects from $\widehat{\mathcal{C}}$ as morphisms from $\mathcal{C}$, e.g., $\widehat{X}$ might be denoted $f$.

Notice that a functor $F : \mathcal{C} \longrightarrow \mathcal{D}$ induces a functor $\widehat{F} : \widehat{\mathcal{C}} \longrightarrow \widehat{\mathcal{D}}$, which sends an object $\widehat{X}$ to $F(\widehat{X})$ and a morphism $\widehat{m} = (m_1, m_2)$ to $(F(m_1), F(m_2))$.

### 6.3.3 Factorising Observations

**Definition 140 $\mathcal{P}$-factorisability**

A functor $F : \mathcal{M} \longrightarrow \mathcal{M}$ is said to be $\mathcal{P}$-*factorisable* if whenever we have an object $\widehat{O}$ in $\widehat{\mathcal{P}}$, an object $\widehat{X}$ in $\widehat{\mathcal{M}}$, and a morphism $\widehat{O} \xrightarrow{\widehat{q}} \widehat{F}(\widehat{X})$ in $\widehat{\mathcal{M}}$, then there exist an object $\widehat{O_1}$ in $\widehat{\mathcal{P}}$ and morphisms $\widehat{O} \xrightarrow{\widehat{q^\star}} \widehat{F}(\widehat{O_1})$ and $\widehat{O_1} \xrightarrow{\widehat{q^\#}} \widehat{X}$ in $\widehat{\mathcal{M}}$ such that the diagram



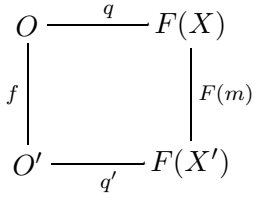

(6.12)

commutes in $\widehat{\mathcal{M}}$. □

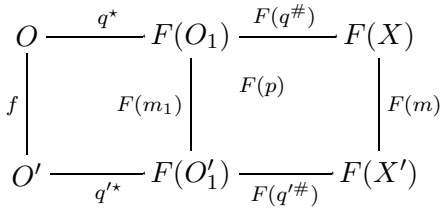
**Definition 141** A functor $F : \mathcal{M} \longrightarrow \mathcal{M}$ is a $\mathcal{P}$-*operator* if it is $\mathcal{P}$-bisimilarity preserving, i.e., if $A$ is $\mathcal{P}$-bisimilar to $B$, then $F(A)$ is $\mathcal{P}$-bisimilar to $F(B)$. □

**Theorem 142** *Any $\mathcal{P}$-factorisable functor $F : \mathcal{M} \longrightarrow \mathcal{M}$ is a* $\mathcal{P}$-operator.

**Proof.** It is sufficient to show that $F$ preserves open maps. Assume $m : X \longrightarrow X'$ is an open map and we are given a commuting diagram




This diagram is a morphism $\widehat{O} \xrightarrow{\widehat{q}} \widehat{F}(\widehat{X})$ in $\widehat{\mathcal{M}}$. By $\mathcal{P}$-factorisability there exist $\widehat{O_1}$ in $\widehat{\mathcal{P}}$ and morphisms $\widehat{O} \xrightarrow{\widehat{q^\star}} \widehat{F}(\widehat{O_1})$ and $\widehat{O_1} \xrightarrow{\widehat{q^\#}} \widehat{X}$ in $\widehat{\mathcal{M}}$ such that (6.12) commutes. Denote $\widehat{O}$ as $f : O \longrightarrow O'$, $\widehat{q}$ as $(q, q')$, $\widehat{O_1}$ as $m_1 : O_1 \longrightarrow O_1'$, $\widehat{q^\star}$ as $(q^\star, q'^\star)$, $\widehat{X}$ as $m : X \longrightarrow X'$, and $\widehat{q^\#}$ as $(q^\#, q'^\#)$. Since $\widehat{O_1} \xrightarrow{\widehat{q^\#}} \widehat{X}$ represents a commuting diagram and $m$ was open, there exists a morphism $p : O_1' \longrightarrow X$ such that the diagram




must commute (by (6.12)). But then

$$\begin{aligned} q &= F(q^\#) \circ q^\star \text{ , by (6.12)} \\ &= F(p) \circ F(m_1) \circ q_\star \\ &= (F(p) \circ q'^\star) \circ f \text{ ,} \end{aligned}$$

and

$$\begin{aligned} q' &= F(q'^{\#}) \circ q'^{\star} \text{ , by (6.12)} \\ &= F(m) \circ (F(p) \circ q'^{\star}) \text{ .} \end{aligned}$$

We conclude that $F(m)$ is open. Hence if $X \xleftarrow{m} Z \xrightarrow{n} Y$ is a span of open maps, $F(X) \xleftarrow{F(m)} F(Z) \xrightarrow{F(n)} F(Y)$ is a span of open maps. ■

## 6.4 Application, an Example

As an example of the application of the theory we consider the category $\mathcal{TS}$ of labelled transition systems [2] from [WN95]. As it is shown there, process-language constructs can be interpreted as universal constructions in $\mathcal{TS}$. In the following subsections, we show how our theory can be applied to the functors associated to these universal constructions. In Sect. 6.4.8 we only consider product, co-product, and restriction. In Sect. 6.5, we examine a recursion operator.

### 6.4.1 The Category of Labelled Transition Systems

In this section we define the category $\mathcal{TS}$ inspired by [WN95].

**Definition 143** The category $\mathcal{TS}$ has as objects $(S, i, L, \longrightarrow)$, labelled transition systems (lts) with labelling set $L$. We require that all states in $S$ be reachable (from the initial state $i$). □

We shall use the abbreviation $T_j$ for $(S_j, i_j, L_j, \longrightarrow_j)$. If clear from the context we will omit the subscript $j$. Also, all the following constructions do produce lts in $\mathcal{TS}$, i.e., all states are reachable.

For technical reasons we assume the existence of a special element $*$ which is not member of any labelling set. A partial function $\lambda$ between two labelling sets $L$ and $L'$ can then be represented as a total function from $L \cup \{*\}$ to $L' \cup \{*\}$ such that $*$ is mapped to $*$. If $a \in L$ is mapped to $*$, we interpret this as meaning that $\lambda$ is undefined on $a$. Overloading the symbol $\lambda$, we shall write this as $\lambda : L \hookrightarrow L'$. Given $T = (S, i, L, \longrightarrow)$, we define $\longrightarrow_*$ to be the set $\longrightarrow \cup \{(s, *, s) \mid s \in S\}$. The transitions $(s, *, s)$ are called *idle* transitions.

**Definition 144** A morphism $m : T_0 \longrightarrow T_1$ is a pair $f = (\sigma_m, \lambda_m)$, where $\sigma_m : S_0 \longrightarrow S_1$ and $\lambda_m : L_0 \hookrightarrow L_1$ are total functions such that

$$\sigma_m(i_0) = i_1 \tag{6.13}$$

$$s \xrightarrow{a}_0 s' \;\Rightarrow\; \sigma_m(s) \xrightarrow{\lambda(a)}_{1*} \sigma_m(s') \tag{6.14}$$

□

[2] This category is different from the one presented in Sec. 6.2; we use this category because it has universal constructions such as, e.g., products and co-products.

The intuition is that initial states are preserved and transitions in $T_0$ are simulated in $T_1$, except when $\lambda_m(a) = *$, in which case they represent inaction in $T_1$. Composition of morphisms is defined component-wise. This defines the category $\mathcal{TS}$. We suppress the subscript $m$ when no confusion is possible.

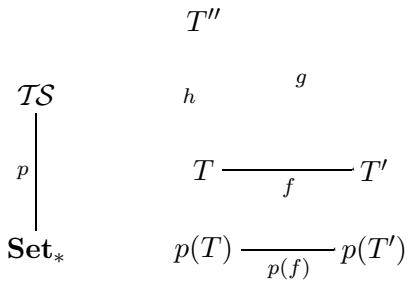
Let $\mathbf{Set}_*$ denote the category whose objects are labelling sets $L$ and whose morphisms are partial functions $\lambda : L \longrightarrow L'$ between labelling sets.

### 6.4.2 More Categorical Preliminaries, Fibred Category Theory

Let $p : \mathcal{TS} \longrightarrow \mathbf{Set}_*$ be the function which sends an lts to its labelling set and a morphism $(\sigma, \lambda) : T_0 \longrightarrow T_1$ to $\lambda : L_0 \longrightarrow L_1$. A *fibre* over $L$, $p^{-1}(L)$, is the subcategory of $\mathcal{TS}$ whose objects have labelling set $L$ and whose morphisms $f$ map to $1_L$, the identity function on $L$, under $p$.

We will use the following notions from fibred category theory.

**Definition 145** A morphism $f : T \longrightarrow T'$ in $\mathcal{TS}$ is said to be *Cartesian* with respect to $p : \mathcal{TS} \longrightarrow \mathbf{Set}_*$ if for any morphism $g : T'' \longrightarrow T'$ in $\mathcal{TS}$ such that $p(g) = p(f)$ there is a unique morphism $h : T'' \longrightarrow T$ such that $p(h) = 1_{p(T)}$ and $f \circ h = g$.



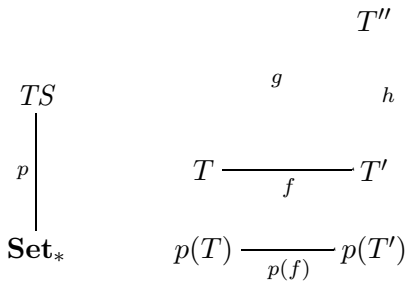

A Cartesian morphism $f : T \longrightarrow T'$ in $\mathcal{TS}$ is said to be a *Cartesian lifting* of the morphism $p(f)$ in $\mathbf{Set}_*$ with respect to $T'$. □

It can be shown now that $p$ is a *fibration*, i.e.,

- any morphism $\lambda : L \longrightarrow L'$ in $\mathbf{Set}_*$ has a Cartesian lifting with respect to any $T'$ in $TS$ such that $p(T') = L'$.
- any composition of Cartesian morphisms is itself Cartesian.

Dually, we define a morphism to be *co-Cartesian*.

**Definition 146** A morphism $f : T \longrightarrow T'$ in $TS$ is said to be *co-Cartesian* with respect to $p : TS \longrightarrow \mathbf{Set}_*$ if for any morphism $g : T \longrightarrow T''$ in $TS$ such that $p(g) = p(f)$ there is a unique morphism $h : T' \longrightarrow T''$ such that $p(h) = 1_{p(T')}$ and $h \circ f = g$.




A co-Cartesian morphism $f : T \longrightarrow T'$ in $TS$ is said to be a *co-Cartesian lifting* of the morphism $p(f)$ in $\mathbf{Set}_*$ with respect to $T'$. □

Similarly, it can be shown that $p$ is a *co-fibration*, i.e., $p^{op} : TS^{op} \longrightarrow \mathbf{Set}_*^{op}$ is a fibration.

In the following, let $\mathcal{U}$ be $\mathcal{TS}$, let $\mathcal{F}$ be the union of all fibres over all labelling sets, and let $\mathcal{M}$ be the subcategory of $\mathcal{F}$ induced by all non-restarting lts, i.e., there are no transitions into the initial state. The reason for staying within fibres is that one commonly insists on having labelled actions simulated by identically labelled actions. Notice that $\mathcal{TS}_{Act}$ from Sect. 6.2 can be viewed as the fibre $p^{-1}(Act)$. Morphisms in $\mathcal{M}$ will always be of the form $(\sigma, 1_L)$, for some labelling set $L$. In particular, all commuting diagrams of the form (6.1) in $\mathcal{M}$ will always belong to some fibre $p^{-1}(L)$. It can also be shown that $\mathcal{M}$ has pullbacks, hence $\sim_{\mathcal{P}}$ is an equivalence relation [JNW93]. The reason we consider non-restarting lts is technical. We will address this issue below.

We shall assume that the category $\mathcal{P}$ of observation is closed under renaming of states and closed under variation of labelling sets, i.e., if $(S, i, L, \longrightarrow)$ is an observation and $L'$ is any labelling set such that $(S, i, L', \longrightarrow)$ is an lts, then it is also an observation.

To emphasise the use of the theory in Sect. 6.3, we will use the notation $\mathcal{M}$ and $\mathcal{P}$.

### 6.4.3 Product

In this section, we consider the product construction, which has strong relations to, e.g., CCS's parallel composition operator, see [WN95] and Sect. 6.4.8. In [WN95], it is shown how CCS's parallel composition operator can be expressed using the product, renaming, and relabelling operators we present below.

**Definition 147** Let $T_0 \times T_1$ denote $(S, i, L, \longrightarrow)$, where

- $S = S_0 \times S_1$, with $i = (i_0, i_1)$ and projections $\rho_0 : S \longrightarrow S_0$, $\rho_1 : S \longrightarrow S_1$ ,
- $L = L_0 \times_* L_1 = (L_0 \times \{*\}) \cup (\{*\} \times L_1) \cup (L_0 \times L_1)$, with projections $\pi_0 : L_0 \times_* L_1 \hookrightarrow L_0$ and $\pi_1 : L_0 \times_* L_1 \hookrightarrow L_1$, and
- $s \xrightarrow{a}_* s' \;\Leftrightarrow\; \rho_0(s) \xrightarrow{\pi_0(a)}_{0*} \rho_0(s') \;\wedge\; \rho_1(s) \xrightarrow{\pi_1(a)}_{1*} \rho_1(s')$ .

□

Let $\Pi_0 = (\rho_0, \pi_0) : T_0 \times T_1 \longrightarrow T_0$ and $\Pi_1 = (\rho_1, \pi_1) : T_0 \times T_1 \longrightarrow T_1$. It can be shown that this construction is a product of $T_0$ and $T_1$ in the category $\mathcal{TS}$.

The product construction allows the two components $T_0$ and $T_1$ to proceed independently of each as well as synchronising on any of their actions. This behaviour is far to generous compared to CCS's parallel composition. However, by restricting away all action pairs from $T_0 \times T_1$ that are not of the form $(a, *)$, $(*, a)$, or $(a, \overline{a})$, corresponding to a move in the left component, right component, and a synchronisation on complementary actions, and relabelling $(a, *)$, $(*, a)$, and $(a, \overline{a})$ to $a$,$a$, and $\tau$, respectively, we obtain CCS's parallel composition. Both restriction and relabelling can be handled in our setting.

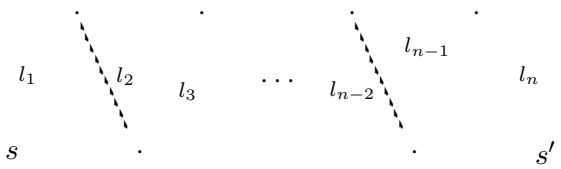
For a fixed $\mathsf{lts}$ $T_0$ the above construction induces an obvious functor $T_0 \times \_ : \mathcal{M} \longrightarrow \mathcal{M}$. We continue by applying our theory to prove a general result for this functor. First we need a definition, which will help formalising the "factoring" of observations in a product object.

**Definition 148** Let $T = (S, i, L, \longrightarrow)$ and let $\lambda : L \hookrightarrow L'$ represent a partial function between labelling sets. Let $\equiv$ be the least equivalence relation on $S$ such that if $s \xrightarrow{a} s'$ and $\lambda(a) = *$, then $s \equiv s'$. Let $[s]$ denote the equivalence class of $s$ under $\equiv$. Define $[T]_\lambda = (S', i', L', \longrightarrow')$, where

- $S' = \{[s] \mid s \in S\}$ and $i' = [i]$,
- $[s] \xrightarrow{b}{}' [s'] \;\Leftrightarrow\; \exists v \in [s], v' \in [s'], a \in L.\, v \xrightarrow{a} v' \;\wedge\; \lambda(a) = b \neq * \,.$

Let $\eta_{(T,\lambda)} : T \longrightarrow [T]_\lambda$ be the pair $(\sigma, \lambda)$, where $\sigma(s) = [s]$. □

A simple argument shows that $\sigma$ is well-defined. If $s \equiv s'$, then there exists a "back and forth" path




where $l_i = *$ or $\lambda(l_i) = *$, for $1 \leq i \leq n$. We conclude that $\sigma(s) = \sigma(s')$.

**Proposition 149** *The morphism $\eta_{(T,\lambda)} : T \longrightarrow [T]_\lambda$ is co-Cartesian with respect to $p$.*

**Proof.** Assume $f : T \longrightarrow T_1$ and $p(f) = p(\eta_{(T,\lambda)})$. Define $(\sigma', 1_{L'}) : [T]_\lambda \longrightarrow T_1$ by $\sigma'([s]) = \sigma_f(s)$. By an argument similar to the above one can show that $\sigma'$ is well-defined. To see that $(\sigma', 1_{L'})$ is a morphism first notice that $\sigma'([i]) = \sigma_f(i) = i_1$. Next, assume $[s] \xrightarrow{b}{}' [s']$, i.e., $\exists v \in [s], v' \in [s'], a \in L.\, v \xrightarrow{a} v' \;\wedge\; \lambda(a) = b \neq *$. Then $\sigma_f(v) \xrightarrow{\lambda(a)}{}_{1*} \sigma_f(v')$, i.e., $\sigma'([s]) \xrightarrow{b}{}_{1*} \sigma'([s'])$. It is easy to see that $(\sigma', 1_{L'})$ is the uniquely determined morphism such that $p((\sigma', 1_{L'})) = 1_{p([T]_\lambda)}$ and $f = (\sigma', 1_{L'}) \circ \eta_{(T,\lambda)}$. ■

**Lemma 150** *For a partial function $\lambda : L \hookrightarrow L'$ between labelling sets, there is a functor $F_\lambda : p^{-1}(L) \longrightarrow p^{-1}(L')$ which sends $f = (\sigma, 1_L) : T_0 \longrightarrow T_1$ to $F_\lambda(f) = (\gamma, 1_{L'}) : [T_0]_\lambda \longrightarrow [T_1]_\lambda$ defined by $\gamma([s]) = [\sigma(s)]$.*

**Proof.** The proof is routine, hence omitted. ∎

We can now show the following theorem.

**Theorem 151** *Let $T_0$ belong to $\mathcal{M}$ and $L_0 = p(T_0)$. Let $\mathcal{P}$ be any subcategory of $\mathcal{U}$ such that whenever we have $O \xrightarrow{f} O'$ in $\mathcal{P}$, where $p(f) = 1_{L_0 \times_* L}$ for some $L$, then $F_{\pi_1}(O) \xrightarrow{F_{\pi_1}(f)} F_{\pi_1}(O')$ also belongs to $\mathcal{P}$. Then $T_0 \times \_ : \mathcal{M} \longrightarrow \mathcal{M}$ is a $\mathcal{P}$-operator.*

**Proof.** By Theorem 142 it is sufficient to show that $T_0 \times \_$ is $\mathcal{P}$-factorisable. So assume $T \xrightarrow{m} T'$ belongs to $\mathcal{M}$, $p(T) = L$, and we are given $\widehat{O} \xrightarrow{\widehat{q}} \widehat{T_0 \times}(\widehat{T})$, i.e., a commuting diagram in $\mathcal{M}$

$$\begin{array}{ccc} O & \xrightarrow{\quad q \quad} & T_0 \times T \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle T_0 \times m} \\ O' & \xrightarrow[\quad q' \quad]{} & T_0 \times T' \end{array}$$

Since $\mathcal{M}$ is the union of fibres we have $p(f) = p(q) = p(q') = p(T_0 \times m) = 1_{L_0 \times_* L}$ for some set $L$. Let $\pi_1 : L_0 \times_* L \hookrightarrow L$ be the projection on the second component. By our assumptions $F_{\pi_1}(O) \xrightarrow{F_{\pi_1}(f)} F_{\pi_1}(O')$ is in $\mathcal{P}$. Let $O_1 = F_{\pi_1}(O)$, $O_1' = F_{\pi_1}(O')$, $q = (\sigma_q, 1_{L_0 \times_* L})$, and $q' = (\sigma_{q'}, 1_{L_0 \times_* L})$. Define

$q^\# = (\sigma, 1_L) : O_1 \longrightarrow T$ , where $\sigma([s]) = \rho_1(\sigma_q(s))$, and

$q'^\# = (\sigma', 1_L) : O_1' \longrightarrow T'$ , where $\sigma'([s']) = \rho_1'(\sigma_{q'}(s'))$

$\rho_1$ and $\rho_1'$ are the projections mentioned in Definition 147. Notice, e.g., that for any $s_1, s_2 \in [s]$ in $O_1$ we have $\rho_1(\sigma_q(s_1)) = \rho_1(\sigma_q(s_2))$. Next, define

$q^\star = (\gamma, 1_{L_0 \times_* L}) : O \longrightarrow T_0 \times O_1$ , where $\gamma(s) = (\rho_0(\sigma_q(s)), [s])$, and

$q'^\star = (\gamma', 1_{L_0 \times_* L}) : O' \longrightarrow T_0 \times O_1'$ , where $\gamma'(s') = (\rho_0'(\sigma_{q'}(s')), [s'])$

It can now be shown that both diagrams

$$\begin{array}{ccc} O & \xrightarrow{\quad q^\star \quad} & T_0 \times O_1 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle 1_{T_0} \times F_{\pi_1}(f)} \\ O' & \xrightarrow[\quad q'^\star \quad]{} & T_0 \times O_1' \end{array} \qquad \begin{array}{ccc} O_1 & \xrightarrow{\quad q^\# \quad} & T \\ {\scriptstyle F_{\pi_1}(f)}\downarrow & & \downarrow{\scriptstyle m} \\ O_1' & \xrightarrow[\quad q'^\# \quad]{} & T' \end{array}$$

exist in $\mathcal{M}$ and commute, i.e., we have morphisms $\widehat{O} \xrightarrow{\widehat{q^\star}} \widehat{T_0 \times}(\widehat{O_1})$ and $\widehat{O_1} \xrightarrow{\widehat{q^\#}} \widehat{T}$ in $\widehat{\mathcal{M}}$. It can also be shown that $q = q^\# \circ q^\star$ and $q' = q'^\# \circ q'^\star$. Hence we have a commuting diagram of the form (6.12). Hence $T_0 \times \_$ is $\mathcal{P}$-factorisable. ∎

### 6.4.4 Co-Product

In this section, we consider the co-product construction, which has strong relations to, e.g., CCS's nondeterministic choice operator, see [WN95] and Sect. 6.4.8.

**Definition 152** Let $T_0 + T_1$ denote $(S, i, L, \longrightarrow)$, where

- $S = (S_0 \times \{i_1\}) \cup (\{i_0\} \times S_1)$, with $i = (i_0, i_1)$ and injections $in_0 : S_0 \longrightarrow S$, $in_1 : S_1 \longrightarrow S$ ,
- $L = L_0 \cup_* L_1 = (L_0 \times \{*\}) \cup (\{*\} \times L_1)$, with injections $j_0 : L_0 \longrightarrow L$ and $j_1 : L_1 \longrightarrow L$, and
- $s \xrightarrow{a} s' \;\Leftrightarrow\; \exists v \xrightarrow{b}_0 v'.\, (in_0(v), j_0(b), in_0(v')) = (s, a, s')$ or

$$\exists v \xrightarrow{b}_1 v'.\, (in_1(v), j_1(b), in_1(v')) = (s, a, s')$$

□

Let $I_0 = (in_0, j_0) : T_0 \longrightarrow T_0 + T_1$ and $I_1 = (in_1, j_1) : T_1 \longrightarrow T_0 + T_1$. It can be shown that this construction is a coproduct of $T_0$ and $T_1$ in the category $\mathcal{TS}$.

As opposed to the product construction, the co-product construction resembles more a process algebraic choice, "+", operator. If we consider non-restarting lts, co-product can be shown to correspond to "+" in a formal sense [WN95].

**Definition 153** Given $T' = (S', i', L', \longrightarrow')$ and a partial function $\lambda : L \hookrightarrow L'$. Let $T'_{\downarrow\lambda} = (S, i, L, \longrightarrow)$, where

- $S = \{s \in S' \mid \exists a_1, \ldots, a_n \in L, s_1, \ldots, s_n \in S'.$
  $i' \xrightarrow{\lambda(a_1)}' s_1 \xrightarrow{\lambda(a_2)}' \ldots \xrightarrow{\lambda(a_n)}' s_n \wedge s_n = s\}$
- $i = i'$
- $s \xrightarrow{b} s' \;\Leftrightarrow\; s \xrightarrow{\lambda(b)}' s'$

Let $\eta_{(T',\lambda)} : T'_{\downarrow\lambda} \longrightarrow T'$ be the pair $(in, \lambda)$, where $in$ is the injection function. □

**Proposition 154** *The morphism $\eta_{(T',\lambda)} : T'_{\downarrow\lambda} \longrightarrow T'$ is Cartesian with respect to $p$.*

**Lemma 155** *For a partial function $\lambda : L \hookrightarrow L'$ between labelling sets, there is a functor $F_{\downarrow\lambda} : p^{-1}(L') \longrightarrow p^{-1}(L)$ which sends $f = (\sigma, 1_{L'}) : T_0 \longrightarrow T_1$ to $F_{\downarrow\lambda} = (\gamma, 1_L) : T_{0\downarrow\lambda} \longrightarrow T_{1\downarrow\lambda}$ defined by $\gamma(s) = \sigma(s)$.*

**Theorem 156** *Let $T_0$ belong to $\mathcal{M}$ and $L_0 = p(T_0)$. Assume $\mathcal{P}$ is a subcategory of $\mathcal{M}$ such that whenever we have $O \xrightarrow{f} O'$ in $\mathcal{P}$ with $p(f) = 1_{L_0 \cup_* L}$ for some $L$, $F_{\downarrow\lambda}(O) \xrightarrow{F_{\downarrow\lambda}(f)} F_{\downarrow\lambda}(O')$ also belongs to $\mathcal{P}$, where $\lambda : L \longrightarrow L_0 \cup_* L$ is the injection function. Then $T_0 + \_ : \mathcal{M} \longrightarrow \mathcal{M}$ is a $\mathcal{P}$-operator.*

**Proof.** It is sufficient to show that $T_0 + \_$ is $\mathcal{P}$-factorisable. So assume $T \xrightarrow{m} T'$ belongs to $\mathcal{M}$, $p(T) = L$, and we are given $\widehat{O} \xrightarrow{\widehat{q}} \widehat{T_0+}(\widehat{T})$, i.e., a commuting diagram in $\mathcal{M}$

$$\begin{array}{ccc} O & \xrightarrow{\quad q \quad} & T_0 + T \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle 1_{T_0}+m} \\ O' & \xrightarrow[\quad q' \quad]{} & T_0 + T' \end{array}$$

Let $p(f) = 1_{L_0 \cup_* L}$. Let $\lambda : L \longrightarrow L_0 \cup_* L$ be the injection function sending $a \in L$ to $(*, a) \in L_0 \cup_* L$. By our assumptions $F_{\downarrow\lambda}(O) \xrightarrow{F_{\downarrow\lambda}(f)} F_{\downarrow\lambda}(O')$ is in $\mathcal{P}$. Let $O_1 = F_{\downarrow\lambda}(O)$, $O_1' = F_{\downarrow\lambda}(O')$, $q = (\sigma_q, 1_{L_0 \cup_* L})$, and $q' = (\sigma_{q'}, 1_{L_0 \cup_* L})$. Define

$q^{\#} = (\sigma, 1_L) : O_1 \longrightarrow T$ , where $\sigma(s) = t$, where $\sigma_q(s) = (r, t)$, and

$q'^{\#} = (\sigma', 1_L) : O_1' \longrightarrow T'$ , where $\sigma'(s') = t'$, where $\sigma_{q'}(s') = (r', t')$

Next, define

$q^{\star} = (\gamma, 1_{L_0 \cup_* L}) : O \longrightarrow T_0 + O_1$ , where $\gamma(s) = (r, i_1)$ if $\sigma_q(s) = (r, i)$,
$\gamma(s) = (i_0, t)$ if $\sigma_q(s) = (i_0, t)$, and

$q'^{\star} = (\gamma', 1_{L_0 \cup_* L}) : O' \longrightarrow T_0 + O_1'$ , where $\gamma'(s') = (r', i_1')$ if $\sigma_{q'}(s') = (r', i')$,
$\gamma'(s') = (i_0', s')$ if $\sigma_{q'}(t') = (i_0', t')$

It can now be shown that both diagrams

$$\begin{array}{ccc} O & \xrightarrow{\quad q^{\star} \quad} & T_0 + O_1 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle 1_{T_0}+F_{\downarrow\lambda}(f)} \\ O' & \xrightarrow[\quad q'^{\star} \quad]{} & T_0 + O_1' \end{array} \qquad \begin{array}{ccc} O_1 & \xrightarrow{\quad q^{\#} \quad} & T \\ {\scriptstyle F_{\downarrow\lambda}(f)}\downarrow & & \downarrow{\scriptstyle m} \\ O_1' & \xrightarrow[\quad q'^{\#} \quad]{} & T' \end{array}$$

exist in $\mathcal{M}$ and commute, i.e., we have morphisms $\widehat{O} \xrightarrow{\widehat{q^{\star}}} \widehat{T_0+}(\widehat{O_1})$ and $\widehat{O_1} \xrightarrow{\widehat{q^{\#}}} \widehat{T}$ in $\widehat{\mathcal{M}}$. It can also be shown that $q = q^{\#} \circ q^{\star}$ and $q' = q'^{\#} \circ q'^{\star}$. Hence we have a commuting diagram of the form (6.12). Hence $T_0 + \_$ is $\mathcal{P}$-factorisable. ■

### 6.4.5 Restriction

In this section, we consider relabelling.

**Definition 157** Given $T' = (S', i', L', \longrightarrow')$ and a labelling set $L$. Let $F\downarrow_L : \mathcal{M} \longrightarrow \mathcal{M}$ denote the functor which sends $T'$ to $T = (S, i, L, \longrightarrow)$, where

- $S = \{s \in S' \mid \exists a_1, \ldots, a_n \in L \cap L', s_1, \ldots, s_n \in S'.$
  $i' \xrightarrow{a_1}{}' s_1 \xrightarrow{a_2}{}' \cdots \xrightarrow{a_n}{}' s_n \wedge s_n = s\}$
- $i = i'$

- $s \xrightarrow{a} s' \;\Leftrightarrow\; s \xrightarrow{a} {}' s',\ a \in L$

and which maps a morphism $m = (\sigma'_m, 1_{L'}) : T'_1 \longrightarrow T'_2$ to $F\downarrow_L (m) = (\sigma_m, 1_L) : F\downarrow_L (T'_1) \longrightarrow F\downarrow_L (T'_2)$, where $\sigma_m(s) = \sigma'_m(s)$. □

We have the following perhaps surprising result.

**Theorem 158** *For any choice of $\mathcal{P}$ the functor $F\downarrow_L$ is a $\mathcal{P}$-operator.*

**Proof.** We show that $F\downarrow_L$ is a $\mathcal{P}$-operator. Assume $T \xrightarrow{m} T'$ and we have

$$\begin{array}{ccc} O & \xrightarrow{\quad q \quad} & F\downarrow_L(T) \\ {\scriptstyle f}\big\downarrow & & \big\downarrow{\scriptstyle F\downarrow_L(m)} \\ O' & \xrightarrow[\quad q' \quad]{} & F\downarrow_L(T') \end{array}$$

that commutes in $\mathcal{M}$. Let $p(T) = L'$. By our assumptions we must have a commuting diagram

$$\begin{array}{ccc} O_1 & \xrightarrow{\quad q^{\#} \quad} & T \\ {\scriptstyle m_1}\big\downarrow & & \big\downarrow{\scriptstyle m} \\ O'_1 & \xrightarrow[\quad q'^{\#} \quad]{} & T' \end{array}$$

where $O = (S, i, L, \longrightarrow)$, $O' = (S', i', L, \longrightarrow)$, $f = (\sigma_f, 1_L)$, $O_1 = (S, i, L', \longrightarrow)$, $O'_1 = (S', i', L', \longrightarrow {}')$, $m_1 = (\sigma_f, 1_{L'})$, $q = (\sigma_q, 1_L)$, $q' = (\sigma_{q'}, 1_L)$, $q^{\#} = (\sigma_q, 1_{L'})$, and $q'^{\#} = (\sigma_{q'}, 1_{L'})$. Notice $F\downarrow_L(O_1) = O$, $F\downarrow L_(O'_1) = O'$, and $F\downarrow_L(m_1) = f$. It can easily be shown that we have a diagram in $\widehat{\mathcal{M}}$ as required in (6.12) and that it commutes. ■

### 6.4.6 Relabelling

Relabelling, as presented in [WN95], is a bit tricky. We will need some auxiliary definitions and we will have to consider (relabelling) functors between fibres.

**Definition 159** Let $T = (S, i, L, \longrightarrow)$ be an lts and $\lambda : L \longrightarrow L'$ be a total function between labelling sets. Define $T\{\lambda\}$ to be the lts $(S, i, L', \longrightarrow {}')$, where

$$s \xrightarrow{a} {}' s' \;\Leftrightarrow\; \exists b.\, s \xrightarrow{b} s' \;\wedge\; \lambda(b) = a\ .$$

□

**Proposition 160** *If $\lambda : L \longrightarrow L'$ is a total function in $\mathbf{Set}_*$, then $T \xrightarrow{f} T\{\lambda\}$, where $f = (1_S, \lambda)$ is co-Cartesian with respect to $p$.*

**Proof.** The proof is routine, hence omitted. ■

Any total function $\lambda : L \longrightarrow L'$ induces a functor $F\{\lambda\} : p^{-1}(L) \longrightarrow p^{-1}(L')$. Notice that $F\{\lambda\}$ is not an endofunctor on $\mathcal{M}$. Instead, given $\lambda : L \longrightarrow L'$ we consider $\lambda' : L \cup L' \longrightarrow L \cup L'$ defined by $\lambda'(a) = \lambda(a)$ if $a \in L$ and $\lambda'(a) = a$ otherwise. Now $p^{-1}(L)$ and $p^{-1}(L')$ embed fully and faithfully in $p^{-1}(L \cup L')$. We will therefore only consider total relabelling functions of the form $\lambda : L \longrightarrow L$.

Let $p_0 : TS \longrightarrow \mathbf{Set}$ be the functor which sends $T$ to $S$ and $(\sigma, \lambda) : T \longrightarrow T'$ to $\sigma$.

**Definition 161** Let $F^{-1}\{\lambda\}(T)$ denote the subcategory of $p^{-1}(L)$ whose objects are $\mathsf{ltss}$ $T'$ such that $F\{\lambda\}(T') = T$ and whose morphisms $f$ map to $1_{p_0(T)}$ under $p_0$; objects in $F^{-1}\{\lambda\}(T)$ have the same set of states as $T$.

An object $T'$ in $F^{-1}\{\lambda\}(T)$ is *minimal* if the only morphisms in $F^{-1}\{\lambda\}(T)$ with codomain $T'$ is the identity morphism on $T'$. □

*Remark.* Notice that if $T'$ is minimal in $F^{-1}\{\lambda\}(T)$, then for any two transitions $s \xrightarrow{a} s'$ and $s \xrightarrow{b} s'$ in $T'$ we have $a \neq b$ implies $\lambda(a) \neq \lambda(b)$.

**Theorem 162** *Given a total relabelling function $\lambda : L \longrightarrow L$. Choose $\mathcal{M} = p^{-1}(L)$. Let $\mathcal{P}$ be a subcategory of $\mathcal{U}$. Assume that for all $O \xrightarrow{f} O'$ in $\mathcal{P}$, where $f = (\sigma_f, 1_L)$ and $F^{-1}\{\lambda\}(O)$ and $F^{-1}\{\lambda\}(O')$ are nonempty, $(\sigma_f, 1_L) : O_1 \longrightarrow O_1'$ belongs to $\mathcal{P}$, whenever $O_1$ and $O_1'$ are minimal elements in $F^{-1}\{\lambda\}(O)$ and $F^{-1}\{\lambda\}(O')$, respectively, and $(\sigma_f, 1_L) : O_1 \longrightarrow O_1'$ defines a morphism. Then $F\{\lambda\} : \mathcal{M} \longrightarrow \mathcal{M}$ is a $\mathcal{P}$-operator.*

**Proof.** Choose $\mathcal{M} = p^{-1}(L)$. We show that $F\{\lambda\} : \mathcal{M} \longrightarrow \mathcal{M}$ is a $\mathcal{P}$-operator, where $\lambda : L \longrightarrow L$ is a total relabelling function. Assume $T \xrightarrow{m} T'$ belongs to $\mathcal{M}$ and we have

$$\begin{array}{ccc} O & \xrightarrow{\quad q \quad} & F\{\lambda\}(T) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle F\{\lambda\}(m)} \\ O' & \xrightarrow[\quad q' \quad]{} & F\{\lambda\}(T') \end{array}$$

that commutes in $\mathcal{M}$. Since $O$ is simulated in $F\{\lambda\}(T)$ we know that $F^{-1}\{\lambda\}(O)$ is nonempty. Similarly, $F^{-1}\{\lambda\}(O')$ is nonempty. Since $O$ is simulated in $F\{\lambda\}(T)$ and $p(m) = 1_L$, there must exist a minimal $O_1$ in $F^{-1}\{\lambda\}(O)$ and a minimal $O_1'$ in $F^{-1}\{\lambda\}(O')$ such that $g = (\sigma_f, 1_L) : O_1 \longrightarrow O_1'$ is a well-defined morphism in $\mathcal{P}$ and such that

$q^{\#} = (\sigma_q, 1_L) : O_1 \longrightarrow T$ , where $q = (\sigma_q, 1_L) : O \longrightarrow F\{\lambda\}(T)$, and

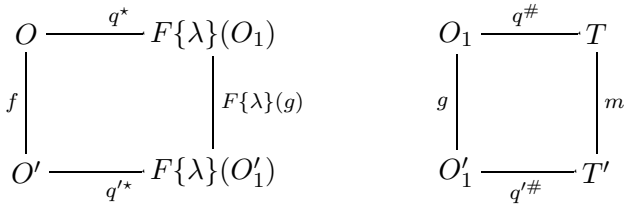
$q'^{\#} = (\sigma_{q'}, 1_L) : O_1' \longrightarrow T'$ , where $q' = (\sigma_{q'}, 1_L) : O' \longrightarrow F\{\lambda\}(T')$
are well-defined morphisms in $\mathcal{M}$.

Next, define

$q^{\star} = (\gamma, 1_L) : O \longrightarrow F\{\lambda\}(O_1)$ , where $\gamma(s) = s$, and

$q'^{\star} = (\gamma', 1_L) : O' \longrightarrow F\{\lambda\}(O_1')$ , where $\gamma'(s') = s'$

It can now be shown that both diagrams




exist in $\mathcal{M}$ and commute, i.e., we have morphisms $\widehat{O} \xrightarrow{\widehat{q^\star}} \widehat{F\{\lambda\}}(\widehat{O_1})$ and $\widehat{O_1} \xrightarrow{\widehat{q^\#}} \widehat{T}$ in $\widehat{\mathcal{M}}$. It can also be shown that $q = q^\# \circ q^\star$ and $q' = q'^\# \circ q'^\star$. Hence we have a commuting diagram of the form (6.12). Hence $F\{\lambda\}$ is $\mathcal{P}$-factorisable. ■

Notice that $\mathcal{M} = p^{-1}(L)$ is no restriction in our case, since $\mathcal{M}$ "consists" of full subcategories of fibres: it is easy to see that a $\mathcal{P}$-open morphism in $p^{-1}(L)$ is also $\mathcal{P}$-open in $\mathcal{M}$.

### 6.4.7 Prefix

**Definition 163** Given $T = (S, i, L, \longrightarrow)$ and a label $\alpha$. Let $\alpha.T = (S', i', L \cup \{\alpha\}, \longrightarrow')$, where

- $S' = \{\{s\} \mid s \in S\} \cup \{\emptyset\}$, $i' = \emptyset$, and
- $v \xrightarrow{b}{}' v' \;\Leftrightarrow\; (v = \emptyset \wedge b = \alpha \wedge v' = \{i\})$ or $(v = \{s\} \wedge v' = \{s'\} \wedge s \xrightarrow{b} s')$ .

□

Any label $\alpha$ induces a functor $\alpha._- : \mathcal{M} \longrightarrow \mathcal{M}$ which sends $f = (\sigma, 1_L) : T \longrightarrow T'$ to $(\sigma', 1_{L\cup\{\alpha\}}) : \alpha.T \longrightarrow \alpha.T'$, where $\sigma'(\emptyset) = \emptyset$ and $\sigma'(\{s\}) = \{\sigma(s)\}$.

**Definition 164** Given $T$ and a label $\alpha$. Let $\alpha^{-1}(T) = (S', i', L, \rightarrow')$, where

- $S'' = \{s \in S \mid \exists v \in L^*.\, i \xrightarrow{\alpha} \xrightarrow{v} s\} \backslash \{s \mid i \xrightarrow{\alpha} s\}$ ,
- $S' = \{\{s\} \mid s \in S''\} \cup \{\{s \mid i \xrightarrow{\alpha} s\}\}$ ,
- $i = \{s \mid i \xrightarrow{\alpha} s\}$ , and
- $r \xrightarrow{a}{}' r' \;\Leftrightarrow\; \exists s \in r, s' \in r'.\, s \xrightarrow{a} s'$ .

□

Any label $\alpha$ induces a functor $\alpha^{-1} : \mathcal{U} \longrightarrow \mathcal{U}$ which sends $f = (\sigma, 1_L) : T \longrightarrow T'$ to $\alpha^{-1}(f) = (\sigma', 1_L) : T_1 \longrightarrow T_2$, where $T_1 = \alpha^{-1}(T)$, $T_2 = \alpha^{-1}(T')$, $\sigma'(i_1) = i_2$, and $\sigma'(\{s\})$ is the unique $v \in S_2$ such that $\sigma(s) \in v$. Notice that $\alpha^{-1}(T)$ may not be non-restarting even though $T$ is.

**Theorem 165** *Let $\mathcal{P}$ be a subcategory of $\mathcal{U}$. Assume that whenever we have $O \xrightarrow{f} O'$ in $\mathcal{P}$, then $\alpha^{-1}(O) \overset{\alpha^{-1}(f)}{\longrightarrow} \alpha^{-1}(O')$ also belongs to $\mathcal{P}$. Then $\alpha.\_$ is a $\mathcal{P}$-operator.*

**Proof.** We show that $\alpha.\_$ is a $\mathcal{P}$-operator. Assume $T \xrightarrow{m} T'$ and we have

$$\begin{array}{ccc} O & \xrightarrow{q} & \alpha.T \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \alpha.m} \\ O' & \xrightarrow[q']{} & \alpha.T' \end{array}$$

that commutes in $\mathcal{M}$. Notice that since $T$ and $T'$ are assumed to be non-restarting, $\alpha^{-1}(O)$ and $\alpha^{-1}(O')$ must also be non-restarting. Assume $\alpha \in L = p(T)$. By our assumptions $\alpha^{-1}(O) \overset{\alpha^{-1}(f)}{\longrightarrow} \alpha^{-1}(O')$ is in $\mathcal{P}$. Let $O_1 = \alpha^{-1}(O)$ and $O_1' = \alpha^{-1}(O')$. Define

$q^{\#} = (\sigma, 1_L) : O_1 \longrightarrow T$ , given by $\sigma(i_1) = i$ and $\sigma(\{s\}) = r$,
where $\sigma_q(s) = \{r\}$ and $q = (\sigma_q, 1_L) : O \longrightarrow \alpha.T$, and

$q'^{\#} = (\sigma', 1_L) : O_1' \longrightarrow T'$ , given by $\sigma'(i_1') = i'$ and $\sigma'(\{s'\}) = r'$,
where $\sigma_{q'}(s') = \{r'\}$ and $q' = (\sigma_{q'}, 1_L) : O' \longrightarrow \alpha.T'$

Next, define

$q^{\star} = (\gamma, 1_L) : O \longrightarrow \alpha.O_1$ , where $\gamma(i) = \emptyset$,
$\gamma(s) = \{i_1\}$ for $s \in \{s\,|, i \xrightarrow{\alpha} s\}$ in $O$, $\gamma(s) = \{\{s\}\}$, else, and

$q'^{\star} = (\gamma', 1_L) : O' \longrightarrow \alpha.O_1'$ , where $\gamma'(i') = \emptyset$,
$\gamma'(s') = \{i_1'\}$ for $s' \in \{s'\,|, i' \xrightarrow{\alpha} s'\}$ in $O'$, $\gamma'(s') = \{\{s'\}\}$, else.

It can now be shown that both diagrams

$$\begin{array}{ccc} O & \xrightarrow{q^{\star}} & \alpha.O_1 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \alpha.g} \\ O' & \xrightarrow[q'^{\star}]{} & \alpha.O_1' \end{array} \qquad \begin{array}{ccc} O_1 & \xrightarrow{q^{\#}} & T \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle m} \\ O_1' & \xrightarrow[q'^{\#}]{} & T' \end{array}$$

exist in $\mathcal{M}$ and commute, i.e., we have morphisms $\widehat{O} \xrightarrow{\widehat{q^{\star}}} \widehat{\alpha.}(\widehat{O_1})$ and $\widehat{O_1} \xrightarrow{\widehat{q^{\#}}} \widehat{T}$ in $\widehat{\mathcal{M}}$. It can also be shown that $q = q^{\#} \circ q^{\star}$ and $q' = q'^{\#} \circ q'^{\star}$. Hence we have a commuting diagram of the form (6.12).

For the case where $\alpha \notin p(T)$ the same reasoning can be used. First extend $T$ and $T'$'s labelling sets to include $\alpha$. The induced $m_{\alpha} : T \longrightarrow T'$ in $p^{-1}(L \cup \{\alpha\})$ will be $\mathcal{P}$-open if and only if $m : T \longrightarrow T'$ is due to our assumptions about $\mathcal{P}$. Now notice that $m_{\alpha}$ and $m$ are identical under $\alpha.\_$. We conclude that $\alpha.\_$ is $\mathcal{P}$-factorisable. ■

### 6.4.8 Putting it together

Let us consider Milner's CCS-operators except recursion, which is handled in next section. Under the common assumption that only guarded sum is considered, it is shown in [WN95] how these CCS-operators can be expressed by the above constructions (functors). For each operator we have obtained a theorem for the corresponding functor which identifies conditions which guarantee that the functor is a $\mathcal{P}$-operator. Or put differently, for each functor we have meta-theorems providing conditions on $\mathcal{P}$ guaranteeing that $\sim_{\mathcal{P}}$ remains a congruence with respect to the functor (operator).

However, we would like to consider more than one functor at the time. Does there exist choices of $\mathcal{P}$, such that $\mathcal{P}$ satisfies the conditions of all our theorems (including relabelling and prefixing) ?

Choosing $\mathcal{P}$ in $\mathcal{M}$ as the full subcategory induced by words (i.e., fibre-wise as done for $\mathcal{P}_M$ in Sec. 6.4.2), we can show that $\sim_{\mathcal{P}}$ also corresponds to Milner's strong bisimulation. Moreover, it is easy to see that $\mathcal{P}$ satisfies *all* conditions of our theorems, i.e., $\sim_{\mathcal{P}}$ must be a congruence with respect to all the operators (functors). For example, let us just consider the conditions from Theorem 151. They state that when viewing the objects of $\mathcal{P}$ as finite strings, $\mathcal{P}$ in general has to be closed under the operation of taking a subsequence, and possibly renaming the labels. Furthermore, as an immediate consequence we conclude that $\sim_{\mathcal{P}}$ is a congruence with respect to the aforementioned CCS operators.

What about other choices of $\mathcal{P}$? If—similarly to the choice of $\mathcal{P}_H$ in $\mathcal{P}_M$ in Sect. 6.2—we choose $\mathcal{P}$ as the subcategory of the previous choice of $\mathcal{P}$ obtained by only keeping identity morphisms and morphisms whose domains are observations having only one state (the empty word), then $\sim_{\mathcal{P}}$ corresponds to Hoare trace equivalence. This choice of $\mathcal{P}$ also trivially satisfies all conditions required by the theorems. Hence, Hoare trace equivalence is a congruence with respect to the presented constructions (and, again, the aforementioned CCS operators).

Choosing $\mathcal{P}$ as, e.g., the subcategory induced by trees will also satisfy all conditions required by the theorems. Hence $\sim_{\mathcal{P}}$, which is a strictly finer equivalence than Milner's strong bisimulation as hinted in Sect. 5.5, must also be a congruence with respect to the presented constructions.

## 6.5 Recursion

For recursion there is no simple way of defining a functor on $\mathcal{M}$ representing Milner's recursion operator. The reason is that one needs some notion of *process variables* which are to be bound by the recursion operator. Some kind of process term language is necessary, as can be seen both in Milner's work [Mil89] and Winskel and Nielsen's [WN95]. However, without introducing a process algebraic term language it is possible to capture a recursion-like operator in a "faithful" way. The restriction is intuitively that free process variable cannot occur under the scope of a parallel composition operator. Such restrictions have been considered by Taubner [Tau89].

First, identify a set of variables *Var* and extend the objects $(S, i, L, \longrightarrow)$ of $\mathcal{M}$ with a partial function $l$ from $S$ to *Var*. Also, we now allow restarting lts. [3] Furthermore, whenever $l$ is defined on a state $s$, there can be no out-going transitions from $s$ and morphisms are now required to respect the labelling function $l$.

We define $F_X : \mathcal{M} \longrightarrow \mathcal{M}$, which intuitively "binds $X$", on objects as follows. Given $T = (S, i, L, \longrightarrow, l)$, then $F_X(T) = (S', i', \longrightarrow', L, l')$, where

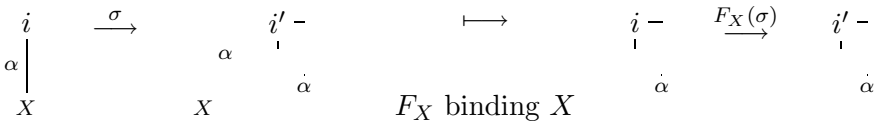
$$S' = \{i\},\ i' = i,\ \longrightarrow' = \emptyset, \text{ and } l' \text{ is totally undefined, when } l(i) = X, \tag{6.15}$$

$$S' = \{s \in S \mid l(s) \neq X\},\ i' = i,\ l' \text{ equals } l \text{ on } S', \text{ when } l(i) \neq X, \text{ where} \tag{6.16}$$

$$\begin{aligned} s \xrightarrow{a}{}' s' \quad \text{if} \quad & s \xrightarrow{a} s' \ \wedge\ l(s') \neq X \\ & \text{or} \\ & \exists\, s''.\, s \xrightarrow{a} s'' \ \wedge\ l(s'') = X \ \wedge\ s' = i \end{aligned} \tag{6.17}$$

Given a morphism $f : T_1 \longrightarrow T_2$. $F_X(f) : F_X(T_1) \longrightarrow F_X(T_2)$ is defined to map $s \in S_1'$ to $f(s)$ if $l_2(f(s)) \neq X$, and $i_2'$ otherwise.

Intuitively, $F_X$ simply redirects all transitions going into $X$-labelled states to the initial state. For example:




$F_X$ has the following desirable property:

**Lemma 166** *For any $X \in Var$, $F_X$ is a functor.*

**Proof.** The proof is routine, hence omitted. ∎

As a special case, let us consider $\mathcal{P}$ as the subcategory of $\mathcal{M}$ corresponding to (6.10) except that final states may now be labelled with variables from *Var*.

**Theorem 167** *For any $X \in Var$, $F_X$ is a $\mathcal{P}$-operator.*

[3] The only implication of this assumption is, that co-product will have to be handled in a way similar to recursion. We could also have considered a recursion operator which "unfolded" the transition systems, and hence stayed within the non-restarting lts.

**Proof.** The first observation is that (6.12) is not going to hold. This is due to the fact that an observation of $F_X(T)$ can correspond to many observations of $T$. However, we can apply the theory from Definition 6.12 on each of these observations individually. So assume $T \xrightarrow{m} T'$ belong to $\mathcal{M}$ and that

$$\begin{array}{ccc} O & \xrightarrow{q} & F_X(T) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle F_X(m)} \\ O' & \xrightarrow[q']{} & F_X(T') \end{array}$$

is a commuting diagram in $\mathcal{M}$. Let us denote $f = (\sigma_f, 1_L)$ and use a similar notation for $q$, $q'$, and $m$. Let $O$ be denoted as

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$$

and $O'$ as

$$s'_0 \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s'_n \xrightarrow{a_{n+1}} \cdots \xrightarrow{a_{n+m}} s'_{n+m} \ .$$

Let $1 \leq j_1 < \cdots < j_r \leq n$ be all indexes such that the is no $a_{j_k}$ transition from $\sigma_q(s_{j_k-1})$ to $\sigma_q(s_{j_k})$ in $T$, where $r \geq 0$. This means that for $1 \leq k \leq r$ there exists a transition $\sigma_q(s_{j_k-1}) \xrightarrow{a_{j_k}} r_k$ in $T$ such that $r_k$ is labelled $X$.

Let $j_0 = 0$ and let $U_1, \ldots, U_r$ be observations in $\mathcal{P}$, where for $1 \leq k \leq r$, $U_k$ is given by

$$(j_{k-1}, \sigma_q(s_{j_{k-1}})) \xrightarrow{a_{j_{k-1}}} \cdots \xrightarrow{a_{j_k-1}} (j_k - 1, \sigma_q(s_{j_k-1})) \xrightarrow{a_{j_k}} (j_k, r_k)$$

with final state labelled by $X$ (labelling set $L$, and initial state $(j_{k-1}, \sigma_q(s_{j_{k-1}}))$). We refer to this procedure as *splitting*.

For $1 \leq k \leq r$, let $U'_k$ be the observation

$$(j_{k-1}, \sigma_f(\sigma_q(s_{j_{k-1}}))) \xrightarrow{a_{j_{k-1}}} \cdots \xrightarrow{a_{j_k-1}} (j_k - 1, \sigma_f(\sigma_q(s_{j_k-1})) \xrightarrow{a_{j_k}} (j_k, \sigma_f(r_k))$$

with labelling set $L$. Again, the final state is labelled by $X$. Notice that if $r > 0$, then $\sigma_{q'}(s'_{j_r}) = i'$ in $T'$.

If there exists no $n < k \leq n + m$ such that there is no $a_k$ transition from $\sigma'_q(s'_{k-1})$ to $\sigma'_q(s'_k)$ in $T'$, then choose $r' = 0$ and $U_{r+r'+1}$ as

$$(j_r, \sigma_q(s_{j_r})) \xrightarrow{a_{j_r+1}} \cdots \xrightarrow{a_n} (n, \sigma_q(s_n))$$

where all states are unlabelled, and $U'_{r+r'+1}$ as

$$(j_r, \sigma_{q'}(s'_{j_r})) \xrightarrow{a_{j_r+1}} \cdots \xrightarrow{a_{n+m}} (n + m, \sigma_{q'}(s'_{n+m}))$$

Else, split

$$s'_{j_r} \xrightarrow{a_{j_r+1}} \cdots \xrightarrow{a_{n+m}} s'_{n+m}$$

obtaining indexes $n \leq j_{r+1} < \cdots < j_{r+r'} \leq n+m$, where $r' > 0$, and observations $U'_{j_{r+1}}, \ldots, U'_{j_{r+r'}}$ with final states labelled with $X$. Let $j_{r+r'+1} = n+m$. Let $U_{r+1}$ be the observation

$$(j_r, \sigma_q(s_{j_r})) \xrightarrow{a_{j_r+1}} \cdots \xrightarrow{a_n} (n, \sigma_q(s_n))$$

with all states unlabelled. For $r+1 < k \leq r+r'+1$ let $U_k$ be the observation consisting of a single unlabelled state $(j_k, i)$. Let $U'_{r+r'+1}$ be the observation

$$(j_{r+r'}, \sigma_{q'}(s'_{j_{r+r'}})) \xrightarrow{a_{j_{r+r'}+1}} \cdots \xrightarrow{a_{n+m}} (n+m, \sigma_{q'}(s'_{n+m}))$$

with all states unlabelled.

For $1 \leq k \leq r+r'+1$ let $V_k$ and $V'_k$ denote the unlabelled versions of $U_k$ and $U'_k$, respectively.

Note that for $1 \leq k \leq r+r'+1$ there exist

- a uniquely determined morphism $f_k : V_k \longrightarrow V'_k$,
- an obvious morphism $q_k : V_k \longrightarrow F_X(T)$, sending a state $(p, s)$ to $s$,
- an obvious morphism $q'_k : V'_k \longrightarrow F_X(T')$,
- a uniquely determined morphism $m_k : U_k \longrightarrow U'_k$,
- an obvious morphism $q_{(k,\#)} : U_k \longrightarrow T$, sending a state $(p, s)$ to $s$,
- an obvious morphism $q'_{(k,\#)} : U'_k \longrightarrow T'$,
- an obvious morphism $q_{(k,\star)} : V_k \longrightarrow F_X(U_k)$, sending a state $(p, s)$ to $s$, and
- an obvious morphism $q'_{(k,\star)} : V'_k \longrightarrow F_X(U'_k)$ .

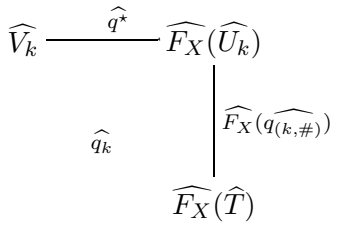
Now for $1 \leq k \leq r+r'+1$

$$\begin{array}{ccc} V_k & \xrightarrow{q_k} & F_X(T) \\ {\scriptstyle f_k}\downarrow & & \downarrow{\scriptstyle F_X(m)} \\ V'_k & \xrightarrow[q'_k]{} & F_X(T') \end{array}$$

commutes. Also, it can be shown that the two diagrams

$$\begin{array}{ccc} V_k & \xrightarrow{q_{(k,\star)}} & F_X(U_k) \\ {\scriptstyle f_k}\downarrow & & \downarrow{\scriptstyle F_X(m_k)} \\ V'_k & \xrightarrow[q'_{(k,\star)}]{} & F_X(U'_k) \end{array} \qquad \begin{array}{ccc} U_k & \xrightarrow{q_{(k,\#)}} & T \\ {\scriptstyle m_k}\downarrow & & \downarrow{\scriptstyle m} \\ U'_k & \xrightarrow[q'_{(k,\#)}]{} & T' \end{array}$$

commute. Denoting these diagrams as morphisms in $\widehat{\mathcal{M}}$ we can show that the diagram




commutes. From the proof of Theorem 142 it follows that there exists morphisms $h_k : V'_k \longrightarrow F_X(T)$, $1 \leq k \leq r + r' + 1$, such that $q_k = h_k \circ f_k$ and $q'_k = F_X(m) \circ h_k$. From these morphisms one can then obtain a morphism $h = (\sigma_h, 1_L) : O' \longrightarrow F_X(T)$ such that $q = h \circ f$ and $q' = F_X(m) \circ h$. To see this, let $\sigma_h$ be the function that maps $s'_j$ to $\sigma_{h_k}((j, s'_j))$, when $j_{k-1} < j \leq j_k$, and to $i$, when $j = 0$. It can now be shown that $h$ indeed satisfies the claimed equalities. ■

## 6.6 Summary

We have examined Joyal, Nielsen, and Winskel's notion of behavioural equivalence, $\mathcal{P}$-bisimilarity [JNW93], with respect to congruence properties. Inspired by [WN95], we observed that endofunctors on $\mathcal{M}$ can be viewed as abstract operators. Staying within the categorical setting, we then identified simple [4] and natural conditions, which ensure that such endofunctors preserve open maps, i.e., that $\mathcal{P}$-bisimilarity is a congruence with respect to the functors. We formalised this as $\mathcal{P}$-factorisability. The main varying parameters were $\mathcal{M}$, $\mathcal{P}$, and the functors.

We then continued by giving a concrete application by fixing $\mathcal{M}$. For a set of endofunctors, we obtained meta-theorems stating conditions on $\mathcal{P}$, which guaranteed that $\mathcal{P}$-bisimilarity would be a congruence with respect the functors.

As for future research, there are many possibilities. Returning to the discussion in the introduction, one could try to merge the two "orthogonal" approaches we mentioned, e.g., try to identify a way of presenting functors by SOS-like rule systems such that one could state conditions about both the rule systems and $\mathcal{P}$, which would guarantee congruence of $\mathcal{P}$-bisimilarity with respect to all functors, whose defining rule systems obeyed a special format.

Another possibility is to continue to work as in Sect. 6.4—other functors may be considered. However, as shown in Chap. 5, other choices of $\mathcal{M}$ make it possible to capture other interesting behavioural equivalences: weak bisimulation or "true concurrency" equivalences. One could look for similar meta-theorems for such choices of $\mathcal{M}$.

In Sect. 6.4 we examined the category $\mathcal{TS}$ of labelled transitions systems. The morphisms of this category simulated transitions in a strong sense, indicated by (6.9). Another behavioural equivalence one might wish to examine is weak bisimulation [Mil89]. Based on our characterisation of weak bisimulation in Sect. 5.3.2 and how the categorical constructions such as products and co-products depended on composing or pairing

[4] We find it a virtue, that the definition of $\mathcal{P}$-factorisability—just as the definition of open maps—doesn't require more than a modest knowledge of category theory.

labels, we propose the following category, in which weak bisimulation could be handled.

**Definition 168** A *weak labelled transition systems* is a tuple $(S, i, L^o, L^i, \longrightarrow)$, where $S$ is the set of states with initial state $i$, $L^o$ and $L^i$ are disjoint sets of labelled, referred to as *observable* and *invisible* labels, respectively, and $\longrightarrow \subseteq S \times L \times S$, where $L = L^o \cup L^i$, is the transition relation. Moreover, we require that all states in $S$ are reachable from the initial state and that the transition systems are non-restarting. □

As in Sect. 6.4.1, we assume the existence of a special element $*$ which is not member of any labelling set. A partial function $\lambda$ between two labelling sets $L$ and $L'$ can then be represented as a total function from $L \cup \{*\}$ to $L' \cup \{*\}$ such that $*$ is mapped to $*$. If $a \in L$ is mapped to $*$, we interpret this as meaning that $\lambda$ is undefined on $a$. Also, given $T = (S, i, L^o, L^i, \longrightarrow)$, we define $\longrightarrow_*$ to be the set $\longrightarrow \cup \{(s, *, s) \mid s \in S\}$. The transitions $(s, *, s)$ are called *idle* transitions. We write $s \stackrel{t}{\Longrightarrow} s'$ if $t = \alpha_1 \cdots \alpha_n \in (L \cup \{*\})^*$ and $s \stackrel{v_1}{\longrightarrow}_* s_1 \stackrel{\alpha_1}{\longrightarrow}_* s'_1 \stackrel{v_2}{\longrightarrow}_* \cdots \stackrel{v_n}{\longrightarrow}_* s_n \stackrel{\alpha_n}{\longrightarrow}_* s'_n \stackrel{v_{n+1}}{\longrightarrow}_* s'$ for some $s_1, \ldots, s'_n \in S$ and $v_1, \ldots, v_{n+1} \in (L^i \cup \{*\})^*$.

**Definition 169** A morphism is a pair $(\sigma, \lambda) : T_1 \longrightarrow T_2$, where $\sigma$ is a total function from $S_1$ to $S_2$, such that

$$\sigma(i_1) \quad = \quad i_2 \tag{6.18}$$

and $\lambda$ is a partial function from $L_1$ to $L_2$ such that

$$\lambda(L_1^i) \quad \subseteq \quad L_2^i \ . \tag{6.19}$$

Moreover, we require

$$s \stackrel{a}{\longrightarrow}_1 s' \quad \Rightarrow \quad \sigma(s) \stackrel{\widehat{\lambda(a)}}{\Longrightarrow}_2 \sigma(s') \ , \tag{6.20}$$

where $\widehat{\ }$: $(L_2 \cup \{*\})^* \longrightarrow (L_2^o)^*$ removes all invisible labels from its argument. □

Defining composition of morphisms as the usual componentwise composition of functions, one obtains the category $\mathcal{WTS}$ of weak labelled transition systems.

The category $\mathcal{TS}$ sits inside $\mathcal{WTS}$. Intuitively, the objects from $\mathcal{TS}$ correspond to objects from $\mathcal{WTS}$ with empty set of invisible labels. Also, weak bisimilarity between two weak labelled transition systems $T_1$ and $T_2$ is defined as the obvious generalisation of Definition 88. Not surprisingly, if one, as done in Sect. 6.4, chooses $\mathcal{M}$ as the union of fibres and $\mathcal{P}$ analogously to Definition 85, $\mathcal{P}$-bisimilarity corresponds to weak bisimilarity.

The category $\mathcal{WTS}$ has product and co-product constructions similar to those in Sect. 6.4.3 and 6.4.4.

**Definition 170** Let $T_0 \times T_1$ denote $(S, i, L^o, L^i, \longrightarrow)$, where

- $S = S_0 \times S_1$, with $i = (i_0, i_1)$ and projections $\rho_0 : S \longrightarrow S_0$, $\rho_1 : S \longrightarrow S_1$ ,
- $L^o = L_0^o \times (L_1 \cup \{*\}) \cup (L_0 \cup \{*\}) \times L_1^o$,
  $L^i = L_0^i \times (L_1^i \cup \{*\}) \cup (L_0^i \cup \{*\}) \times L_1^i$, with projections $\pi_0 : L \hookrightarrow L_0$ and $\pi_1 : L \hookrightarrow L_1$, and
- $s \xrightarrow{a}_* s' \;\Leftrightarrow\; \rho_0(s) \xrightarrow{\pi_0(a)}_{0*} \rho_0(s') \;\wedge\; \rho_1(s) \xrightarrow{\pi_1(a)}_{1*} \rho_1(s')$ .

□

Let $\Pi_0 = (\rho_0, \pi_0) : T_0 \times T_1 \longrightarrow T_0$ and $\Pi_1 = (\rho_1, \pi_1) : T_0 \times T_1 \longrightarrow T_1$. It can be shown that $\Pi_0$ and $\Pi_1$ are well-defined morphisms and that this construction is a product of $T_0$ and $T_1$ in the category $\mathcal{WTS}$. The proof is case-based and routine.

However, constructions similar to those from, e.g., Definition 148 don't seem go through. We are still able to show that, e.g., the operation $T_0 \times \_ : \mathcal{M} \longrightarrow \mathcal{M}$ is a $\mathcal{P}$-operator. The proof is a bit more elaborate than that of Theorem 151.

**Definition 171** Let $T_0 + T_1$ denote $(S, i, L, \longrightarrow)$, where

- $S = (S_0 \times \{i_1\}) \cup (\{i_0\} \times S_1)$, with $i = (i_0, i_1)$ and injections $in_0 : S_0 \longrightarrow S$, $in_1 : S_1 \longrightarrow S$ ,
- $L^o = (L_0^o \times \{*\}) \cup (\{*\} \times L_1^o)$,
  $L^i = (L_0^i \times \{*\}) \cup (\{*\} \cup L_1^i)$, with injections $j_0 : L_0 \longrightarrow L$ and $j_1 : L_1 \longrightarrow L$, and
- $s \xrightarrow{a} s' \;\Leftrightarrow\; \exists v \xrightarrow{b}_0 v'.\, (in_0(v), j_0(b), in_0(v')) = (s, a, s')$ or

$$\exists v \xrightarrow{b}_1 v'.\, (in_1(v), j_1(b), in_1(v')) = (s, a, s')$$

□

Let $I_0 = (in_0, j_0) : T_0 \longrightarrow T_0 + T_1$ and $I_1 = (in_1, j_1) : T_1 \longrightarrow T_0 + T_1$. It can be shown that this construction is a co-product of $T_0$ and $T_1$ in the category $\mathcal{WTS}$.

In general, the operation $T_0 + \_ : \mathcal{M} \longrightarrow \mathcal{M}$ is not a $\mathcal{P}$-operator. If one tries to apply the definition of $\mathcal{P}$-factorisability, a case analysis leads us to the following (simplified) counterexample.

Let us consider the (open) map

$$\cdot \xrightarrow{\;b\;} \cdot \xrightarrow{\;\tau\;} \odot$$

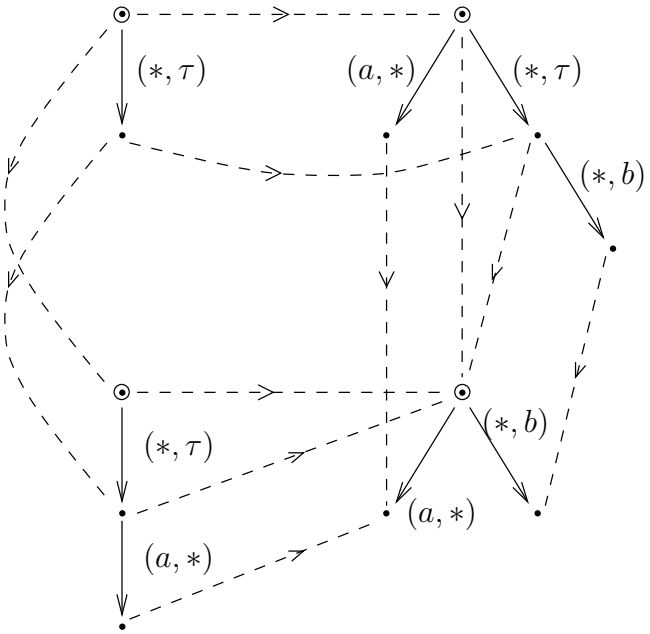
$$\cdot \xrightarrow[\;b\;]{} \odot$$

where $b$ is visible and $\tau$ invisible. Let $T_0$ be $\odot \xrightarrow{a} \cdot$, where $a$ is visible. Taking the operation $T_0 + \_$ with the above open maps produces

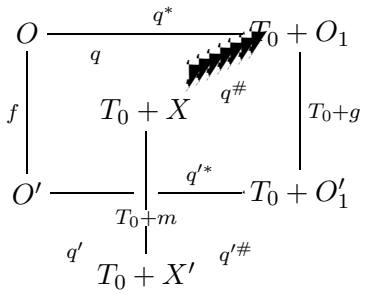
$$\cdot \underline{\quad}^{(*,b)} \cdot \underline{\quad}^{(*,\tau)} \odot \underline{\quad}^{(a,*)} \cdot$$

$$\cdot \overline{\quad}_{(*,b)} \odot \overline{\quad}_{(a,*)} \cdot$$

Now consider the commuting diagram




The objects to the left and the morphism connecting them belong to $\mathcal{P}$. [5] Since the diagram is of the form (6.1) and commuting, it corresponds to a morphism $\widehat{O} \xrightarrow{\widehat{q}} \widehat{T_0+}(\widehat{X})$ in $\widehat{\mathcal{M}}$. Hence, we have to show that there exist an object $\widehat{O_1}$ in $\widehat{\mathcal{P}}$ and morphisms $\widehat{O} \xrightarrow{\widehat{q^*}} \widehat{F}(\widehat{O_1})$ and $\widehat{O_1} \xrightarrow{\widehat{q^\#}} \widehat{X}$ in $\widehat{\mathcal{M}}$ such that the diagram (6.12) commutes. Drawing this diagram in $calM$, yields




[5] Objects with "mixed left and right" labels are used in connection with the product construction.

where $f : O \longrightarrow O'$ and $m : X \longrightarrow X'$ are given by the above diagram. Assume such a commuting diagram exists. Since the upper "triangle" in the diagram commutes, $O_1$ must necessarily be of the form

$$\odot \xrightarrow{\alpha} \cdot \xrightarrow{\beta_1} \cdots \xrightarrow{\beta_n} \cdot \quad ,$$

where $\alpha$ is an invisible label, and $n \geq 0$. Hence, $O'_1$ must also be of the form

$$\odot \xrightarrow{\alpha} \cdot \xrightarrow{\beta_1} \cdots \xrightarrow{\beta_m} \cdot \quad ,$$

where $m \geq n$. Since $q'^*$ is assumed to be a morphism and the above diagram lies in a fibre, $O'$'s initial transition $\odot \xrightarrow{(*,\tau)} \cdot$ must be mapped to the idle transition at $T_0 + O'_1$'s initial state—$O'$'s transition $\cdot \xrightarrow{(a,*)} \cdot$ has to be mapped to the $(a, *)$-labelled transition in $T_0 + O'_1$ originating from $T_0$. However, then

$$\begin{array}{ccc} O & \xrightarrow{\quad q^* \quad} & T_0 + O_1 \\ {\scriptstyle f}\big\downarrow & & \big\downarrow{\scriptstyle T_0+g} \\ O' & \xrightarrow{\quad q'^* \quad} & T_0 + O'_1 \end{array}$$

cannot commute. Abstractly, we see that the conditions for $\mathcal{P}$-factorisability implies, that an initial invisible transition in $T_0 + X$, which resolves the nondeterministic choice '+', must be matched by a similar resolving choice in $T_0 + X'$.

# Part III

# On Set Constraints and Rational Spaces

Chapter 7

# A Complete Gentzen-style Axiomatisation for Set Constraints

Contents

## 7.1 Introduction

We propose a Gentzen-style axiomatisation involving sequents of the form $\Phi \vdash \Psi$, where $\Phi$ and $\Psi$ are finite sets of set constraints. The intended interpretation of the sequent $\Phi \vdash \Psi$ is that if all the constraints in $\Phi$ hold of some model, then at least one of the constraints $\Psi$ holds in that model.

This axiomatisation can be thought of as a deductive system for refuting unsatisfiable systems of mixed positive and negative constraints. Deriving the sequent $\Phi \vdash \Psi$ is tantamount to refuting the mixed system $\Phi \cup \{s \neq t \mid s = t \in \Psi\}$. Systems of the restricted form $\Phi \vdash \bot$ correspond to systems of positive set constraints alone.

We also give an example of how verification of finite state concurrent systems can be performed using set constraints. More specifically, we show how Milner's protocol can be verified for the absence of deadlocks by encoding it in `clp(sc)`, a logic programming language over set constraints introduced by Kozen [Koz94].

In Sect. 7.2.1–7.3, we briefly review the basic definitions and known results we will need regarding set constraints, termset algebras, term automata, and normal forms. In Sect. 7.4 we present our axiomatisation. Then, in Sect. 7.5 we summarise and discuss future work.

## 7.2 Definitions

In this section we give the necessary definitions.

### 7.2.1 Set Expressions and Set Constraints

Let $\Sigma$ be a finite ranked alphabet consisting of symbols $f$, each with an associated arity. Symbols in $\Sigma$ of arity 0, 1, 2, and $n$ are called *nullary*, *unary*, *binary*, and *n-ary*, respectively. Nullary elements are denoted by $a, b, \ldots$ and are called *constants*. The set of elements of $\Sigma$ of arity $n$ is denoted $\Sigma_n$. In the sequel, the use of expressions of the form $f(t_1, \ldots, t_n)$ carries the implicit assumption that $f$ is of arity $n$.

The set of *ground terms* over $\Sigma$ is denoted $T_\Sigma$. It is the least set such that if $t_1, \ldots, t_n \in T_\Sigma$ and $f \in \Sigma_n$, then $f(t_1, \ldots, t_n) \in T_\Sigma$. If $X = \{x, y, \ldots\}$ is a set of variables, then $T_\Sigma(X)$ denotes the set of terms over $\Sigma$ and $X$, considering variables in $X$ as symbols of arity 0.

Let $\mathrm{B} = (\cup,\ \cap,\ \sim,\ 0,\ 1)$ denote the usual signature of Boolean algebra. Let $\Sigma$+B denote the signature consisting of the disjoint union of $\Sigma$ and B. Boolean operators such as $-$ (set difference) and $\oplus$ (symmetric difference) are defined from these as usual. A *set expression* over $X$ is an element of $T_{\Sigma+\mathrm{B}}(X)$. We use $s, t, \ldots$ to denote set expressions. A typical set expression could be:

$$f(g(x \cup y), \sim(g(a) \cap b))$$

where $g$, $f$ are symbols of arity 1 and 2, respectively, $a$, $b$ are constants, and $x, y \in X$. A *Boolean expression* over $X$ is an element of $T_{\mathrm{B}}(X)$.

A *positive set constraint* is a formal inclusion $s \subseteq t$, where $s$ and $t$ are set expressions. For notational convenience we allow equational constraints $s = t$, although inclusions and equations are interdefinable: $s \subseteq t$ is equivalent to $s \cup t = t$, and $s = t$ to $s \oplus t \subseteq 0$. A *negative set constraint* is the negation of a positive set constraint: $s \not\subseteq t$ or $s \neq t$. We use $\varphi, \psi, \ldots$ to denote set constraints and $\Phi, \Psi, \ldots$ to denote finite sets of set constraints.

### 7.2.2 Axioms of Termset Algebra

In [Koz93], the following axiomatisation of the algebra of sets of ground terms was introduced:

$$
\begin{aligned}
f(\ldots, x \cup y, \ldots) &= f(\ldots, x, \ldots) \cup f(\ldots, y, \ldots) && (7.1)\\
f(\ldots, x - y, \ldots) &= f(\ldots, x, \ldots) - f(\ldots, y, \ldots) && (7.2)\\
\bigcup_{f \in \Sigma} f(1, \ldots, 1) &= 1 && (7.3)\\
f(1, \ldots, 1) \cap g(1, \ldots, 1) &= 0,\ f \neq g && (7.4)\\
f(x_1, \ldots, x_n) = 0 &\Rightarrow \bigvee_{i=1}^{n} (x_i = 0) && (7.5)\\
\text{axioms} &\quad \text{of} \quad \text{Boolean algebra} && (7.6)
\end{aligned}
$$

The ellipses in (7.1) and (7.2) indicate that the explicitly given arguments occur in corresponding places, and that the implicit arguments in corresponding places agree. Models of the axioms are called *termset algebras*. The standard interpretation $2^{T_\Sigma}$, where the Boolean operators have their usual set-theoretic interpretations and elements $f \in \Sigma_n$ are interpreted as

$$
\begin{aligned}
f &: (2^{T_\Sigma})^n \to 2^{T_\Sigma}\\
f(A_1, \ldots, A_n) &= \{ f(t_1, \ldots, t_n) \mid t_i \in A_i,\ 1 \leq i \leq n \}\ ,
\end{aligned}
$$

forms a model of these axioms.

Some immediate consequences of these axioms are

$$
\begin{aligned}
f(\ldots, 0, \ldots) &= 0 && (7.7)\\
f(\ldots, \sim x, \ldots) &= f(\ldots, 1, \ldots) - f(\ldots, x, \ldots) && (7.8)\\
f(\ldots, x \oplus y, \ldots) &= f(\ldots, x, \ldots) \oplus f(\ldots, y, \ldots) && (7.9)\\
f(\ldots, x \cap y, \ldots) &= f(\ldots, x, \ldots) \cap f(\ldots, y, \ldots) && (7.10)\\
x \subseteq y &\Rightarrow f(\ldots, x, \ldots) \subseteq f(\ldots, y, \ldots)\ . && (7.11)
\end{aligned}
$$

Also, a *generalised DeMorgan law* can be derived:

$$\begin{aligned} \sim f(x_1,\ldots,x_n) &= \bigcup_{g\neq f} g(1,\ldots,1) \\ &\quad \cup \bigcup_{i=1}^{n} f(\underbrace{1,\ldots,1}_{i-1}, \sim x_i, \underbrace{1,\ldots,1}_{n-i}) \end{aligned} \tag{7.12}$$

The law intuitively says that a ground term *not* having head symbol $f$ and $i^{\text{th}}$ subterm satisfying $x_i$ either has head symbol different from $f$ or has head symbol $f$ but one of its $i^{\text{th}}$ subterms does not satisfy $x_i$. The law is useful for pushing occurrences of the negation operator $\sim$ inward.

### 7.2.3 Term Automata and Models

Following Courcelle [Cou83], we define *(*$\Sigma$*-)terms*.

**Definition 172** Let $\omega$ denote the set of natural numbers and let $\Sigma$ be a finite ranked alphabet. A ($\Sigma$-)*term* is a partial function $t : \omega^* \to \Sigma$ whose domain is nonempty, prefix-closed, and respects arities in the sense that if $t(\gamma)$ is defined then

$$\{i \mid t(\gamma i) \text{ is defined}\} = \{1, 2, \ldots, \text{arity}(t(\gamma))\} .$$

If $\alpha$ is in the domain of $t$, the subterm of $t$ rooted at $\alpha$ is the the term $\lambda\beta.t(\alpha\beta)$. A term is (in)finite if its domain is (in)finite, and is *regular* if it has only finitely many subterms. □

*Example.* The finite term $f(g(a), f(a, g(b)))$ is formally a partial map $t$ with domain $\{\epsilon, 1, 2, 11, 21, 22, 221\}$ such that $t(\epsilon) = t(2) = f$, $t(1) = t(22) = g$, $t(11) = t(21) = a$, and $t(221) = b$. The infinite term $f(a, f(a, f(a, \ldots)))$ is formally a map $s$ whose domain is the infinite set described by the regular expression $2^* + 2^*1$ such that $s(\alpha) = f$ for $\alpha \in 2^*$ and $s(\alpha) = a$ for $\alpha \in 2^*1$. The infinite term $s$ is regular since it has only two subterms, namely $s$ and $a$.

### 7.2.4 Term Automata

It is well known that an infinite regular term can be represented by a finite labelled graph such that the infinite term is obtained by "unwinding" the graph (see [Cou83, Col82]). We use the automata-theoretic formulation introduced in [KPS92] of this idea.

**Definition 173** A *term automaton* over $\Sigma$ is a tuple $\mathcal{M} = (Q, \Sigma, \ell, \delta)$ where:

- $Q$ is a set of *states* (not necessarily finite)
- $\Sigma$ is a ranked alphabet
- $\ell : Q \to \Sigma$ is a *labelling*

- $\delta : Q \times \omega \to Q$ is a partial function such that for all $q \in Q$,

$$\{i \mid \delta(q,i) \text{ is defined}\} \quad = \quad \{1, 2, \ldots, \text{arity}(\ell(q))\} \ .$$

The function $\delta$ extends uniquely to a partial function $\widehat{\delta} : Q \times \omega^* \to Q$ according to the inductive definition

$$\begin{aligned} \widehat{\delta}(q, \epsilon) &= q \\ \widehat{\delta}(q, \gamma i) &= \delta(\widehat{\delta}(q,\gamma), i) \ , \end{aligned}$$

with the understanding that $\delta$ is strict (undefined if one of its arguments is undefined). For each $q \in Q$, the partial function

$$t_q \quad = \quad \lambda\gamma.\ell(\widehat{\delta}(q,\gamma))$$

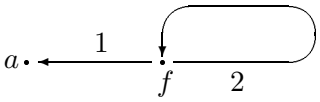
is a $\Sigma$-term in the sense of Definition 172. Notice that $t_p$ may equal $t_q$ even though $p \neq q$.

Every term in the sense of Definition 172 is $t_q$ for some state $q$ of some term automaton. In fact, $t = t_t$ in the syntactic term automaton

$$I_\Sigma \quad = \quad (\{\Sigma\text{-terms}\},\ \Sigma,\ \ell,\ \delta)$$

where $\ell(t) = t(\epsilon)$ and $\delta(t,i) = \lambda\gamma.t(i\gamma)$, $1 \leq i \leq \text{arity}(\ell(t))$. In this sense the notion of term automaton (Definition 173) is a generalisation of the notion of term (Definition 172).

A term is regular if and only if it is $t_q$ for some state $q$ of some finite term automaton [KPS93, Lemma 8]. For example, if $q$ is the state labelled $f$ in the term automaton




then $t_q$ is the infinite regular term $s$ of Example 7.2.3.

A term automaton $\mathcal{M}$ is *closed* if for any $f \in \Sigma_n$ and $q_1, \ldots, q_n \in Q$ there exists a $q \in Q$ such that

$$\ell(q) = f \text{ and } \delta(q,i) = q_i \ , \ 1 \leq i \leq n \ . \tag{7.13}$$

□

A *model* is a closed term automaton $\mathcal{M}$. We refer to the states of $\mathcal{M}$—rather then their associated partial functions $t_q$—as the *terms* of $\mathcal{M}$, and use the notation $\mathbf{t} \in \mathcal{M}$ to indicate $\mathbf{t} \in Q$. A term $\mathbf{t}'$ of $\mathcal{M}$ is a *subterm* of $\mathbf{t}$ at depth $k$ if there exists a $\gamma \in \omega^k$ such that $\widehat{\delta}(\mathbf{t}, \gamma) = \mathbf{t}'$. A term $\mathbf{t}$ of $\mathcal{M}$ is (in)finite if $t_{\mathbf{t}}$ is (in)finite, and said to be labelled by $t'$ if $t_{\mathbf{t}} = t'$. The model is *standard* if the function $q \mapsto t_q : Q \to T_\Sigma$ is a bijection. We denote a standard model by $T_\Sigma$.

*Remark.* For any term automaton $\mathcal{M} = (Q,\ \Sigma,\ \ell,\ \delta)$ there is a closed term automaton $\mathcal{M}' = (Q',\ \Sigma,\ \ell',\ \delta')$ such that $Q \subseteq Q'$, $\ell'$ and $\delta'$ coincide with $\ell$ and $\delta$ on states

from $Q$, and $Q'$ is a minimal set of states—with respect to subset inclusion—with these properties; $\mathcal{M}'$ is said to be a *minimal closure* of $\mathcal{M}$. $\mathcal{M}'$ can be obtained as follows: Let $\mathcal{M}_0 = \mathcal{M}$ and let $\mathcal{M}_{i+1}$ be obtained from $\mathcal{M}_i$ by adding exactly one new term $\mathbf{t}$ to $Q_i$ for every $f \in \Sigma_n$ and $\mathbf{t}_1, \ldots, \mathbf{t}_n \in Q_i$ for which (7.13) doesn't hold. $\ell_{i+1}$ is the extension of $\ell_i$ that maps $\mathbf{t}$ to $f$ and $\delta_{i+1}$ is the extension of $\delta_i$ that maps $(\mathbf{t}, i)$ to $\mathbf{t}_i$, $1 \leq i \leq n$. Define $\mathcal{M}'$ as the $\omega$-limit of these term automata.

### 7.2.5 Term Automata and Set-Theoretic Termset Algebras

Let $\mathcal{M}$ be the term automaton $(Q,\ \Sigma,\ \ell,\ \delta)$. For $f \in \Sigma_n$, define the partial function $R_f^{\mathcal{M}} : Q \to Q^n$ and the set-theoretic function $f^{\mathcal{M}} : (2^Q)^n \to 2^Q$ by

$$R_f^{\mathcal{M}}(q) = \begin{cases} (\delta(q,1), \ldots, \delta(q,n)) , & \text{if } \ell(q) = f \\ \text{undefined} , & \text{otherwise.} \end{cases} \tag{7.14}$$

$$\begin{aligned} f^{\mathcal{M}}(A_1, \ldots, A_n) &= \{q \in Q \mid \ell(q) = f \text{ and } \delta(q,i) \in A_i,\ 1 \leq i \leq n\} \\ &= (R_f^{\mathcal{M}})^{-1}(A_1 \times \cdots \times A_n) . \end{aligned} \tag{7.15}$$

Set expressions are interpreted over $2^Q$, the powerset of $Q$, which forms an algebra of signature $\Sigma$+B, where the Boolean operators have their usual set-theoretic interpretations and elements $f \in \Sigma$ are interpreted as $f^{\mathcal{M}}$. If $\mathcal{M}$ is closed, one can show that this gives a termset algebra. Such an algebra, or a subalgebra of such an algebra, is called a *set-theoretic termset algebra*.

Let $\mathcal{M}$ be a model. A *set valuation* over $\mathcal{M}$ is a map

$$\sigma : X \to 2^Q$$

assigning a subset of terms of $\mathcal{M}$ to each variable in $X$. We can extend any set valuation $\sigma$ uniquely to a ($\Sigma$+B)-homomorphism

$$\sigma : T_{\Sigma+\mathrm{B}}(X) \to 2^Q$$

by induction on the structure of set expressions in the usual way. A set valuation $\sigma$ over $\mathcal{M}$ satisfies the positive set constraint $s \subseteq t$ if $\sigma(s) \subseteq \sigma(t)$, and satisfies the negative set constraint $s \not\subseteq t$ if $\sigma(s) \not\subseteq \sigma(t)$. We write $\sigma \models_{\mathcal{M}} \Phi$ if $\sigma$ satisfies all set constraints in $\Phi$; $\Phi$ is said to be *satisfiable* in $\mathcal{M}$ and $\sigma$ a *solution* to $\Phi$. The set $\Phi$ is satisfiable if it is satisfiable over some model. We write $\Phi \models_{\mathcal{M}} \Psi$ if $\sigma \models_{\mathcal{M}} \Phi$ implies $\sigma \models_{\mathcal{M}} \psi$ for some $\psi \in \Psi$. When no confusion is possible, we suppress the subscript $\mathcal{M}$.

## 7.3 Systems in Normal Form and Solutions

Let $X' \subseteq X$. *Positive (negative) literals* from $X'$ are expressions $x$ ($\sim x$) for $x \in X'$. A maximal conjunction of literals from $X'$ is a conjunction of positive and negative literals from $X'$, where each variable in $X'$ occurs exactly once.

A triple $(t_B,\ \Phi,\ \Delta)$ is a system of set constraints in *normal form* (or just a system in normal form) if there is a finite set $X' \subseteq X$ such that (i) $t_B \in T_{\mathrm{B}}(X')$ is of the form $\cup_{\alpha\in U}\alpha$, for some set $U$ of maximal conjunctions of literals from $X'$, (ii) for each $f \in \Sigma_n$ and $\alpha_1, \ldots, \alpha_n \in U$ there is exactly one set constraint in $\Phi$ of the form $f(\alpha_1, \ldots, \alpha_n) \subseteq \bigcup_{\alpha\in E_{f(\alpha_1,\ldots,\alpha_n)}} \alpha$, where $E_{f(\alpha_1,\ldots,\alpha_n)} \subseteq U$, and (iii) $\Delta$ is a finite set of Boolean expressions $\{\bigcup_{\alpha\in I_1} \alpha, \ldots, \bigcup_{\alpha\in I_m} \alpha\}$, where $I_k \subseteq U$ for $1 \leq k \leq m$. The set $U$ is referred to as the set of atoms[1] specified by $t_B$.

The triple $(t_B,\ \Phi,\ \Delta)$ corresponds to the set of set constraints $\{t_B = 1\} \cup \Phi \cup \{\bigcup_{\alpha\in I_1}\ \alpha \neq 0, \ldots, \bigcup_{\alpha\in I_m}\ \alpha \neq 0\}$ and is said to be (un)satisfiable if the latter is. A set valuation satisfies $(t_B,\ \Phi,\ \Delta)$ if it satisfies the corresponding set constraints. If $\Delta$ is empty, we denote the system in normal form by $(t_B,\ \Phi)$ and call it a system of positive set constraints in normal form (or just a positive system in normal form). Every system of mixed positive and negative set constraints is equivalent to a system in normal form [AKW95].

Each positive system in normal form $(t_B,\ \Phi)$ has an associated hypergraph; the nodes are the elements of $U$ and the hyperedges are specified by the sets $E_{f(\alpha_1,\ldots,\alpha_n)}$. Let $\mathcal{M}$ be a model. A *run* over $\mathcal{M}$ through the hypergraph is a function $\theta : Q \to U$ such that

$$\theta(\mathbf{t}) \in E_{f(\theta(\mathbf{t}_1),\ldots,\theta(\mathbf{t}_n))}\ ,$$

where $\ell(\mathbf{t}) = f \in \Sigma_n$ and $\delta(\mathbf{t}, i) = \mathbf{t}_i$, for $1 \leq i \leq n$. Each subset $U' \subseteq U$ induces a subhypergraph by restricting the nodes and hyperedges to $U'$. The subhypergraph induced by $U'$ is *closed* if for each $f \in \Sigma_n$ and $\alpha_1, \ldots, \alpha_n \in U'$ the set $E_{f(\alpha_1,\ldots,\alpha_n)} \cap U'$ is nonempty. It can be proved that $(t_B,\ \Phi)$ is satisfiable over a standard model if and only if there is a nonempty $U' \subseteq U$ that induces a closed subhypergraph in the hypergraph associated with $(t_B,\ \Phi)$. Intuitively, from a run $\theta$ one can obtain a set valuation $\sigma_\theta$ over a standard model satisfying $(t_B,\ \Phi)$, and—vice versa—from a set valuation $\sigma$ satisfying $(t_B,\ \Phi)$ one can obtain a run $\theta_\sigma$ over a standard model through the hypergraph associated with $(t_B,\ \Phi)$. For details see [AKVW93, Koz93, Koz95].

## 7.4 Completeness and Incompleteness

In this section we give a Gentzen-style axiomatisation for sequents $\Phi \vdash \Psi$, based on the axioms of termset algebra. The intended interpretation of the sequent $\Phi \vdash \Psi$ is that if all the constraints in $\Phi$ hold of some model, then at least one of the constraints $\Psi$ holds in that model. We prove (i) completeness over standard models for satisfiability of positive set constraints (if $\Phi$ is unsatisfiable, then $\Phi$ is refutable, i.e., $\Phi \vdash \bot$ is derivable), (ii) incompleteness over standard models for satisfiability of mixed positive and negative set constraints (i.e., not all true sequents $\Phi \vdash \Psi$ are derivable), and (iii) completeness over nonstandard models.

Any set constraint can be represented as an inclusion $s \subseteq t$, or an equation $u = 0$, or an equation $v = 1$. In the following, any set expression $s$ occurring in a context

[1] The elements of $U$ are the atoms of the free Boolean algebra on generators $X'$ modulo $t_B$.

$$\Phi \vdash \Phi \text{ (ident)} \quad \frac{\Phi \vdash \Psi}{\Phi', \Phi \vdash \Psi, \Psi'} \text{ (weakening)}$$

$$\frac{\Phi, \sim t_i \vdash \Psi, 1 \leq i \leq n}{\Phi, \sim f(t_1, \ldots, t_n) \vdash \Psi} \ (f\text{-intro} \vdash)$$

$$\frac{\Phi, s, t \vdash \Psi}{\Phi, s \cap t \vdash \Psi} \ (\cap\text{-intro} \vdash) \quad \frac{\Phi, s \cap t \vdash \Psi}{\Phi, s, t \vdash \Psi} \ (\cap\text{-elim} \vdash)$$

$$\frac{\Phi, \varphi[t \leftarrow t'], t = t' \vdash \Psi}{\Phi, \varphi, t = t' \vdash \Psi} \ (\text{substitution} \vdash)$$

$$\frac{\Phi, t = t' \vdash \psi[t \leftarrow t'], \Psi}{\Phi, t = t' \vdash \psi, \Psi} \ (\vdash \text{substitution})$$

For $x$ not in $\Phi, t$:

$$\frac{\Phi, x = t \vdash \Psi}{\Phi \vdash \Psi} \ (x\text{-elim} \vdash)$$

For any instance $s = t$ of the termset algebra axioms:

$$\frac{\Phi, s \vdash \Psi}{\Phi, t \vdash \Psi} \ (\text{termset} \vdash) \quad \frac{\Phi \vdash \Psi, s}{\Phi \vdash \Psi, t} \ (\vdash \text{termset})$$

Figure 7.1: The axiomatisation

expecting a set constraint denotes the set constraint $s = 1$. An inclusion $s \subseteq t$ can then be represented as the term $\sim s \cup t$, denoting the set constraint $\sim s \cup t = 1$, and an equation $s = t$ as the term $(\sim s \cup t) \cap (\sim t \cup s)$. A set $\Phi$ denotes the conjunction or disjunction of its elements, depending on whether it occurs on the left or right side of a $\vdash$, respectively. A comma denotes conjunction or disjunction, depending on whether it occurs on the left or right side of a $\vdash$, respectively. We use $\bot$ for the empty disjunction on the right side of $\vdash$; $\bot$ can be read as 0. The rules are shown in Fig. 7.1.
The sequents above and under a bar are referred to as the premises and conclusion of the rule, respectively. $\varphi[t \leftarrow t']$ denotes the substitution of all occurrences of the expression $t$ in $\varphi$ by the expression $t'$.

*Derivation trees* are inductively defined finite trees whose nodes are labelled with sequents $\Phi \vdash \Psi$. A single node labelled with any sequent $\Phi \vdash \Psi$ is a derivation tree, and if there exist derivation trees $\mathcal{T}_1, \ldots, \mathcal{T}_n$ whose roots are labelled with sequents matching the premises of a rule, then the tree whose root is labelled with the conclusion of that rule and has $\mathcal{T}_1, \ldots, \mathcal{T}_n$ as immediate subtrees is itself a derivation tree. A sequent $\Phi \vdash \Psi$ is *derivable from a set* $S$ of sequents if and only if there is a derivation tree all of whose

leaves are labelled by sequents in $S$ and whose root is labelled $\Phi \vdash \Psi$. If $S$ only contains sequents of the form $\Delta \vdash \Delta$ or $\Delta, \sim c \vdash \Gamma$ (corresponding to the rules (ident) and ($f$-intro $\vdash$) for $n = 0$, respectively), then the derivation tree is called a *tableau* and $\Phi \vdash \Psi$ is said to be *derivable*.

*Example.* As an example of how the rules are used, let us consider how $\Phi, \sim t \vdash \Psi$ can be derived from $\Phi, \sim f(\ldots, t, \ldots) \vdash \Psi$, hence it is not necessary to postulate as an axiom the corresponding rule

$$\frac{\Phi, \sim f(\ldots, t, \ldots) \vdash \Psi}{\Phi, \sim t \vdash \Psi} \ (f\text{-elim} \vdash)$$

Assume $f(\ldots, t, \ldots)$ is $f(t_1, \ldots, t_{i-1}, t, t_i, \ldots, t_{n-1})$ and let $x_1, \ldots, x_{n-1}$ be distinct new variables not occurring in $\Phi, f(\ldots, t, \ldots), \Psi$. A derivation (sketch) could be:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\Phi, \sim f(\ldots, t, \ldots) \vdash \Psi}{\Phi, \sim t, \sim f(\ldots, t, \ldots), x_1 = t_1, \ldots, x_{n-1} = t_{n-1} \vdash \Psi}}{\Phi, \sim t, \sim f(x_1, \ldots, x_{i-1}, 0, x_i, \ldots, x_{n-1}), x_1 = t_1, \ldots, x_{n-1} = t_{n-1} \vdash \Psi}}{\Phi, \sim t, 1, x_1 = t_1, \ldots, x_{n-1} = t_{n-1} \vdash \Psi}}{\Phi, \sim t, 1 \vdash \Psi}}{\Phi, \sim t \cap 1 \vdash \Psi}}{\Phi, \sim t \vdash \Psi}$$

The rules applied—bottom-up—are (termset $\vdash$), ($\cap$-intro $\vdash$), ($x$-elim $\vdash$) (several times), (termset $\vdash$) ((7.7) applied to 1), (substitution $\vdash$) (several times, $\sim t$ can be rewritten into $t = 0$, substitute $t$ for 0, then $t_1$ for $x_1$, *etc.*), and finally (weakening).

**Lemma 174** *All rules are sound.*

**Proof.** The proof is straightforward. As an example, assume we are given a model $\mathcal{M}$. Let us consider the ($f$-intro $\vdash$) rule. Assume we have a set valuation $\sigma$ (over $\mathcal{M}$) which satisfies $\Phi, \sim f(t_1, \ldots, t_n)$ and that $\Phi, \sim t_i \models \Psi$ holds for $1 \leq i \leq n$. Since $\sigma(f(t_1, \ldots, t_n)) = \emptyset$, we conclude by $\mathcal{M}$ being closed and the definition of set valuations that $\sigma(t_{i_0}) = \emptyset$ for some $1 \leq i_0 \leq n$. But then $\sigma$ satisfies $\Phi, \sim t_{i_0}$, and by our assumptions $\sigma$ must also satisfy a set constraint in $\Psi$. Hence, $\Phi, \sim f(t_1, \ldots, t_n) \models \Psi$. ∎

The following theorem shows that the deductive system is complete over standard models for satisfiability of positive set constraints.

**Theorem 175** *If a finite set of positive set constraints $\Phi$ is unsatisfiable in any standard model, then $\Phi \vdash \bot$ is derivable.*

**Proof.** We construct a tableau whose root is labelled $\Phi \vdash \bot$ in two stages. In the first stage we show how one can obtain an equivalent finite set of set constraints $\{t_B = 1\} \cup \Phi'$ from $\Phi$, such that $(t_B,\ \Phi')$ is a positive system in normal form. Simultaneously, we show how to derive $\Phi \vdash \bot$ from $t_B, \Phi' \vdash \bot$. This is essentially a formalisation of the normal form algorithm of [AKVW93] in terms of the sequent rules.

Given $\Phi$, replace all occurrences of a subexpression $f(t_1, \ldots, t_n)$ in a set constraint in $\Phi$ by occurrences of a variable $x$ and add the set constraints

$$\begin{aligned} x &= f(y_1, \ldots, y_n) \\ y_i &= t_i,\ 1 \leq i \leq n\ , \end{aligned} \tag{7.16}$$

where $x, y_1, \ldots, y_n$ are new variables. We refer to this as *flattening*. Repeat this until all set constraints are purely Boolean or of the form (7.16). Let $\Delta$ denote the obtained completely flattened set of set constraints. Notice that $\Delta$ is equivalent to $\Phi$. Using ($x$-elim $\vdash$) and (substitution $\vdash$) we can derive $\Phi \vdash \bot$ from $\Delta \vdash \bot$.

Any set constraint of the form (7.16) in $\Delta$ can be replaced by two inclusions

$$f(y_1, \ldots, y_n) \subseteq x \tag{7.17}$$

$$\sim f(y_1, \ldots, y_n) \subseteq \sim x. \tag{7.18}$$

Applying the generalised DeMorgan law (7.12), the inclusion (7.18) is equivalent to

$$\bigcup_{\substack{g \neq f \\ g \in \Sigma}} g(1, \ldots, 1) \cup \bigcup_{i=1}^{n} f(\underbrace{1, \ldots, 1}_{i-1}, \sim y_i, \underbrace{1, \ldots, 1}_{n-i}) \subseteq \sim x$$

and can be replaced by the inclusions

$$\begin{aligned} g(1, \ldots, 1) &\subseteq \sim x \qquad g \neq f \\ f(\underbrace{1, \ldots, 1}_{i-1}, \sim y_i, \underbrace{1, \ldots, 1}_{n-i}) &\subseteq \sim x \qquad 1 \leq i \leq n\ . \end{aligned} \tag{7.19}$$

Let $\Delta'$ denote the current set of set constraints. Since $x = f(y_1, \ldots, y_n)$ may be represented as the term $(\sim x \cup f(y_1, \ldots, y_n)) \cap (\sim f(y_1, \ldots, y_n) \cup x)$ in the sequents, use ($\cap$-intro $\vdash$) and (termset $\vdash$) to obtain the inclusions (7.17) and (7.18). Use (termset $\vdash$) to replace $\sim(\sim f(y_1, \ldots, y_n)) \cup \sim x$ (corresponding to (7.18)) by

$$(\bigcap_{\substack{g \neq f \\ g \in \Sigma}} \sim g(1, \ldots, 1) \cup \sim x) \cap (\bigcap_{i=1}^{n} \sim f(\underbrace{1, \ldots, 1}_{i-1}, \sim y_i, \underbrace{1, \ldots, 1}_{n-i}) \cup \sim x) \tag{7.20}$$

and use ($\cap$-intro $\vdash$) to split this term into terms

$$\begin{aligned} \sim g(1, \ldots, 1) \cup \sim x &\qquad g \neq f \\ \sim f(\underbrace{1, \ldots, 1}_{i-1}, \sim y_i, \underbrace{1, \ldots, 1}_{n-i}) \cup \sim x &\qquad 1 \leq i \leq n\ , \end{aligned}$$

corresponding to the inclusions (7.19). This shows that $\Delta \vdash \bot$ can be derived from $\Delta' \vdash \bot$.

Let $X'$ denote the set of variables occurring in $\Delta'$. At this point, $\Delta'$ only contains either purely Boolean set constraints or set constraints of the form

$$f(x_1, \ldots, x_n) \quad \subseteq \quad x \tag{7.21}$$

where $x_1, \ldots, x_n, x$ are either positive or negative literals from $X'$ or the constant 1. Collect all purely Boolean set constraints and rewrite them, using the laws of Boolean algebra, into one equivalent Boolean set constraint $\bigcup_{\alpha \in U} \alpha = 1$, where $U$ is a set of maximal conjunctions of literals from $X'$. Let $t_B$ denote the left side of this set constraint. The current set of set constraints is now of the form $\{t_B = 1\} \cup \Delta''$, where all set constraints in $\Delta''$ are of the form (7.21). Using ($\cap$-elim $\vdash$) to collect all Boolean terms into one Boolean term and (termset $\vdash$) to replace it by $t_B$, we derive $\Delta' \vdash \bot$ from $t_B, \Delta'' \vdash \bot$.

For $x \in X'$, let $U(x)$ and $U(\sim x)$ denote the set of atoms from $U$ in which $x$ occurs positively and negatively, respectively. Also, let $U(1)$ denote $\bigcup_{\alpha \in U} \alpha$. Using the set constraint $t_B = 1$ we can replace each set constraint of the form (7.21) in $\{t_B = 1\} \cup \Delta''$ by

$$f(\bigcup_{\alpha \in U(x_1)} \alpha, \ldots, \bigcup_{\alpha \in U(x_n)} \alpha) \quad \subseteq \quad \bigcup_{\alpha \in U(x)} \alpha \ , \tag{7.22}$$

which can be rewritten as separate inclusions

$$f(\alpha_1, \ldots, \alpha_n) \quad \subseteq \quad \bigcup_{\alpha \in U(x)} \alpha \ , \ \alpha_i \in U(x_i), \ 1 \leq i \leq n \ . \tag{7.23}$$

For any $f \in \Sigma_n$ and $\alpha_1, \ldots, \alpha_n \in U$, collect all inclusions of the form (7.23) and rewrite them into one

$$f(\alpha_1, \ldots, \alpha_n) \quad \subseteq \quad \bigcup_{\alpha \in E_{f(\alpha_1, \ldots, \alpha_n)}} \alpha \ , \tag{7.24}$$

where $E_{f(\alpha_1, \ldots, \alpha_n)}$ is the intersection of all sets $U(x)$ from the right sides of the collected inclusions. The resulting set of set constraints $\{t_B = 1\} \cup \Phi'$ is equivalent to $\Phi$ and $(t_B, \ \Phi')$ is a positive system in normal form.

The rewriting of inclusion of the form (7.21) into inclusions of the form (7.23) can be done using (termset $\vdash$) to rewrite $x_i$ into $x_i \cap 1$, $1 \leq i \leq n$, (substitution $\vdash$) to substitute $t_B$ for 1, and (termset $\vdash$) and ($\cap$-intro $\vdash$) to obtain the separate inclusions. To obtain the inclusions corresponding to (7.24), use ($\cap$-elim $\vdash$) to collect the appropriate inclusions and (termset $\vdash$) to rewrite them into (7.24). Hence we can derive $t_B, \Delta'' \vdash \bot$ from $t_B, \Phi' \vdash \bot$. This completes the derivation of $\Phi \vdash \bot$ from $t_B, \Phi' \vdash \bot$.

For the second stage we construct a tableau with root $t_B, \Phi' \vdash \bot$, which together with the derivation of $\Phi \vdash \bot$ from $t_B, \Phi' \vdash \bot$ completes the proof.

Since $(t_B,\ \Phi')$ is a positive system in normal form and equivalent to $\Phi$, there must exist an inclusion in $\Phi'$ of the form (7.24) with $E_{f(\alpha'_1,\ldots,\alpha'_n)} = \emptyset$, else the corresponding hypergraph would be closed and $\Phi$ would be satisfiable in a standard model.

If there exists such an inclusion with $n = 0$, then we have found the desired tableau. Otherwise, use ($f$-intro $\vdash$) to derive $t_B, \sim f(\alpha'_1,\ldots,\alpha'_n), \Phi'' \vdash \bot$ from $t_B, \sim \alpha'_i, \Phi'' \vdash \bot, 1 \leq i \leq n$, where $\sim f(\alpha'_1,\ldots,\alpha'_n), \Phi''$ is $\Phi'$. Each of these sequents represents the discarding of one of the atoms in $U$. Consider any $1 \ \leq i \leq n$ and let $U_i$ denote the set $U - \{\alpha'_i\}$ and $t_{B_i}$ denote the conjunction of all elements in $U_i$. Use ($\cap$-elim $\vdash$) and (termset $\vdash$) to derive the sequent $t_B, \sim\alpha'_i, \Phi'' \vdash \bot$ from the sequent $t_{B_i}, \Phi'' \vdash \bot$, which itself can be derived from $t_{B_i}, \Phi''_i \vdash \bot$, where $\Phi''_i$ contains all inclusions of the form (7.24) in which $\alpha'_i$ does not occur on the left of the inclusion and $\alpha'_i$ has been removed from the sets $E_{f(\alpha_1,\ldots,\alpha_n)}$; this can be done using (weakening), (substitution $\vdash$), and (termset $\vdash$). Notice that $(t_{B_i},\ \Phi''_i)$ is a positive system in normal form and is unsatisfiable because $(t_B,\ \Phi')$ is unsatisfiable.

By repeatedly applying the above procedure to all $t_{B_i}, \Phi''_i \vdash \bot$ we conclude that there must exist a tableau deriving $t_B, \Phi' \vdash \bot$ from sequents of the form $\Psi', \sim c \vdash \bot$. ∎

Now suppose we are given a set of mixed positive and negative set constraints $\Phi = \{s_1 = t_1, \ldots, s_n = t_n\} \cup \{s'_1 \neq t'_1, \ldots, s'_m \neq t'_m\}$. Observe that $\Phi$ is unsatisfiable if and only if $\{s_1 = t_1, \ldots, s_n = t_n\} \models \{s'_1 = t'_1, \ldots, s'_m = t'_m\}$. The following theorem shows that the deductive system is incomplete over standard models for satisfiability of mixed positive and negative set constraints.

**Theorem 176** *The axiomatisation is incomplete for systems of mixed positive and negative set constraints over standard models.*

**Proof.** The sequent $x = f(x) \models x = 0$ certainly holds in all standard models. However, $x = f(x) \vdash x = 0$ cannot be derived, since the rules are sound for nonstandard models as well, and if infinite terms are allowed then $x = f(x) \models x = 0$ is no longer valid: in any model containing an infinite term labelled $f(f(f(\ldots)))$, the set of terms labelled $f(f(f(\ldots)))$ is a nontrivial solution to the set constraint $x = f(x)$. ∎

We continue by considering nonstandard models.

**Lemma 177** *A system of set constraints in normal form* $(t_B,\ \Phi,\ \Delta)$*, where* $\Delta = \{\bigcup_{\alpha\in I_1} \alpha, \ldots, \bigcup_{\alpha\in I_m} \alpha\}$*, is satisfiable if and only if there exists a set* $U' \subseteq U$ *such that*

$$\forall f \in \Sigma_n.\, \forall \alpha_1, \ldots, \alpha_n \in U'.\, E_{f(\alpha_1,\ldots,\alpha_n)} \cap U' \neq \emptyset\ , \tag{7.25}$$

$$\forall \alpha \in U'.\, \exists f \in \Sigma_n.\, \exists \alpha_1, \ldots, \alpha_n \in U'.\, \alpha \in E_{f(\alpha_1,\ldots,\alpha_n)}\ ,\ \textit{and} \tag{7.26}$$

$$\forall 1 \leq k \leq m.\, I_k \cap U' \neq \emptyset\ , \tag{7.27}$$

*where* $U$ *are the atoms corresponding to* $t_B$*.*

**Proof.** For the "only if" direction, assume $(t_B,\ \Phi,\ \Delta)$ is satisfiable and let $\sigma : X \to 2^Q$ denote a satisfying set valuation over $\mathcal{M}$. Then $U' = \{\alpha \in U \mid \sigma(\alpha) \neq \emptyset\}$ satisfies the properties (7.25)–(7.27). To see this, notice (i) that (7.25) follows from $\sigma$ being a satisfying set valuation, the definition of $U'$, and axiom (7.5), (ii) that (7.26) follows from $\sigma(1) = \sigma(t_B) = \sigma(\bigcup_{\alpha\in U} \alpha) = \sigma(\bigcup_{\alpha\in U'} \alpha)$ and axiom (7.3), and (iii) that (7.27) follows from $\sigma$ being a satisfying set valuation and the definition of $U'$.

For the "if" direction, assume $U' \subseteq U$ satisfies (7.25)–(7.27). Since $U'$ induces a closed subhypergraph in the hypergraph associated with $(t_B,\ \Phi)$ there exist set valuations over standard models—whose associated runs map terms into $U'$—satisfying $(t_B,\ \Phi)$. Let $U'' \subseteq U'$ be the set of atoms $\alpha$ for which there exists such a set valuation $\sigma$ with $\sigma(\alpha) \neq \emptyset$. Let $l = |U''|$, and let $\sigma_i : X \to 2^{Q_i}$ be set valuations over standard models $\mathcal{M}_i$, $1 \leq i \leq l$, satisfying $(t_B,\ \Phi)$ such that $U'' = \{\alpha \in U' \mid \exists 1 \leq i \leq l.\, \sigma_i(\alpha) \neq \emptyset\}$. Moreover, we may assume that the states of the standard models $\mathcal{M}_1, \ldots, \mathcal{M}_l$ are all mutually disjoint. Note that there exists a model $\mathcal{M}$ whose set of terms (states) contains $\bigcup_{i=1}^{l} Q_i$ and is minimal with respect to subset-inclusion. Moreover, its functions $\ell$ and $\delta$ restricted to $Q_i$ coincide with $\ell_i$ and $\delta_i$; see the remark in Sect. 7.2.3. Let $\theta_i : Q_i \to U''$, $1 \leq i \leq l$, be the runs corresponding to the set valuations $\sigma_i$, i.e., for $\mathbf{t} \in Q_i$, $\theta_i(\mathbf{t})$ is the unique $\alpha$ such that $\mathbf{t} \in \sigma_i(\alpha)$. We define a run $\varrho : Q \to U'$ over $\mathcal{M}$, whose image is $U'$, as the limit of a chain of partial functions from $Q$ to $U'$. Let

$$\varrho_0(\mathbf{t}) = \begin{cases} \theta_i(\mathbf{t})\ , \text{ if } \mathbf{t} \in Q_i,\ 1 \leq i \leq l, \\ \text{undefined, otherwise.} \end{cases}$$

Given $\varrho_j$, define $\varrho_{j+1}$ as follows. For $\mathbf{t} \in Q$,

- if $\varrho_j$ is defined on $\mathbf{t}$, then $\varrho_{j+1}(\mathbf{t})$ is defined as $\varrho_j(\mathbf{t})$
- else, if $\ell(\mathbf{t}) = f \in \Sigma_n$ and $\delta(\mathbf{t}, i) = \mathbf{t}_i$ on which $\varrho_j$ is defined, for $1 \leq i \leq n$, then pick any $\alpha \in E_{f(\varrho_j(\mathbf{t}_1),\ldots,\varrho_j(\mathbf{t}_n))} \cap U'$ and define $\varrho_{j+1}(\mathbf{t}) = \alpha$
- otherwise, $\varrho_{j+1}$ is undefined on $\mathbf{t}$.

Now define $\varrho$ as the limit of $\varrho_0, \varrho_1, \ldots$ It is easy to see that

$$\varrho(\mathbf{t}) \in E_{f(\varrho(\mathbf{t}_1),\ldots,\varrho(\mathbf{t}_n))} \cap U' \tag{7.28}$$

for any $\mathbf{t} \in Q$, where $\ell(\mathbf{t}) = f \in \Sigma_n$ and $\delta(\mathbf{t}, i) = \mathbf{t}_i$, for $1 \leq i \leq n$. Hence, $\varrho$ is a run in the closed subhypergraph induced by $U'$ and the corresponding set valuation $\varsigma_\varrho : X \to Q$ satisfies $(t_B,\ \Phi)$.

If $U'' = U'$ the set valuation satisfies $(t_B,\ \Phi,\ \Delta)$. So assume $U''' = U' \backslash U''$ is nonempty. Pick any $\alpha_{j_1} \in U'''$. We construct a finite tree structure $\mathcal{T}_{j_1}$ whose nodes are labelled by symbols from $\Sigma$. The tree structure is expanded from the root and down as long as certain conditions are met. Also, to each node of the tree we associate an element from $U'$. From $\mathcal{T}_{j_1}$ we obtain a new term $\mathbf{t}_{j_1}$ which will be added to the terms of $\mathcal{M}$. The term $\mathbf{t}_{j_1}$ will then be mapped to $\alpha_{j_1}$ by an extension of $\varsigma_\varrho$.

By (7.26) there exist $f \in \Sigma_n$ and $\alpha_1, \ldots, \alpha_n \in U'$ such that $\alpha_{j_1} \in E_{f(\alpha_1,\ldots,\alpha_n)}$. The root of the tree structure is labelled by $f$ and $\alpha_{j_1}$ is the associated atom.

For all $1 \leq i \leq n$ such that $\varsigma_\varrho(\alpha_i) \neq \emptyset$, pick a $\mathbf{t}_i \in \varsigma_\varrho(\alpha_i)$. Such a $\mathbf{t}_i$ corresponds to the $i^{\text{th}}$ child of the root. The atom associated with the node $\mathbf{t}_i$ is $\varrho(\mathbf{t}_i)$. The nodes $\mathbf{t}_i$ are not expanded further and are referred to as $\mathcal{M}$-nodes.

For all $1 \leq i \leq n$ such that $\varsigma_\varrho(\alpha_i) = \emptyset$, add a new $i^{\text{th}}$ child $\mathsf{n}$ whose associated atom is $\alpha_i$. If there is another node $\mathsf{n}'$ on the path from $\mathsf{n}$ to the root whose associated atom is $\alpha_i$, then label $\mathsf{n}$ by the symbol $f \in \Sigma$ that labels $\mathsf{n}'$. The node $\mathsf{n}$ is not be expanded further; $\mathsf{n}$ is referred to as a *repeat-node* and $\mathsf{n}'$ as its *twin-node*.

Repeat the above procedure for the leaves $\mathsf{n}$ which are neither $\mathcal{M}$-nodes nor repeat-nodes.

Since $U'$ is finite, we obtain a finite tree structure $\mathcal{T}_{j_1}$, all of whose internal nodes are labelled by symbols in $\Sigma$ whose arities respect the branching structure. Moreover, any path from the root either ends at an $\mathcal{M}$-node or in a repeat-node. If there are any repeat nodes, the tree structure corresponds to an infinite regular term.

From $\mathcal{M}$ we obtain a new term automaton $\mathcal{M}'_1$ by adding new nodes to $Q$ for all nodes of $\mathcal{T}_{j_1}$ that are not $\mathcal{M}$-nodes or repeat-nodes and by defining $\ell'_1$ and $\delta'_1$ to be the obvious extensions of $\ell$ and $\delta$ obtained by $\mathcal{T}_{j_1}$, when repeat-nodes are identified with their twin-nodes. Also, $\mathcal{T}_{j_1}$ permits $\varrho$ to be extended to a function $\varrho'_1 : Q'_1 \to U'$ such that the inclusion (7.28) is still valid if $\varrho'_1$ is defined on the occurring terms. Notice that this function is not a run since $\mathcal{M}'_1$ is not a model.

Applying the same procedure for the remaining atoms in $U'''$ we obtain a sequence of term automata $\mathcal{M}, \mathcal{M}'_1, \ldots, \mathcal{M}'_p$ and a corresponding sequence of functions $\varrho, \varrho'_1, \ldots, \varrho'_p$, where $p = |U'''|$, each one extending the previous in the sequence as described above (except for $\mathcal{M}$ and $\varrho$).

Let $\mathcal{M}''$ be a minimal closure of $\mathcal{M}'_p$. We define a run $\theta_{\mathcal{M}''} : Q'' \to U'$ as the limit of a chain of partial functions from $Q''$ to $U'$. Let $\eta_0 = \varrho'_p$ and define $\eta_{i+1}$ from $\eta_i$ as follows. For $\mathbf{t} \in Q''$,

- if $\eta_i$ is defined on $\mathbf{t}$, then $\eta_{i+1}(\mathbf{t})$ is defined as $\eta_i(\mathbf{t})$
- else, if $\ell''(\mathbf{t}) = f \in \Sigma_n$ and $\delta''(\mathbf{t}, i) = \mathbf{t}_i$ on which $\eta_i$ is defined, for $1 \leq i \leq n$, then pick any $\alpha \in E_{f(\eta_i(\mathbf{t}_1),\ldots,\eta_i(\mathbf{t}_n))} \cap U'$ and define $\eta_{i+1}(\mathbf{t}) = \alpha$
- otherwise, $\eta_{i+1}$ is undefined on $\mathbf{t}$.

Define $\theta_{\mathcal{M}''}$ as the limit of $\eta_0, \eta_1, \ldots$ Since $\mathcal{M}''$ is the minimal closure of $\mathcal{M}'_p$ and $\varrho'_p$ is defined on all of $Q'_p$, any $\mathbf{t} \in Q''\backslash Q'_p$ has the property that for some natural number $k$, $\varrho'_p$ is defined on all subterms of $\mathbf{t}$ at depth $k$ or more. This ensures that $\theta_{\mathcal{M}''}$ is defined everywhere on $Q''$. It is easy to see that

$$\theta_{\mathcal{M}''}(f(\mathbf{t})) \in E_{f(\theta_{\mathcal{M}''}(\mathbf{t}_1),\ldots,\theta_{\mathcal{M}''}(\mathbf{t}_n))} \cap U' \tag{7.29}$$

for any $\mathbf{t} \in Q''$, where $\ell''(\mathbf{t}) = f \in \Sigma_n$ and $\delta''(\mathbf{t}, i) = \mathbf{t}_i$, for $1 \leq i \leq n$. Hence, $\theta_{\mathcal{M}''}$ is a run through the hypergraph associated with $(t_B,\ \Phi)$ and the set valuation $\sigma_{\theta_{\mathcal{M}''}}$

corresponding to $\theta_{\mathcal{M}''}$ satisfies $(t_B,\ \Phi,\ \Delta)$, since the image of $\theta_{\mathcal{M}''}$ is $U'$ and (7.27) holds. ■

The last theorem shows that our deductive system is complete for satisfiability of mixed positive and negative set constraints.

**Theorem 178** *If a finite set of mixed positive and negative set constraints*

$$\{s_1 = t_1, \ldots, s_n = t_n\} \cup \{s'_1 \neq t'_1, \ldots, s'_m \neq t'_m\}$$

*is unsatisfiable, then*

$$s_1 = t_1, \ldots, s_n = t_n \vdash s'_1 = t'_1, \ldots, s'_m = t'_m$$

*is derivable.*

**Proof.** Assume $\{s_1 = t_1, \ldots, s_n = t_n\} \cup \{s'_1 \neq t'_1, \ldots, s'_m \neq t'_m\}$ is not satisfiable in any model. We show how to derive

$$s_1 = t_1, \ldots, s_n = t_n \vdash s'_1 = t'_1, \ldots, s'_m = t'_m\ . \tag{7.30}$$

Notice that by repeatedly using ($\vdash$ termset), ( $x$-elim $\vdash$), and ($\vdash$ substitution) we can derive (7.30) from

$$\begin{gathered} s_1 = t_1, \ldots, s_n = t_n, \\ x_1 = (s'_1 \cap \sim t'_1) \cup (\sim s'_1 \cap t'_1), \ldots, x_m = (s'_m \cap \sim t'_m) \cup (\sim s'_m \cap t'_m) \vdash \\ x_1 = 0, \ldots, x_m = 0\ , \end{gathered} \tag{7.31}$$

where $x_1, \ldots, x_m$ are new variables. Now apply the procedure from the proof of Theorem 175 to derive (7.31) from

$$t_B, \Phi \vdash x_1 = 0, \ldots, x_m = 0\ , \tag{7.32}$$

where $(t_B,\ \Phi)$ is a positive system in normal form such that $\{t_B = 1\} \cup \Phi$ is equivalent to $s_1 = t_1, \ldots, s_n = t_n, x_1 = (s'_1 \cap \sim t'_1) \cup (\sim s'_1 \cap t'_1), \ldots, x_m = (s'_m \cap \sim t'_m) \cup (\sim s'_m \cap t'_m)$.

Let $U$ denote the set of atoms specified by $t_B$. Applying ($\vdash$ substitution) and ($\vdash$ termset) we derive (7.32) from

$$t_B, \Phi \vdash \bigcup_{\alpha \in I_1} \alpha = 0, \ldots, \bigcup_{\alpha \in I_m} \alpha = 0\ , \tag{7.33}$$

where $I_j$ is $U(x_j)$, the set of atoms in $U$ in which $x_j$ occurs positively. Notice that $(t_B,\ \Phi,\ \{\bigcup_{\alpha \in I_1} \alpha, \ldots, \bigcup_{\alpha \in I_m} \alpha\})$ is a system in normal form. If the set constraints corresponding to the left of $\vdash$ in (7.33) are not satisfiable, we can derive

$$t_B, \Phi \vdash \bot\ , \tag{7.34}$$

using the technique from Theorem 175, and using (weakening) we can derive (7.33). In fact, in the following, whenever the set constraints to the left of a $\vdash$ in any sequent considered are unsatisfiable, we conclude that the sequent is derivable. So assume $(t_B,\ \Phi)$ is satisfiable. Using (termset $\vdash$), ($\cap$-elim $\vdash$), and (7.3) we derive (7.33) from

$$t_B, \Phi, 1 = \bigcup_{\substack{f\in\Sigma\\ \alpha_1,\ldots,\alpha_n\in U}} f(\alpha_1,\ldots,\alpha_n) \vdash \bigcup_{\alpha\in I_1}\alpha = 0, \ldots, \bigcup_{\alpha\in I_m}\alpha = 0\ , \tag{7.35}$$

which can be derived using (termset $\vdash$), ($\cap$-intro $\vdash$), ($\cap$-elim $\vdash$), and (weakening) from

$$t_B, \Phi, 1 = \bigcup_{\substack{f\in\Sigma\\ \alpha_1,\ldots,\alpha_n\in U}} \bigcup_{\alpha\in E_{f(\alpha_1,\ldots,\alpha_n)}} \alpha \vdash \bigcup_{\alpha\in I_1}\alpha = 0, \ldots, \bigcup_{\alpha\in I_m}\alpha = 0\ , \tag{7.36}$$

which again can be derived from

$$t'_B, \Phi' \vdash \bigcup_{\alpha\in I_1}\alpha = 0, \ldots, \bigcup_{\alpha\in I_m}\alpha = 0\ , \tag{7.37}$$

using (substitution $\vdash$), (termset $\vdash$), and (weakening), where $U' \subseteq U$ is the set

$$\bigcup_{f\in\Sigma}\bigcup_{\alpha_1,\ldots,\alpha_n\in U}\bigcup_{\alpha\in E_{f(\alpha_1,\ldots,\alpha_n)}} \alpha\ ,$$

$t'_B$ is the term $\bigcup_{\alpha\in U'}\alpha$, $\Phi'$ consists of all inclusions of the form

$$f(\alpha_1,\ldots,\alpha_n) \ \subseteq \bigcup_{\alpha\in E'_{f(\alpha_1,\ldots,\alpha_n)}} \alpha\ ,$$

where $f\in\Sigma_n$, $\alpha_1,\ldots,\alpha_n\in U'$, and $E'_{f(\alpha_1,\ldots,\alpha_n)} = E_{f(\alpha_1,\ldots,\alpha_n)} \cap U'$.

Using ($\vdash$ termset) and ($\vdash$ substitution) we can derive (7.37) from

$$t'_B, \Phi' \vdash \bigcup_{\alpha\in I'_1}\alpha = 0, \ldots, \bigcup_{\alpha\in I'_m}\alpha = 0\ , \tag{7.38}$$

where $I'_j = I_j \cap U'$, $1\le j\le m$. Notice $(t'_B,\ \Phi',\ \{\bigcup_{\alpha\in I'_1}\alpha,\ldots,\bigcup_{\alpha\in I'_m}\alpha\})$ is a system in normal form and is satisfiable only if $(t_B,\ \Phi,\ \{\bigcup_{\alpha\in I_1}\alpha,\ldots,\bigcup_{\alpha\in I_m}\alpha\})$ is.

If any $I_j = \emptyset$, $1\le j\le m$, (7.38) is easily seen to be derivable. So assume this is not the case. By repeating the steps from (7.33) to (7.38) we eventually obtain a sequent of the form (7.38), where some $I_j = \emptyset$ or all atoms $\alpha\in U'$ occur in some set $E'_{f(\alpha_1,\ldots,\alpha_n)}$. Now assume the latter is the case. Since $(t'_B,\ \Phi',\ \{\bigcup_{\alpha\in I'_1}\alpha,\ldots,\bigcup_{\alpha\in I'_m}\alpha\})$ is unsatisfiable, we conclude by Lemma 177 that there exist $f\in\Sigma$ and $\alpha_1,\ldots,\alpha_n\in U'$ such that $E'_{f(\alpha_1,\ldots,\alpha_n)} = \emptyset$. So using (termset $\vdash$) and ($f$-intro $\vdash$) we derive (7.38) from

$$t'_B, \Phi'', {\sim}\alpha_i \vdash \bigcup_{\alpha\in I'_1}\alpha = 0, \ldots \bigcup_{\alpha\in I'_m}\alpha = 0\ ,\ 1\le i\le n\ , \tag{7.39}$$

where $\Phi''$ is $\Phi'$ without the inclusion $f(\alpha_1, \ldots, \alpha_n) \subseteq \bigcup_{\alpha \in E'_{f(\alpha_1,\ldots,\alpha_n)}} \alpha$. The sequents in (7.39) whose set constraints to the left of $\vdash$ are unsatisfiable can be derived using the technique from the proof of Theorem 175. The remaining sequents can be derived by repeating steps similar to those used in phase two in the proof of Theorem 175 and those used to derive (7.37) from (7.38) to eliminate the atom $\alpha_i$, and then repeating steps similar to those used to derived (7.33) from (7.39). This procedure eventually terminates, since atoms are being discarded in each iteration. ■

## 7.5 Summary

We have introduced and investigated a deductive system for deriving sequents $\Phi \vdash \Psi$, where $\Phi$ and $\Psi$ are finite sets of set constraints. Using standard and nonstandard models involving set-theoretic termset algebras as introduced in [Koz93], we have shown that the deductive system is (i) complete for restricted sequents of the form $\Phi \vdash \bot$ over standard models, (ii) incomplete for general sequents $\Phi \vdash \Psi$ over standard models, but (iii) complete for general sequents over nonstandard models.

Having chosen term automata as the basis for our models, we naturally get models that allow "multiple copies" of a term $t$, i.e., we may have $t_p = t_q$ for different states $p$ and $q$ of the term automaton. One natural and interesting question that remains is whether the system is complete for general sequents over models that forbid such "multiple copies" but allow infinite terms.

## 7.6 Remarks on model-checking and set constraints

In this section we sketch how model checking may in some cases be performed using the logic programming language `clp(sc)` over set constraints [Koz94]. [2]

To be more specific, we consider deadlock detection in Milner's scheduler [Mil89], a widely used benchmark example.

### 7.6.1 Milner's Scheduler

We assume the reader is a bit familiar with CCS, hence, we will omit some details. The transition graph of a single cycler is depicted below.

Intuitively, a cycler $C_i$ is scheduling a process $P_i$, giving it permission to initiate a task by $t_i$ ($i$ will implicitly range over $1, \ldots, n$). The process $P_i$ signals the completion of the task using $\overline{t_{i,c}}$. The cyclers are synchronising via $c_p$ and $c_r$ (suitably renamed). Only one cycler at the time is allowed to give its process permission to initiate its task. We will call this "granting power".

An $n$-scheduler is obtained by connecting the $n$ cyclers in a ring. This is done using parallel composition, renaming, and restriction.

[2] The language is currently being implemented in C++, hence we have not been able to run performance tests.

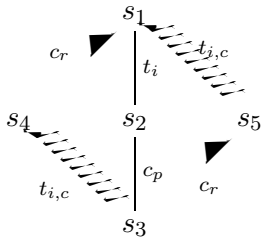



Figure 7.2: Transition graph of the cycler $C_i$.

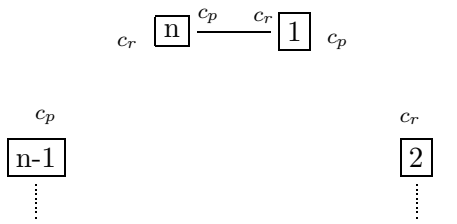



Figure 7.3: An $n$-scheduler.

The figure illustrates that cycler $i$ synchronises its $c_p$-action with the $c_r$-action of cycler $(i+1) \bmod n$, meaning that cycler $i$ can pass, using $c_p$, cycler $(i+1) \bmod n$ granting power.

These are the only synchronisations that are required and allowed among the cyclers. Except for one, all of the $n$ schedulers start in state $s_4$, waiting to receive granting power. The remaining scheduler starts in state $s_1$, indicating that it has the granting power.

The deadlock detection problem is deciding whether or not this system can reach a state from which no further actions can be performed.

### 7.6.2 clp(sc)

clp(sc) is a constraint logic programming language over set constraints [Koz94]. Here we only describe its syntax and semantics informally. For more details, see [Koz94].

Assume $\Pi = \{p, q, r, \ldots\}$ is a finitely ranked alphabet of relation symbols not containing the symbols $=$ or $\subseteq$. An *atomic* formula is an expression of the form $p(u_1, \ldots, u_n)$, where $p \in \Pi_n$ and $u_1, \ldots, u_n$ is an $n$-tuple of set expressions. A *program clause* is either

$$A.$$
$$a :- B_1, \ldots, B_n.$$

where $A$ is an atomic formula and the $B_i$'s are either atomic formulas or positive set constraints. A *program* $\pi$ is a finite set of program clauses.

A *query* is an expression of the form

$$? - B_1, \ldots, B_n.$$

where the $B_i$'s are either atomic formulas or positive set constraints. For example, given the program consisting of

$$\begin{aligned} &sng(a). \\ &sng(f(x_1, \ldots, x_n) : -sng(x_1), \ldots, sng(x_n). \end{aligned}$$

for all constants $a \in \Sigma$ and function symbols $f \in \Sigma_{n>0}$, the query

$$? - sng(x).$$

will succeed if and only if $x$ is a singleton set.

### 7.6.3 The Encoding

In this section we describe how a ring of $n$ schedulers can be encoded in `clp(sc)`. We then give a query that corresponds to the existence of a deadlock in the system.

Our alphabet $\Sigma$ will consist of $\{S_1, \ldots, S_5,\ s_1(\_), \ldots, s_5(\_),\ t_i(\_), t_{i,c}(\_), c_p(\_), c_r(\_)\}$, where we have indicated their arity by specifying how many arguments they take.

The encoding of cycler $C_i$:

$$\begin{aligned} x_i &= S_1 \cup s_1(t_i(y)) \quad , \\ y_i &= S_2 \cup s_2(c_p(z)) \quad , \\ z_i &= S_3 \cup s_3(t_{i,c}(v)) \cup s_3(c_r(w)) \quad , \\ v_i &= S_4 \cup s_4(c_r(x)) \quad , \\ w_i &= S_5 \cup s_5(t_{i,c}(x)) \end{aligned}$$

The intuition is that each variable $x, \ldots, w$ will contain all terms corresponding to finite paths from state $s_1, \ldots, s_5$, respectively. For example, the term $s_2(c_p(s_3(c_r(S_5))))$ corresponds to the path $s_2 \xrightarrow{c_p} s_3 \xrightarrow{c_r} S_5$.

Next we define clauses that expressing that $n$ terms may be interleaved up to the required and allowed synchronisations of the cyclers.

$$sync(x_1, \ldots, x_n) \ :- \ x_1 \cup \cdots \cup x_n \subseteq S_1 \cup \cdots \cup S_5.$$

$$\begin{aligned} sync(s_1(t_1(x_1)), x_2, \ldots, x_n) &:- sync(x_1, x_2, \ldots, x_n). \\ sync(x_1, s_1(t_2(x_2)), \ldots, x_n) &:- sync(x_1, x_2, \ldots, x_n). \\ &\vdots \end{aligned}$$

$$
\begin{array}{rcl}
sync(x_1,\ldots,x_{n-1},s_1(t_n(x_n))) & :- & sync(x_1,x_2,\ldots,x_n).\\
\\
sync(s_3(t_1(x_1)),x_2,\ldots,x_n) & :- & sync(x_1,x_2,\ldots,x_n).\\
sync(x_1,s_3(t_2(x_2)),\ldots,x_n) & :- & sync(x_1,x_2,\ldots,x_n).\\
\vdots & & \\
sync(x_1,\ldots,x_{n-1},s_3(t_n(x_n))) & :- & sync(x_1,x_2,\ldots,x_n).\\
\\
sync(s_2(c_p(x_1)),s_3(t_r(x_2)),x_3,\ldots,x_n) & :- & sync(x_1,x_2,\ldots,x_n).\\
sync(x_1,s_2(c_p(x_2)),s_3(t_r(x_3)),x_4,\ldots,x_n) & :- & sync(x_1,x_2,\ldots,x_n).\\
\vdots & & \\
sync(s_3(t_r(x_1)),x_2,\ldots,x_{n-1},s_2(c_p(x_n))) & :- & sync(x_1,x_2,\ldots,x_n).\\
\\
sync(s_2(c_p(x_1)),s_4(c_r(x_2)),x_3,\ldots,x_n) & :- & sync(x_1,x_2,\ldots,x_n).\\
sync(x_1,s_2(c_p(x_2)),s_4(c_r(x_3)),x_4,\ldots,x_n) & :- & sync(x_1,x_2,\ldots,x_n).\\
\vdots & & \\
sync(s_4(c_r(x_1)),x_2,\ldots,x_{n-1},s_2(c_p(x_n))) & :- & sync(x_1,x_2,\ldots,x_n).
\end{array}
$$

For example, for $n = 2$ the query

$$? - sync(s_1(t_1(s_2(c_p(S_3)))),s_4(c_r(s_1(t_2(S_2))))).$$

will succeed, while the query

$$? - sync(s_4(c_r(s_1(t_i(s_2)))),s_1(t_i(s_2(c_p(s_3))))).$$

will fail. In the first case, cycler 1 permits its process to initiate a task ($t_i$), then by synchronisation passes on the granting power ($c_p$) to process two ($c_r$), which can now permit its process to initiate a task ($t_i$).

The existence of a deadlock corresponds to the existence of execution paths, one for each cycler, that can be interleaved under the required and allowed synchronisations such that the scheduler reaches a state from which no action can be performed. The following clauses capture the situation where two neighbouring cyclers have reached states from which neither can perform further actions under the synchronisation requirements.

$$
\begin{array}{rcl}
nocom(s_1(t_i(x_1)),x_2) & :- & nocom(x_1,x_2).\\
\vdots & & \\
nocom(s_5(t_i(x_1)),x_2) & :- & nocom(x_1,x_2).
\end{array}
$$

$$
\begin{array}{lcl}
nocom(x_1, s_1(t_i(x_2))) & :- & nocom(x_1, x_2). \\
 & \vdots & \\
nocom(x_1, s_5(t_i(x_2))) & :- & nocom(x_1, x_2).
\end{array}
$$

$$
\begin{array}{l}
nocom(s_1, s_4). \\
nocom(s_4, s_4).
\end{array}
$$

### 7.6.4 Solving the Deadlock Detection Problem

With the above clauses and positive set constraints, the query:

$$
\begin{array}{ll}
?- & << cycler_i >>, V_1 \subseteq y_1, V_2 \subseteq x_2, \ldots, V_n \subseteq x_n, sng(V_1), \ldots, sng(V_n), \\
 & sync(V_1, \ldots, V_n), nocom(V_1, V_2), \ldots, nocom(V_{n-1}, V_n), nocom(V_n, V_1).
\end{array}
$$

succeeds if and only if there is a reachable deadlock in the system. The notation $<<$ *cycler* $>>$ denotes the set constraints encoding a cycler.

Whether or not this approach is feasible depends heavily on the implementation of `clp(sc)` and probably on this specific encoding of Milner's scheduler. It is known that the computational complexity of the satisfiability problem is EXPTIME-complete, when $|\Sigma_0| > 0$, $|\Sigma_1| > 1$, and $|\Sigma_{n>2}| = 0$. From Chap. 3, we conclude that an upper bound on deadlock detection problems of the above sort can be expected to be PSPACE. It then seems that the encoding into set constraints gives us an exponential penalty. However, the constraint solving algorithm presented in [Koz94] uses the connection between set constraints and hypergraphs [AKW95, AKVW93]; it involves repeated applications of a common refinement step (forming the conjunction of constraints) followed by a minimisation step. It may turn out that Kozen's proposed efficient unification algorithm, yields acceptable running times for such verification problems. Other approaches based on intermediate minimisation steps, which are faced with similar discouraging worst case running times have turned out to perform quite well in practice [HJJ$^+$95, And95].

# Chapter 8

# On Rational Spaces

**Contents**

## 8.1 Introduction

It has been noticed that many results in the literature on set constraints have a topological flavour. Recently in [Koz95], Kozen defined rational spaces as a family of topological spaces with a regular structure, developed the basic theory, and showed how many results in the literature could be re-derived by general topological principles. By endowing a hypergraph with a topology on its set of states, $D$, and requiring that certain sets of hyperedges are closed in the derived product topology, the set of runs over the hypergraph can be given a topology, yielding a rational space. A category of rational spaces was obtained by defining morphisms as rational maps. These are continuous maps preserving the rational structure. Certain singleton rational subspaces were defined as rational points and shown to play an important role.

We give a Myhill-Nerode-like characterisation of rational points and, based on this characterisation, we give a simple and direct proof that the rational points of a finitary rational space are dense [Koz95]. We show that the rational points in finitary rational spaces in some sense exactly capture the topological structure of the space. We investigate congruences in $\Sigma$-hypergraphs and their interplay with the Myhill-Nerode characterisation. Congruences in rational spaces are strongly related to the notion of bisimulation [Mil89] in models of concurrency.

We also determine the computational complexity of some decision problems related to rational embeddings. In fact, these problems are related to systems of set constraints. In [Koz95], Kozen gives a on-to-one correspondence, up to logical equivalence on one side and so-called rational equivalence preserving $X$ on the other, between (finite) systems of set constraints over variables $X$ and certain (finitary) subspaces of a certain rational space. The given correspondence preserves the partial order of logical entailment between systems of set constraints over $X$ and so-called $X$-preserving rational embeddings between the corresponding subspaces

In Sect. 8.2 we review the basic definitions of $\Sigma$-hypergraphs of rational spaces. In Sect. 8.3, we give our characterisation of rational points and related results. In Sect. 8.4, we define congruences on $\Sigma$-hypergraphs and in Sect. 8.5 we present the complexity results. Finally, in Sect. 8.6 we summarise and discuss future work.

## 8.2 Preliminary Definitions

Let $\Sigma$ be a finite ranked alphabet consisting of symbols $f$, each with an associated arity $n$. Symbols in $\Sigma$ of arity 0, 1, 2, and $n$ are called *nullary*, *unary*, *binary*, and *$n$-ary*, respectively. Nullary elements are denoted by $a, b, \ldots$ and are called *constants*. The set of elements of $\Sigma$ of arity $n$ is denoted $\Sigma_n$. In the sequel, the use of expressions of the form $f(t_1, \ldots, t_n)$ carries the implicit assumption that $f$ is of arity $n$.

The set of *ground terms* over $\Sigma$ is denoted $T_\Sigma$. It is the least set such that if $t_1, \ldots, t_n \in T_\Sigma$ and $f \in \Sigma_n$, then $f(t_1, \ldots, t_n) \in T_\Sigma$. If $X = \{x, y, \ldots\}$ is a set of variables, then $T_\Sigma(X)$ denotes the set of terms over $\Sigma$ and $X$, considering elements in $X$ as symbols of arity 0.

To avoid trivial cases, we assume that $\Sigma$ always contains at least one constant and one symbol of arity greater than zero.

### 8.2.1 Hypergraphs

Let $\Sigma$ be a fixed finite ranked alphabet.

**Definition 179** A $\Sigma$-*hypergraph* is a pair $\mathcal{D} = (D, E)$, where $D$ is a set of states and $E$ is an indexed family of hyperedge relations

$$E_f : D^n \longrightarrow 2^D \ , \qquad n = \text{arity}(f) \ , \tag{8.1}$$

one for every $f \in \Sigma$. □

Hence, for constant $a$, $E_a$ is a subset of $D$, and for unary $g$, $E_g$ is a binary relation on $D$. When no confusion is possible, we may omit $\Sigma$—e.g., we may refer to $\mathcal{D}$ as a hypergraph.

**Definition 180** A hypergraph $(D, E)$ is said to be *entire* if every $E_f(d_1, \ldots, d_n)$ is nonempty, *deterministic* if every $E_f(d_1, \ldots, d_n)$ is a singleton, and *unrestricted* if every $E_f(d_1, \ldots, d_n)$ is $D$. □

**Definition 181** A *run* of a hypergraph $\mathcal{D} = (D, E)$ is a map $\theta : T_\Sigma \longrightarrow D$ such that for all $f(t_1, \ldots, t_n) \in T_\Sigma$,

$$\theta(f(t_1, \ldots, t_n)) \in E_f(\theta(t_1), \ldots, \theta(t_n)) \ . \tag{8.2}$$

The set of runs of $\mathcal{D}$ is denoted $\mathcal{R}(\mathcal{D})$. □

### 8.2.2 Rational Spaces

We recall the basic definition from [Koz95].

**Definition 182** A *topological* $\Sigma$-*hypergraph* is a $\Sigma$-hypergraph $\mathcal{D} = (D, E)$, finite or infinite, endowed with a topology on $D$ whose hyperedges

$$\{(d, d_1, \ldots, d_n) \,|\, d \in E_f(d_1, \ldots, d_n)\} \tag{8.3}$$

are closed in the product topology on $D^{n+1}$. □

**Definition 183** A *space of runs* over $\Sigma$ is the space $\mathcal{R}(\mathcal{D})$ of runs of a topological $\Sigma$-hypergraph $\mathcal{D}$, where the topology on $\mathcal{R}(\mathcal{D})$ is inherited from the product topology on $D^{T_\Sigma}$. The space $\mathcal{R}(\mathcal{D})$ is called *finitary* if $D$ is finite. □

The product topology on $D^{T_\Sigma}$ is the smallest topology such that all projections $\pi_t : D^{T_\Sigma} \longrightarrow D$, mapping $\theta$ to $\theta(t)$, are continuous. Hence, it is generated by the subbasic open sets

$$\{\theta \,|\, \theta(t) \in x\} \ , \quad t \in T_\Sigma \ , \quad x \text{ open in } D \ . \tag{8.4}$$

Recall that open sets in $D^{T_\Sigma}$ are then obtained as arbitrary unions of finite intersections of subbasic open sets. The space $\mathcal{R}(\mathcal{D})$ of runs of $\mathcal{D}$ is a subspace of this space. The topology is thus generated by subbasic open sets (8.4) restricted to $\mathcal{R}(\mathcal{D})$.

**Proposition 184 [Koz95]**
*If $\mathcal{D}$ is finite and discrete, then $\mathcal{R}(\mathcal{D})$ is a complete metric space (all Cauchy sequences converge) under the metric*

$$d(\eta, \eta') \;=\; 2^{-depth(t)} \;, \tag{8.5}$$

*where $t$ is a term of minimal depth on which $\eta$ and $\eta'$ differ, or $0$ if no such term exists.*

**Definition 185** A *rational space* is a space of runs $\mathcal{R}(\mathcal{D})$ such that $\mathcal{D}$ is Hausdorff and compact. □

In [Koz95] it is proved that if $\mathcal{D}$ is Hausdorff and/or compact, then so is the space of runs $\mathcal{R}(\mathcal{D})$. Hence, every rational space is Hausdorff and compact. Also, if a rational space $\mathcal{R}(\mathcal{D})$ is finitary, then $\mathcal{D}$ must be discrete.

**Definition 186** Let $\mathcal{R}(\mathcal{D})$ and $\mathcal{R}(\mathcal{E})$ be rational spaces over $\Sigma$. A *rational map* from $\mathcal{R}(\mathcal{D})$ to $\mathcal{R}(\mathcal{E})$ is a function $\widehat{h} : \theta \mapsto h \circ \theta$ defined by a continuous map $h : \mathcal{D} \longrightarrow \mathcal{E}$ such that

$$h(E_f^{\mathcal{D}}(d_1, \ldots, d_n)) \;\subseteq\; E_f^{\mathcal{E}}(h(d_1), \ldots, h(d_n)) \;. \tag{8.6}$$

A rational map $\widehat{h} : \mathcal{R}(\mathcal{D}) \longrightarrow \mathcal{R}(\mathcal{E})$ is called a *rational embedding* if it is one-to-one, and a *refinement* if it is bijective. □

Notice that $\widehat{h}$ can be one-to-one or bijective even though $h$ is not one-to-one.

If $\mathcal{D} = (D, E)$ and $\mathcal{D}' = (D, E')$ are two hypergraphs over the same set of states $D$, and if $E_f(d_1, \ldots, d_n) \subseteq E'_f(d_1, \ldots, d_n)$ for all $f \in \Sigma$ and $d_1, \ldots, d_n \in D$, then the identity map on $D$ induces an embedding $\mathcal{R}(\mathcal{D}) \longrightarrow \mathcal{R}(\mathcal{D}')$, and $\mathcal{R}(\mathcal{D})$ is called a *narrowing* of $\mathcal{R}(\mathcal{D}')$.

If $\mathcal{D} = (D, E)$ is the induced subhypergraph of $\mathcal{D}' = (D', E')$ on some subset $D \subseteq D'$, i.e., if $E_f(d_1, \ldots, d_n) = E'_f(d_1, \ldots, d_n) \cap D$ for all $f \in \Sigma$ and $d_1, \ldots, d_n \in D$, then the inclusion map $D \longrightarrow D'$ induces an embedding $\mathcal{R}(\mathcal{D}) \longrightarrow \mathcal{R}(\mathcal{D}')$, and $\mathcal{R}(\mathcal{D})$ is called an *induced subspace* of $\mathcal{R}(\mathcal{D}')$.

**Definition 187** A *rational subspace* of a rational space is any embedded image of another rational space. In other words, a subspace $\mathcal{R}$ of a rational space $\mathcal{R}(\mathcal{E})$ is a rational subspace if there exists a rational space $\mathcal{R}(\mathcal{D})$ and a rational embedding $\widehat{h} : \mathcal{R}(\mathcal{D}) \longrightarrow \mathcal{R}(\mathcal{E})$ such that $\mathcal{R} = \widehat{h}(\mathcal{R}(\mathcal{D}))$.

A rational subspace is *entire* if it is the image of a rational space defined on an entire hypergraph. □

**Definition 188** A *rational point* of a rational space is a singleton rational subspace $\mathcal{R}$ that is the embedded image of a finitary rational space $\mathcal{R}(\mathcal{D})$. □

Without loss of generality we may assume that $\mathcal{R}(\mathcal{D})$ is a singleton in Defintion 188.

## 8.3 Myhill-Nerode

In this section we give an alternative characterisation of rational points. Based on this characterisation we give a simple and direct proof that the rational points are dense in any finitary rational space. We then continue by showing that the rational points in finitary rational spaces in some sense exactly capture the topological structure of the spaces, namely, if a rational map $\widehat{h} : \mathcal{R}(\mathcal{D}) \longrightarrow \mathcal{R}(\mathcal{D}')$ between finitary rational spaces induces a bijection between their rational points, then the spaces are homeomorphic.

**Definition 189** Let $*$ be a symbol not in $\Sigma$. A $\Sigma$-*context* is a term in $T_\Sigma(\{*\})$ containing exactly one occurrence of $*$. We denote a context by $C[\ ]$. Given a ground term $t \in T_\Sigma$ and a context $C[\ ]$ we let $C[t]$ denote the ground term in $T_\Sigma$ obtained by replacing $*$ in $C[\ ]$ by $t$. □

**Definition 190** Let $\mathcal{D} = (D, E)$ be a hypergraph, $\theta$ a run in $\mathcal{R}(\mathcal{D})$, and $\approx \subseteq D \times D$ a binary relation. The relation $\approx_\theta \subseteq T_\Sigma \times T_\Sigma$ is then defined by

$$t \approx_\theta t' \quad \text{iff} \quad \forall C[\,].\, \theta(C[t]) \approx \theta(C[t']) \tag{8.7}$$

□

If $\approx$ is an equivalence relation, then so is $\approx_\theta$. In fact, $\approx_\theta$ is a congruence with respect to $\Sigma$, i.e., if $t_1 \approx_\theta t'_1, \ldots, t_n \approx_\theta t'_n$, then $f(t_1, \ldots, t_n) \approx_\theta f(t'_1, \ldots, t'_n)$.

The following theorem presents a Myhill-Nerode-like theorem for rational points.

**Theorem 191** *Let $=$ denote the identity relation on states of $\mathcal{D}$. A run $\theta \in \mathcal{R}(\mathcal{D})$ is a rational point if and only if $=_\theta$ has finite index.*

**Proof.** Assume $\theta$ is a rational point of $\mathcal{R}(\mathcal{D})$. Let $\widehat{h} : \mathcal{R}(\mathcal{D}') \longrightarrow \mathcal{R}(\mathcal{D})$ be a witnessing rational embedding, defined by $h : D' \longrightarrow D$. Let $\mathcal{R}(\mathcal{D}') = \{\eta\}$ and let $\mathcal{D}''$ be the entire subhypergraph of $\mathcal{D}'$ induced by $\eta(T_\Sigma)$. Then, with a slight abuse of notation, we have $\eta \in \mathcal{R}(\mathcal{D}'')$. Since $\mathcal{R}(\mathcal{D}')$ is a singleton $\mathcal{D}''$ must be deterministic and $\mathcal{R}(\mathcal{D}'')$ a singleton. Notice that $h$ induces a rational embedding from $\mathcal{R}(\mathcal{D}'')$ to $\mathcal{R}(\mathcal{D})$, witnessing that $\theta = \widehat{h}(\eta)$ is a rational point. Hence, we may assume without loss of generality that $\mathcal{D}'$ is deterministic. It follows that $t =_\eta t'$ if and only if $\eta(t) = \eta(t')$, and since $\mathcal{D}'$ is finite, $=_\eta$ must have finite index. Next, we conclude that $=_\eta$ refines $=_\theta$, because for any context $C[\ ]$ and terms $t =_\eta t'$

$$\begin{aligned}
\theta(C[t]) &= \widehat{h}(\eta)(C[t]) \\
&= h(\eta(C[t])) \\
&= h(\eta(C[t'])) \\
&= \widehat{h}(\eta)(C[t']) \\
&= \theta(C[t'])\ .
\end{aligned}$$

But then $=_\theta$ has finite index.

Conversely, assume that $=_\theta$ has finite index. Let $\mathcal{D}'$ be the hypergraph whose states $D'$ are the equivalence classes of $=_\theta$ and whose hyperedges are given by

$$E_f(d_1,\ldots,d_n) \;=\; \{[f(t_1,\ldots,t_n)]_{=_\theta} \mid \exists\, t_i \in d_i, 1 \le i \le n\}\ ,$$

where $[t]_{=_\theta}$ denotes the equivalence class of $t$ in $=_\theta$. We claim that $\mathcal{D}'$ is deterministic.

- each $E_f(d_1,\ldots,d_n)$ is nonempty: pick $t_1,\ldots,t_n$ in $d_1,\ldots,d_n$, respectively; then $[f(t_1,\ldots,t_n)]_{=_\theta} \in E_f(d_1,\ldots,d_n)$.
- each $E_f(d_1,\ldots,d_n)$ contains at most one element: if $d, d' \in E_f(d_1,\ldots,d_n)$ and $d \neq d'$, then there exist $t_1, t_1' \in d_1, \ldots, t_n, t_n' \in d_n$ such that $d = [f(t_1,\ldots,t_n)]_{=_\theta}$ and $d' = [f(t_1',\ldots,t_n')]_{=_\theta}$. But the congruence properties of $=_\theta$ imply $f(t_1,\ldots,t_n) =_\theta f(t_1',\ldots,t_n')$, contradicting $d \neq d'$.

It follows that $\mathcal{R}(\mathcal{D}')$ must be a singleton $\{\eta\}$. An inductive argument shows that $\eta(t) = [t]_{=_\theta}$. Let $D'$ be endowed with the discrete topology. Since $D'$ is finite, $D'$ is Hausdorff and compact. Define $h : D' \longrightarrow D$ by $[t]_{=_\theta} \mapsto \theta(t)$. The mapping is well-defined and trivially continuous. For $d \in E_f(d_1,\ldots,d_n)$ assume $d = [f(t_1,\ldots,t_n)]_{=_\theta}$, where $t_1 \in d_1, \ldots, t_n \in d_n$. Then $h(d) = \theta(f(t_1,\ldots,t_n))$, $h(d_1) = \theta(t_1), \ldots, h(d_n) = \theta(t_n)$. Since $\theta(f(t_1,\ldots,t_n)) \in E_f(\theta(t_1),\ldots,\theta(t_n))$ by definition of $\theta$ we conclude $h(E_f(d_1,\ldots,d_n)) \subseteq E_f(h(d_1),\ldots,h(d_n))$, i.e., $\widehat{h} : \mathcal{R}(\mathcal{D}') \longrightarrow \mathcal{R}(\mathcal{D})$ is a rational embedding and $\widehat{h}(\eta)$ is a rational point of $\mathcal{R}(\mathcal{D})$. Since $\theta(t) = h([t]_{=_\theta}) = \widehat{h}(\eta)(t)$, $\theta$ is a rational point of $\mathcal{R}(\mathcal{D})$. ■

We can now give simple proofs of two results from [Koz95].

**Theorem 192** *The rational points of any finitary rational space $\mathcal{R}(\mathcal{D})$ are dense.*

**Proof.** Recall that $\mathcal{R}(\mathcal{D})$ is a complete metric space under the metric (8.5). Let $\theta$ be any point of $\mathcal{R}(\mathcal{D})$, $\mathcal{D} = (D, E)$. We wish to show that there exist rational points arbitrarily close to $\theta$.

Let $\mathcal{D}' = (D', E')$ be a deterministic narrowing of the induced subspace on $D' = \theta(T_\Sigma)$. Then $\mathcal{D}'$ is entire and deterministic. For $f \in \Sigma_n$ and $d_1,\ldots,d_n \in D'$, let $H_f(d_1,\ldots,d_n)$ be the unique element of $E_f'(d_1,\ldots,d_n)$.

For each $k \ge 0$, define inductively

$$\eta(f(t_1,\ldots,t_n)) \;=\; \begin{cases} \theta(f(t_1,\ldots,t_n))\ , & \text{if } \mathit{depth}(f(t_1,\ldots,t_n)) \le k \\ H_f(\eta(t_1),\ldots,\eta(t_n))\ , & \text{otherwise.} \end{cases}$$

Then $\eta \in \mathcal{R}(\mathcal{D})$, since if $\mathit{depth}(f(t_1,\ldots,t_n)) \le k$, then

$$\begin{aligned} \eta(f(t_1,\ldots,t_n)) &= \theta(f(t_1,\ldots,t_n)) \\ &\in E_f(\theta(t_1),\ldots,\theta(t_n)) \\ &= E_f(\eta(t_1),\ldots,\eta(t_n))\ , \end{aligned}$$

and if $depth(f(t_1, \ldots, t_n)) > k$, then

$$\begin{aligned} \eta(f(t_1, \ldots, t_n)) &= H_f(\eta(t_1), \ldots, \eta(t_n)) \\ &\in E'_f(\eta(t_1), \ldots, \eta(t_n)) \\ &\subseteq E_f(\eta(t_1), \ldots, \eta(t_n)) \ . \end{aligned}$$

The point $\eta$ is of distance at most $2^{-k}$ from $\theta$, since it agrees with $\theta$ on all terms of depth at most $k$.

Finally, we show that $\eta$ is a rational point. If $depth(s)$, $depth(t) > k$ and $\eta(s) = \eta(t)$, then for all contexts $C[\ ]$, $\eta(C[s]) = \eta(C[t])$. This can be shown by induction on the structure of $C[\ ]$. Basis, $C[\,] = *$:

$$\eta(*[s]) \ = \ \eta(s) \ = \quad \eta(t) \ = \ \eta(*[t]) \ ,$$

Induction step, $C[\,] = f(t_1, \ldots, t_{i-1}, C'[\,], t_{i+1}, \ldots, t_n)$:

$$\begin{aligned} \eta(C[s]) &= \eta(f(t_1, \ldots, t_{i-1}, C'[s], t_{i+1}, \ldots, t_n)) \\ &= H_f(\eta(t_1), \ldots, \eta(t_{i-1}), \eta(C'[s]), \eta(t_{i+1}), \ldots, \eta(t_n)) \\ &= H_f(\eta(t_1), \ldots, \eta(t_{i-1}), \eta(C'[t]), \eta(t_{i+1}), \ldots, \eta(t_n)) \\ &= \eta(f(t_1, \ldots, t_{i-1}, C'[t], t_{i+1}, \ldots, t_n)) \\ &= \eta(C[t]) \ . \end{aligned}$$

It follows from Theorem 191 that $\eta$ is arational point, since there are only finitely many terms of depth $k$ or less, and these account for finitely many $=_\eta$-classes; and for terms $t$ of depth greater than $k$, the above argument shows that the $=_\eta$-class is determined by $\eta(t)$. Thus $=_\eta$ is of finite index. ■

**Corollary 193** *Every nonempty finitary rational space contains a rational point.*

Next, we continue by showing how rational points of finitary rational spaces capture the topology of the spaces.

### 8.3.1 Rational Points and The Topology

In this section we show that the rational points in finitary rational spaces in some sense exactly capture the topological structure of the space.

Rational maps always preserve rational points. In fact, for finitary rational spaces injectivitiy can be determined by looking at rational points only.

**Lemma 194** *Let $\widehat{h} : \mathcal{R}(\mathcal{D}) \longrightarrow \mathcal{R}(\mathcal{D}')$ be a rational map between finitary rational spaces. If $\gamma, \theta \in \mathcal{R}(\mathcal{D})$ are distinct runs such that $\widehat{h}(\gamma) = \widehat{h}(\theta)$, then there exist distinct rational points $\eta_1, \eta_2 \in \mathcal{R}(\mathcal{D})$ such that $\widehat{h}(\eta_1) = \widehat{h}(\eta_2)$.*

**Proof.** Assume $\gamma$ and $\theta$ are the above mentioned runs. Let

$$P \;=\; \{\, (d_1, d_2) \,|\, \text{there exist infinitely many } t \text{ such that } (\gamma(t), \theta(t)) = (d_1, d_2) \,\}\ .$$

Since $\mathcal{D}$ and $\mathcal{D}'$ are finitary the set $P$ is finite and hence there must exist natural numbers $0 < n$ and $0 < k$ such that

- $\exists t_0.\ \mathit{depth}(t_0) < n \wedge \gamma(t_0) \neq \theta(t_0)$
- $\forall t.\ \mathit{depth}(t) \geq n \;\Rightarrow\; (\gamma(t), \theta(t)) \in P$
- $\forall (d_1, d_2) \in P.\, \exists t.\ n \leq \mathit{depth}(t) < n + k \wedge (\gamma(t), \theta(t)) = (d_1, d_2)$

For each $(d_1, d_2) \in P$ there exists a term $r_{(d_1,d_2)}$ such that $n \leq \mathit{depth}(r_{(d_1,d_2)}) < n+k$ and $(\gamma(r_{(d_1,d_2)}), \theta(r_{(d_1,d_2)})) = (d_1, d_2)$. Let $\mathit{link} : T_\Sigma \longrightarrow T_\Sigma$ be the partial function defined by

- if $\mathit{depth}(t) < n + k$, then $\mathit{link}(t) = t$
- if $\mathit{depth}(t) = n + k$, then $\mathit{link}(t) = r_{(\gamma(t),\theta(t))}$
- if $\mathit{depth}(t) > n + k$, then $\mathit{link}(t)$ is undefined

Let $\eta_1$ and $\eta_2$ be partial functions from $T_\Sigma$ to $D$ defined by

- if $\mathit{depth}(t) \leq n + k$, then $\eta_1(t) = \gamma(t)$, else undefined
- if $\mathit{depth}(t) \leq n + k$, then $\eta_2(t) = \theta(t)$, else undefined

We will simultaneously extend the domain of definition of the functions $\mathit{link}, \eta_1$, and $\eta_2$ by defining them on terms of increasing depth, starting by depth $n + k + 1$. We will maintain the invariant

1. $\mathit{link}, \eta_1$, and $\eta_2$ have same domain of definition, namely all terms up to a certain depth
2. if $t \in \mathit{dom}(\eta_1)$, then
   - 2.1. if $\mathit{depth}(t) \geq n$, then $(\eta_1(t), \eta_2(t)) \in P$
   - 2.2. $\mathit{depth}(t) \geq \mathit{depth}(\mathit{link}(t))$ and if $\mathit{depth}(t) \geq n + k$, then $\mathit{link}(t) = r_{(\eta_1(t),\eta_2(t))}$
   - 2.3. if $\mathit{depth}(f(t_1, \ldots, t_n)) > n + k$, then $\eta_i(f(t_1, \ldots, t_n)) = \eta_i(f(\mathit{link}(t_1), \ldots, \mathit{link}(t_n)))$, for $i = 1, 2$
   - 2.4. $(\eta_1(t), \eta_2(t)) = (\eta_1(\mathit{link}(t)), \eta_2(\mathit{link}(t)))$
   - 2.5. $h(\eta_1(t)) = h(\eta_2(t))$
3. if $f(t_1, \ldots, t_n) \in \mathit{dom}(\eta_1)$, then $\eta_i(f(t_1, \ldots, t_n)) \in E_f(\eta_i(t_1), \ldots, \eta_i(t_n))$, for $i = 1, 2$

Assume $link, \eta_1$, and $\eta_2$ have been defined on exactly all terms of depth less than $m > n+k$, and that the invariant holds. Pick a term $t = f(t_1, \ldots, t_n)$ of depth $m$. Let $t'_1 = link(t_1), \ldots, t'_n = link(t_n)$, and $t' = f(t'_1, \ldots, t'_n)$. Since $depth(t) > n+k$ we conclude $n \leq depth(t') \leq n+k$, by 2.2. Then, $(\eta_1(t'), \eta_2(t')) = (\gamma(t'), \theta(t')) \in P$, by the definitions of $\eta_1$, $\eta_2$, and $n$. Let $link(t) = r_{(\gamma(t'),\theta(t'))}$ and $(\eta_1(t), \eta_2(t)) = (\gamma(t'), \theta(t'))$.

Having defined $link, \eta_1$, and $\eta_2$ for all terms of depth $m$, claim that the invariant is maintained. We need only consider terms $t$ of depth $m$. Clearly, 1. holds. From the above, 2.1., 2.2., and 2.3. follow easily. Next, observe that 2.4. follows from 2.1., 2.2., the fact that $(\eta_1(r), \eta_2(r)) = (\gamma(r), \theta(r))$ for terms $r$ with $depth(r) < n+k$, and that $(\gamma(r_{(d_1,d_2)}), \theta(r_{(d_1,d_2)})) = (d_1, d_2)$. 2.5. follows from 2.1. and $\widehat{h}(\gamma) = \widehat{h}(\theta)$. To see that 3. holds, let $t = f(t_1, \ldots, t_n)$ and $t' = f(t'_1, \ldots, t'_n)$ as above. Then

$$\begin{aligned}
\eta_1(t) = \eta_1(t') &\in E_f(\eta_1(t'_1), \ldots, \eta_1(t'_n)) \ , \quad \text{(3. used on } t') \\
&= E_f(\eta_1(link(t_1)), \ldots, \eta_1(link(t_n))) \\
&= E_f(\eta_1(t_1), \ldots, \eta_1(t_n)) \ , \quad \text{(2.4. used on } t_1, \ldots, t_n)
\end{aligned}$$

A similar argument holds for $\eta_2(t)$. Let $\eta_1$ and $\eta_2$ denote the total functions obtained by considering the limit to infinity of the above construction. By 3., $\eta_1, \eta_2 \in \mathcal{R}(\mathcal{D})$ and by 2.5., $\widehat{h}(\eta_1) = \widehat{h}(\eta_2)$. Also, $\eta_1 \neq \eta_2$, since they differ on the term $t_0$.

We conclude by showing that $\eta_1$ and $\eta_2$ are rational points. Let the *height* of a context $C[\ ] \in T_\Sigma(\{*\})$ be the depth of the element $*$. By induction in the height of the context $C[\ ]$, we show that if $t$ and $t'$ are terms of depth $n+k$ or greater, then $(\eta_1(t), \eta_2(t)) = (\eta_1(t'), \eta_2(t'))$ implies $(\eta_1(C[t]), \eta_2(C[t])) = (\eta_1(C[t']), \eta_2(C[t']))$.

For the context of height 0 there is nothing to prove, so assume $C[\ ]$ is a context of height $l > 0$. Then there exist contexts $C'[\ ]$ and $C''[\ ]$ of height 1 and $l-1$, respectively, such that $C[t] = C'[C''[t]]$ and $C[t'] = C'[C''[t']]$. By induction $(\eta_1(C''[t]), \eta_2(C''[t])) = (\eta_1(C''[t']), \eta_2(C''[t']))$ and by 2.2. $link(C''[t]) = link(C''[t'])$. Without loss of generality, let $C'[\ ] = g(t_1, \ldots, t_{i-1}, *, t_{i+1}, \ldots, t_m)$, where $g \in \Sigma_{m>0}$, $1 \leq i \leq m$, and $t_1, \ldots, t_{i-1}$, $t_{i+1}, \ldots, t_m$ are arbitrary terms. Then, by 2.3.

$$\begin{aligned}
\eta_1(C[t]) &= \eta_1(g(t_1, \ldots, t_{i-1}, C''[t], t_{i+1}, \ldots, t_m)) \\
&= \eta_1(g(link(t_1), \ldots, link(t_{i-1}), link(C''[t]), link(t_{i+1}), \ldots, link(t_m))) \\
&= \eta_1(g(link(t_1), \ldots, link(t_{i-1}), link(C''[t']), link(t_{i+1}), \ldots, link(t_m))) \\
&= \eta_1(g(t_1, \ldots, t_{i-1}, C''[t'], t_{i+1}, \ldots, t_m)) \\
&= \eta_1(C[t'])
\end{aligned}$$

Similarly, $\eta_2(C[t]) = \eta_2(C[t'])$. Hence, $t =_{\eta_1} t'$ and $t =_{\eta_2} t'$. Since there are only finitely many terms of depth less than $n+k$ and only finitely many elements in $P$, we conclude that $=_{\eta_1}$ and $=_{\eta_2}$ have finite index. By Theorem 191, $\eta_1$ and $\eta_2$ are rational points. ■

The following theorem is the main result of this section.

**Theorem 195** *Let $\widehat{h} : \mathcal{R}(\mathcal{D}) \longrightarrow \mathcal{R}(\mathcal{D}')$ be a rational map between finitary rational spaces. Assume $\widehat{h}$ is a bijection between the rational points of the spaces. Then $\widehat{h}$ is a homeomorphism.*

**Proof.** By Lemma 194 and the fact that $\widehat{h}(\theta)$ is a rational point if $\theta$ is rational point, we conclude that $\widehat{h}$ must be one-to-one.

Recall that any finitary rational space is a complete metric space under the metric

$$d(\eta, \eta') \;=\; 2^{-depth(t)} \;,$$

where $t$ is a term of minimal depth on which $\eta$ and $\eta'$ differ, or 0 if no such term exists. Let $\theta \in \mathcal{R}(\mathcal{D}')$. By Theorem 192 there exist a sequence of rational points $\theta_1, \theta_2, \ldots \in \mathcal{R}(\mathcal{D}')$ converging to $\theta$ such that $d(\theta, \theta_i) < 2^{-i}$, for $i = 1, 2, \ldots$. Let $\{\eta_i\} = \widehat{h}^{-1}(\theta_i)$, for $i = 1, 2, \ldots$. Since $D$ is finite, there must be infinitely many $\eta_i$'s that agree on the finitely many terms of depth 1. Let $\eta_{i_{(1,1)}}, \eta_{i_{(1,2)}}, \ldots$ be such an infinite subsequence of $\eta_1, \eta_2, \ldots$. Let $\gamma_1 = \eta_{i_{(1,1)}}$. By a similar argument, there must be an infinite subsequence $\eta_{i_{(2,1)}}, \eta_{i_{(2,2)}}, \ldots$ of $\eta_{i_{(1,2)}}, \eta_{i_{(1,3)}}, \ldots$ that agree on all terms of depth at most 2. Let $\gamma_2 = \eta_{i_{(2,1)}}$. Repeating this procedure we obtain the infinite subsequence $\gamma_1, \gamma_2, \ldots$ of $\eta_1, \eta_2, \ldots$. This sequence is a Cauchy sequence. Let $\gamma$ denote the run this sequence converges to. Notice $\gamma_1, \gamma_2, \ldots$ is mapped under $\widehat{h}$ to the infinite subsequence $\theta_{i_{(1,1)}}, \theta_{i_{(2,1)}}, \ldots$ of $\theta_1, \theta_2, \ldots$, which also converges to $\theta$. Since $\widehat{h}$ is continuous $\gamma$ must be mapped to $\theta$. We conclude that $\widehat{h}$ is onto.

Since any rational space is compact and Hausdorff and $\widehat{h}$ is bijective we conclude that $\widehat{h}$ is a homeomorphism. ■

*Remark.* Notice that in general, if $f : S_1 \longrightarrow S_2$ is a continuous function between compact, complete metric topological spaces, such that $f$ is a bijection between a dense subset of $S_1$ and a dense subset of $S_2$, then $f$ may not even be a bijection, and hence, $S_1$ and $S_2$ not homeomorphic. For example, consider the interval $[0; 1]$ and the circle $C$ obtained by "gluing" the endpoints of $[0; 1]$ together. Their topology is given by the usual Euclidean metric. The obvious mapping $f$ from $[0; 1]$ to $C$ defined by mapping $0 \leq x < 1$ to $x$ in $C$ and 1 to "0" in $C$ is continuous. Moreover, the rational points in $[0; 1[$ are dense in $[0; 1]$ and their image under $f$ is dense in $C$. The function $f$ is a bijection between these dense subsets, but not between $[0; 1]$ and $C$.

**Proposition 196** *Let $\widehat{h} : \mathcal{R}(\mathcal{D}) \longrightarrow \mathcal{R}(\mathcal{D}')$ be a rational map between finitary rational spaces. If $\widehat{h}$ is not onto, then there exists a rational point $\theta \in \mathcal{R}(\mathcal{D}')$ not in the image of $\widehat{h}$.*

**Proof.** The proof is similar to that of Theorem 195. ■

## 8.4 Congruences

In this section we define congruences on hypergraphs. The definition has strong resemblance to Milner's strong bisimulation. We then investigate the relationship to special

rational maps, so-called full homomorphisms, and to the Myhill-Nerode characterisation from the previous section.

**Definition 197** Let $\mathcal{D} = (D, E)$ be a hypergraph. A $\mathcal{D}$-*bisimulation* is a reflexive, symmetric relation $\approx \subseteq D \times D$ such that whenever $d \approx d'$, then

$$\forall\, a \in \Sigma_0.\, d \in E_a \;\Leftrightarrow\; d' \in E_a \tag{8.8}$$

$$\begin{aligned} \forall\, f \in \Sigma_{n>0}.\, \forall 1 \leq i \leq n.\, \forall d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_n. \\ \forall\, d'' \in E_f(d_1, \ldots, d_{i-1}, d, d_{i+1}, \ldots, d_n). \\ \exists\, d''' \in E_f(d_1, \ldots, d_{i-1}, d', d_{i+1}, \ldots, d_n).\, d'' \approx d''' \;. \end{aligned} \tag{8.9}$$

A $\mathcal{D}$-*congruence* is a $\mathcal{D}$-bisimulation that is an equivalence relation. □

The identity relation on $D$ is a $\mathcal{D}$-congruence and the largest $\mathcal{D}$-congruence is given by

$$\bigcup \{ \approx \mid \approx \text{ is a } \mathcal{D}\text{-congruence } \} \;.$$

Notice that if $\Sigma$ only contains constant and unary symbols, then the constants can be seen as state labels while the unary symbols can be seen as edge labels. Then the largest $\mathcal{D}$-congruence, $\sim$, has the following relation to Milner's strong bisimulation [Mil89]: $\sim$ is the largest strong bisimulation with the additional property that $d \sim d'$ if and only if $d$ and $d'$ are labelled identically (with respect to the constants). In general, if $\theta \in \mathcal{R}(\mathcal{D})$, $a \in \Sigma_0$, $d \sim d'$, and $\theta(a) = d$, then there exists $\theta' \in \mathcal{R}(\mathcal{D})$ such that $\theta(a) = d'$ and $\theta(b) = \theta'(b)$, for $a \neq b \in \Sigma_0$.

**Definition 198** Let $\mathcal{D} = (D, E)$ be a hypergraph and $\approx$ be a $\mathcal{D}$-congruence. $\mathcal{D}/\!\approx$ is the hypergraph $(D/\!\approx, E/\!\approx)$ given by

$$D/\!\approx \;=\; \{[d]_\approx \mid d \in D\} \tag{8.10}$$

$$E/\!\approx_f ([d_1]_\approx, \ldots, [d_n]_\approx) = \{[d]_\approx \mid d \in E_f(d_1, \ldots, d_n)\} \;. \tag{8.11}$$

□

Notice that the hyperedge relations are well-defined because $d_1 \approx d_1', \ldots, d_n \approx d_n'$ implies

$$\{[d]_\approx \mid d \in E_f(d_1, \ldots, d_n)\} \;=\; \{[d]_\approx \mid d \in E_f(d_1', \ldots, d_n')\} \;.$$

If $\sim$ is the largest $\mathcal{D}$-congruence and $\mathcal{D}'$ is $\mathcal{D}/\!\sim$, then it can be shown that the only $\mathcal{D}'$-congruence is the identity relation.

**Definition 199** Given $\Sigma$-hypergraphs $\mathcal{D}$ and $\mathcal{D}'$. A mapping $h : D \longrightarrow D'$ is a *homomorphism from* $\mathcal{D}$ if

$$\forall a \in \Sigma_0.\, h^{-1}(E'_a) = E_a \tag{8.12}$$

$$\forall f \in \Sigma_f.\, \forall d_1, \ldots, d_n \in D.\, h(E_f(d_1, \ldots, d_n)) = E'_f(h(d_1), \ldots, h(d_n)) \tag{8.13}$$

The homomorphism $h$ is *full*, if $h(D) = D'$. Two full homomorphisms $h_1 : D \longrightarrow D_1$ and $h_2 : D \longrightarrow D_2$ are *equivalent* if $\mathcal{D}_1$ and $\mathcal{D}_2$ are isomorphic under some $f : D_1 \longrightarrow D_2$ such that $h_2 = f \circ h_1$. □

**Proposition 200** *Given $\mathcal{D}$. The $\mathcal{D}$-congruences are in one-to-one correspondence with the full homomorphisms, up to equivalence, from $\mathcal{D}$.*

**Proof.** Given a $\mathcal{D}$-congruence $\approx$. Define $[\,]_\approx : D \longrightarrow D/\!\approx$ as the mapping $d \mapsto [d]_\approx$. By Definition 197 and 198 it follows easily that $[\,]_\approx$ is a full homomorphism.

Conversely, given $\mathcal{D}'$ and a homomorphism $h : D \longrightarrow D'$. For $d, d' \in D$ define $d \approx_h d'$ if $h(d) = h(d')$. We claim $\approx_h$ is a $\mathcal{D}$-congruence. Clearly, $\approx_h$ is an equivalence relation. Also, if $d \in E_a$ for some $a \in \Sigma_0$, then $h(d) = h(d') \in E'_a$. Hence, by (8.12), $d' \in E_a$. If $f \in \Sigma_{n>0}$, $1 \leq i \leq n$, $d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_n \in D$, and $d'' \in E_f(d_1, \ldots, d_{i-1}, d, d_{i+1}, \ldots, d_n)$, then

$$\begin{aligned} h(d'') &\in h(E_f(d_1, \ldots, d_{i-1}, d, d_{i+1}, \ldots, d_n)) \\ &= E'_f(h(d_1), \ldots, h(d_{i-1}), h(d), h(d_{i+1}), \ldots, h(d_n)) \\ &= E'_f(h(d_1), \ldots, h(d_{i-1}), h(d'), h(d_{i+1}), \ldots, h(d_n)) \\ &= h(E_f(d_1, \ldots, d_{i-1}, d', d_{i+1}, \ldots, d_n))\ . \end{aligned}$$

So there must exist a $d''' \in E_f(d_1, \ldots, d_{i-1}, d', d_{i+1}, \ldots, d_n)$ such that $h(d'') = h(d''')$, i.e., $d'' \approx_h d'''$. Hence, $\approx_h$ is a $\mathcal{D}$-congruence.

The mapping that maps a $\mathcal{D}$-congruence $\approx$ to the full homomorphism $[\,] : D \longrightarrow D/\!\approx$ is clearly one-to-one. Conversely, given any full homomorphism $h : D \longrightarrow D'$, $\mathcal{D}/\!\approx_h$ is isomorphic to $\mathcal{D}'$ under the mapping $[d]_{\approx_h} \mapsto h(d)$. ■

Given a run $\theta \in \mathcal{R}(\mathcal{D}/\!\approx)$, inductively in $t$ define $\eta_\theta$ such that $\eta_\theta(t) \in \theta(t)$ as follows: assume $t = f(t_1, \ldots, t_n)$ and that $\eta_\theta(t_1), \ldots, \eta_\theta(t_n)$ have been defined. Since $\theta$ is a run, $\theta(t) \in E/\!\approx_f (\theta(t_1), \ldots, \theta(t_n))$. By the definition of $\mathcal{D}/\!\approx$, $\theta(t) = [d]_\approx$, where $d \in E_f(d_1, \ldots, d_n)$ and $d_1 \in \theta(t_1), \ldots, d_n \in \theta(t_n)$. Inductively we may assume that $\eta_\theta(t_1) \in \theta(t_1), \ldots, \eta_\theta(t_n) \in \theta(t_n)$, so $\eta_\theta(t_1) \approx d_1, \ldots, \eta_\theta(t_n) \approx d_n$. Since $\approx$ is a $\mathcal{D}$-congruence we conclude that there exists a $d' \in E_f(\eta_\theta(t_1), \ldots, \eta_\theta(t_n))$ such that $d \approx d'$. Define $\eta_\theta(t) = d'$. Then $\eta_\theta(t) \in \theta(t)$. It is easy to see that $\eta_\theta$ is indeed a run of $\mathcal{D}$. We refer to $\eta_\theta$ as a run *extracted* from $\theta$. Notice that if $\eta_\theta$ is extracted from $\theta$, then the mapping $t \mapsto [\eta_\theta(t)]_\approx$ equals $\theta$. Also,

$$\begin{array}{lll} t \approx_{\eta_\theta} t' & \text{iff} & \forall C[\,].\, \eta_\theta(C[t]) \approx \eta_\theta(C[t']) \\ & \text{iff} & \forall C[\,].\, \theta(C[t]) = \theta(C[t'])\ , \quad (\eta_\theta(r) \in \theta(r) \text{ for any } r) \\ & \text{iff} & t =_\theta t'\ , \end{array}$$

i.e., $\approx_{\eta_\theta} = =_\theta$. Conversely, if $\eta \in \mathcal{R}(\mathcal{D})$, then the mapping $\theta_\eta : T_\Sigma \longrightarrow D/\!\approx$ given by $t \mapsto [\eta(t)]_\approx$ is a run of $\mathcal{R}(\mathcal{D}/\!\approx)$.

The construction of $\approx_\theta$ from Definition 190 induces an equivalence relation on $\mathcal{R}(\mathcal{D})$ as follows. It will be notationally convenient to "overload" the symbol $\approx$.

**Definition 201** Let $\mathcal{D} = (D, E)$ be a hypergraph and $\approx \subseteq D \times D$ be a relation. For runs $\eta, \theta \in \mathcal{R}(\mathcal{D})$ define

$$\eta \approx \theta \ \text{ iff } \ \approx_\eta = \approx_\theta \ . \tag{8.14}$$

□

**Theorem 202** *Let $\mathcal{D} = (D, E)$ be a hypergraph. Let $\sim$ be a $\mathcal{D}$-congruence. Then the mapping from $\mathcal{R}(\mathcal{D}/\!\sim)/\!= $ to $\mathcal{R}(\mathcal{D})/\!\sim$ defined by $[\theta]_= \mapsto [\eta_\theta]_\sim$, where $\eta_\theta$ is a run extracted from $\theta$, is well-defined. Moreover, it is a bijection.*

**Proof.** To see that the described mapping is well-defined assume that $\eta, \gamma$ are two runs extracted from $\theta_1$ and $\theta_2$, respectively, where $\theta_1, \theta_2 \in [\theta]_= \in \mathcal{R}(\mathcal{D}/\!\sim)/\!=$. We show that they are mapped to the same element of $\mathcal{R}(\mathcal{D})/\!\sim$, i.e., that $[\eta]_\sim = [\gamma]_\sim$. By definition this means $\sim_\eta = \sim_\gamma$. We show $t \sim_\eta t'$ if and only if $t \sim_\gamma t'$. From the definitions we get

$$\begin{array}{lll} t \sim_\eta t' & \text{iff} & t =_{\theta_1} t' \\ & \text{iff} & t =_{\theta_2} t'\ , \quad (\theta_1, \theta_2 \in [\theta]_=) \\ & \text{iff} & t \sim_\gamma t'\ . \end{array}$$

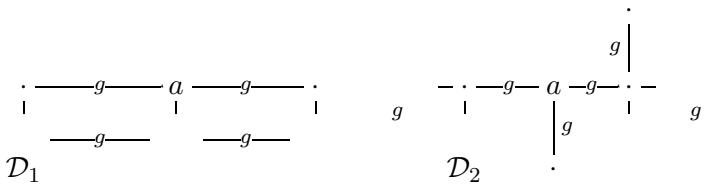
We continue by showing that the mapping is one-to-one. Assume $[\theta_1]_=, [\theta_2]_= \in \mathcal{R}(\mathcal{D}/\!\sim)/\!=$ are distinct. Then, without loss of generality there are $t, t'$ such that $t \neq_{\theta_1} t'$ and $t =_{\theta_2} t'$. Then there exists a context $C'[\,]$ such that $\theta_1(C'[t]) \neq \theta_1(C'[t'])$ while $\theta_2(C[t]) = \theta_2(C[t'])$ for all contexts $C[\,]$. Let $\eta$ and $\gamma$ be runs extracted from $\theta_1$ and $\theta_2$ respectively. Recall that if $\eta_\theta$ is extracted from $\theta$, then $\eta_\theta(t)$ is an element of $\theta(t) \in D/\!\sim$ for any term $t$. Then $C'[t] \not\sim_\eta C'[t']$ and $C[t] \sim_\gamma C[t']$ for all contexts $C[\,]$, i.e., $[\theta_1]_=$ and $[\theta_2]_=$ are mapped to distinct elements in $\mathcal{R}(\mathcal{D})/\!\sim$.

We conclude by showing that the mapping is onto. Choose $[\eta]_\sim \in \mathcal{R}(\mathcal{D})/\!\sim$. Define $\theta : T_\Sigma \longrightarrow D/\!\sim$ by $\theta(t) = [\eta(t)]_\sim$. By an inductive argument one can show that $\theta \in \mathcal{R}(\mathcal{D}/\!\sim)$. Let $\gamma$ be a run extracted from $\theta$. Then for any term $t$, $\gamma(t) \in \theta(t)$, i.e., $\gamma(t) \sim \eta(t)$. So

$$\begin{array}{rll} t \sim_\gamma t' & \text{iff} & \forall C[\,].\, \gamma(C[t]) \sim \gamma(C[t']) \\ & \text{iff} & \forall C[\,].\, \eta(C[t]) \sim \eta(C[t']) \\ & \text{iff} & t \sim_\eta t' \ , \end{array}$$

i.e., $[\gamma]_\sim = [\eta]_\sim$, hence $[\theta]_=$ is mapped to $[\eta]_\sim$. ■

Notice that taking the "quotient" on the spaces $\mathcal{R}(\mathcal{D}/\!\sim)$ and $\mathcal{R}(\mathcal{D})$ is necessary, as one of the following examples shows, where $a$ is a constant and $g$ a unary symbol.




Let $S \not\hookleftarrow S'$ denote that there exists no bijection between the sets $S$ and $S'$. Then there are $\mathcal{D}$-congruences $\sim$ such that

$$\begin{array}{lll} \mathcal{R}(\mathcal{D}_1) \not\hookleftarrow \mathcal{R}(\mathcal{D}_1/\!\sim) & \mathcal{R}(\mathcal{D}_1) \not\hookleftarrow \mathcal{R}(\mathcal{D}_1/\!\sim)/= & \mathcal{R}(\mathcal{D}_2/\!\sim) \not\hookleftarrow \mathcal{R}(\mathcal{D}_2/\!\sim)/= \\ \mathcal{R}(\mathcal{D}_1) \not\hookleftarrow \mathcal{R}(\mathcal{D}_1)/\!\sim & \mathcal{R}(\mathcal{D}_2/\!\sim) \not\hookleftarrow \mathcal{R}(\mathcal{D}_2)/\!\sim & \\ \mathcal{R}(\mathcal{D}_2) \not\hookleftarrow \mathcal{R}(\mathcal{D}_2)/= & \mathcal{R}(\mathcal{D}_1/\!\sim) \not\hookleftarrow \mathcal{R}(\mathcal{D}_1)/= & \end{array}$$

The following proposition shows that if a rational map is a refinement, then this property is preserved by the quotient construction.

**Proposition 203** *Given a refinement $\widehat{h} : \mathcal{R}(\mathcal{D}) \longrightarrow \mathcal{R}(\mathcal{D}')$. Let $\sim \subseteq D \times D$ be defined by $d \sim d'$ if and only if $h(d) = h(d')$. Let $\approx$ be any $\mathcal{D}$-congruence in $\sim$. Then $g : D/\!\approx \longrightarrow D'$ defined by $[d]_\approx \mapsto h(d)$ is well-defined and defines a refinement $\widehat{g} : \mathcal{R}(\mathcal{D}/\!\approx) \longrightarrow \mathcal{R}(\mathcal{D}')$.*

**Proof.** Clearly, since $\approx \subseteq \sim$, $g$ is well-defined. Given $f \in \Sigma_n$ and $[d_1]_\approx$, ..., $[d_n]_\approx \in D/\!\approx$. Then

$$\begin{array}{rcl} g(E/\!\approx_f([d_1]_\approx, \ldots, [d_n]_\approx)) & = & g(\{[d]_\approx \,|\, d \in E_f(d_1, \ldots, d_n)\}) \\ & = & h(\{d \,|\, d \in E_f(d_1, \ldots, d_n)\}) \\ & = & h(E_f(d_1, \ldots, d_n)) \\ & \subseteq & E'_f(h(d_1), \ldots, h(d_n)) \\ & = & E'_f(g([d_1]_\approx), \ldots, g([d_n]_\approx)) \ . \end{array}$$

Hence, $g$ defines a rational map. Given $\theta \in \mathcal{R}(\mathcal{D}/\!\approx)$. Let $\eta_\theta$ be a run extracted from $\theta$. Then $\eta_\theta(t) \in \theta(t)$, so

$$\widehat{g}(\theta)(t) = g(\theta(t)) = h(\eta_\theta(t)) = \widehat{h}(\eta_\theta)(t) \ ,$$

i.e., $\widehat{g}(\theta) = \widehat{h}(\eta_\theta)$. Hence, $\widehat{g}$ must be one-to-one.

Pick $\theta' \in \mathcal{R}(\mathcal{D}')$. Then there exists a $\eta \in \mathcal{R}(\mathcal{D})$ such that $\widehat{h}(\eta) = \theta'$. The map $\theta : T_\Sigma \longrightarrow D/\approx$, defined by $\theta(t) = [\eta(t)]_\approx$ is a run in $\mathcal{R}(\mathcal{D}/\approx)$, from which $\eta$ can be extracted. Hence, $\widehat{g}(\theta) = \widehat{h}(\eta) = \theta'$, and $\widehat{g}$ is onto. ■

## 8.5 Complexity of Rational Embeddings

In this section we consider decision problems related to rational embeddings between finitary rational spaces.

**Definition 204** Given $\mathcal{D}$. The *non-emptiness problem* is the problem of deciding whether or not $\mathcal{R}(\mathcal{D}) \neq \emptyset$. □

**Lemma 205** *Given a finite $\mathcal{D}$. The problem of deciding if $\mathcal{R}(\mathcal{D}) \neq \emptyset$ is NP-complete.*

**Proof.** To show NP-hardness, we reduce the NP-complete 3-CNF satisfiability problem [HU79] to the accessibility problem. We assume the reader is familiar with 3-CNF satisfiability.

Let $F = F_1 \wedge \cdots \wedge F_m$ be an expression in 3-CNF, where each $F_i$, $1 \leq i \leq m$, is a clause of the form $(\alpha_{i_1} \vee \alpha_{i_2} \vee \alpha_{i_3})$ and each $\alpha_{i_j}$, $1 \leq j \leq 3$, is a literal, i.e., a negated or non-negated boolean variable. Let $Var$ be the set $\{x_1, \ldots, x_k\}$ of boolean variables that occur in $F$. Let $Var(\alpha_{i_j}) = \ell$, where $x_\ell$ is the variable in $Var$ that occurs in $\alpha_{i_j}$.

Define $\Sigma = \Sigma_0 \cup \Sigma_3$, where

$$\begin{aligned} \Sigma_0 &= \{a_1, \ldots, a_k\} \\ \Sigma_3 &= \{f_1, \ldots, f_m\} \ . \end{aligned}$$

Define a $\Sigma$-hypergraph $\mathcal{D} = (D, E)$ by

$$\begin{aligned} D &= \{tt_1, \mathit{ff}_1, \ldots, tt_k, \mathit{ff}_k, *\} \\ E_{a_\ell} &= \{tt_\ell, \mathit{ff}_\ell\} \ , \quad 1 \leq \ell \leq k \ . \end{aligned}$$

In order to define $E_{f_i}$, let us say that $d \in D$ *matches* $\alpha_{i_j}$ if $d \in \{tt_\ell, \mathit{ff}_\ell\}$, where $\ell = Var(\alpha_{i_j})$. Intuitively, $d$ then corresponds to a truth assignment of $\alpha_{i_j}$ by interpreting $d = tt_\ell$ as $x_\ell$ being assigned the value $\mathsf{true}$ and $d = \mathit{ff}_\ell$ as $x_\ell$ being assigned the value $\mathsf{false}$. Assume $d_1, d_2$, and $d_3$ match $\alpha_{i_1}, \alpha_{i_2}$, and $\alpha_{i_3}$, respectively, and the corresponding truth assignments are consistent, i.e., if $d_{j_1}, d_{j_2} \in \{tt_\ell, \mathit{ff}_\ell\}$, $1 \leq j_1, j_2 \leq 3$, then $d_{j_1} = d_{j_2}$. Then $d_1, d_2$, and $d_3$ are said to *falsify* $F_i$, if $F_i$ evaluates to $\mathsf{false}$ under the corresponding truth assignments. Now define

$$E_{f_i}(d_1, d_2, d_3) = \begin{cases} \emptyset & , \text{ if } d_1, d_2, \text{ and } d_3 \text{ falsify } F_i \\ \{*\} & , \text{ else} \end{cases}$$

$\mathcal{D}$ can be build in time polynomial in the size of $F$ and has the property that $\mathcal{R}(\mathcal{D}) \neq \emptyset$ if and only if $F$ is satisfiable. An inductive argument shows that a satisfying truth

assignment $\sigma : Var \longrightarrow \{\mathsf{true},\mathsf{false}\}$ uniquely determines a run $\theta : T_\Sigma \longrightarrow D$ that maps $a_\ell$ to $tt_\ell$ or $ff_\ell$ depending on whether $\sigma(x_\ell)$ is $\mathsf{true}$ or $\mathsf{false}$, respectively. Also, from any run $\theta$ of $\mathcal{D}$, a satisfying truth assignment can be obtained by defining $\sigma(x_\ell)$ to be $\mathsf{true}$ or $\mathsf{false}$ depending on whether $\theta(a_\ell)$ is $tt_\ell$ or $ff_\ell$, respectively.

To see that the problem of determining whether not $\mathcal{R}(\mathcal{D}) \neq \emptyset$ lies in NP, just observe that a finite hypergraph $\mathcal{D}$ has a run if and only if it has an entire subhypergraph. ■

**Definition 206** Given $\mathcal{D}$ and $d \in D$. $d$ is said to be $\mathcal{D}$-*accessible* if there exists a run $\theta \in \mathcal{R}(\mathcal{D})$ and a ground term $t$ such that $\theta(t) = d$. The *accessibility problem* is the problem of deciding whether or not $d$ is $\mathcal{D}$-accessible. □

**Lemma 207** *Given a finite $\mathcal{D}$ and a $d \in D$. The accessibility problem is NP-hard.*

**Proof.** This follows from an easy modification of the proof of Lemma 205. ■

**Lemma 208** *Given a finite $\mathcal{D}$ and a $d \in D$. The accessibility problem lies in NP.*

**Proof.** From [AKW95] it follows that the problem can be reduced in time polynomial in the size of $\mathcal{D}$ to the Nonlinear Reachability Problem (NRP). Since the NRP is NP-complete [Ste94] it follows that the accessibility problem is in NP. ■

**Lemma 209** *Given a rational map $\widehat{h} : \mathcal{R}(\mathcal{D}) \longrightarrow \mathcal{R}(\mathcal{D}')$ between finitary rational spaces. Then, $\widehat{h}$ is a rational embedding if and only if there exists no $\Sigma$-hypergraph $\mathcal{D}^* = (D^*, E^*)$ such that*

$$\begin{array}{ll}
(i) & D^* \subseteq D \times D \text{ and } \forall (d,d') \in D^*.\, h(d) = h(d') \\
(ii) & (d,d') \in E^*_f((d_1,d'_1),\ldots,(d_n,d'_n)) \;\Rightarrow \\
 & \qquad (d \in E_f(d_1,\ldots,d_n) \wedge\; d' \in E_f(d'_1,\ldots,d'_n)) \\
(iii) & \exists (d_0,d'_0) \in D^*.\, d_0 \neq d'_0 \;\wedge\; (d_0,d'_0) \text{ is } \mathcal{D}^*\text{-accessible.}
\end{array}$$

**Proof.** Assume $\widehat{h}$ is not a rational embedding. Choose $\eta_1, \eta_2 \in \mathcal{R}(\mathcal{D})$ such that $\widehat{h}(\eta_1) = \widehat{h}(\eta_2)$ and $\eta_1 \neq \eta_2$. Let $t_0$ be a ground term such that $\eta_1(t_0) \neq \eta_2(t_0)$. Let

$$D^* \;=\; \{(\eta_1(t), \eta_2(t)) \,|\, t \in T_\Sigma\}\ .$$

For $f \in \Sigma_n$ and $(d_1,d'_1),\ldots,(d_n,d'_n) \in D^*$, define

$$\begin{array}{rcl}
E^*_f((d_1,d'_1),\ldots,(d_n,d'_n)) & = & \{(d,d') \,|\, \exists f(t_1,\ldots,t_n) \in T_\Sigma. \\
 & & (\eta_1(t_1),\eta_2(t_1)) = (d_1,d'_1),\ldots, \\
 & & (\eta_1(t_n),\eta_2(t_n)) = (d_n,d'_n) \;\wedge \\
 & & (\eta_1(f(t_1,\ldots,t_n)),\eta_2(f(t_1,\ldots,t_n))) = (d,d')\}\ .
\end{array}$$

Then (i) holds because $\widehat{h}(\eta_1) = \widehat{h}(\eta_2)$ and (ii) because $\eta_1$ and $\eta_2$ are runs over $\mathcal{D}$. Also, (iii) holds because $\theta : T_\Sigma \longrightarrow D^*$ defined by $\theta(t) = (\eta_1(t), \eta_2(t))$ is a run of $\mathcal{D}^*$ and $\theta(t_0) = (d_0, d'_0)$, where $d_0 \neq d'_0$.

Conversely, assume (i)–(iii) are true. Let $\theta$ be a run witnessing that $(d_0, d_0')$ is $\mathcal{D}^*$-accessible. Define $\eta_1 : T_\Sigma \longrightarrow D$ and $\eta_2 : T_\Sigma \longrightarrow D$ by $\eta_1(t) = d$ and $\eta_2(t) = d'$, where $\theta(t) = (d, d')$. By (ii) $\eta_1$ and $\eta_2$ are runs of $\mathcal{D}$, by (i) $\widehat{h}(\eta_1) = \widehat{h}(\eta_2)$, and by (iii) $\eta_1 \neq \eta_2$. Hence, $\widehat{h}$ is not a rational embedding. ∎

Based on the above lemmas we now obtain:

**Theorem 210** *Given a rational map $\widehat{h} : \mathcal{R}(\mathcal{D}) \longrightarrow \mathcal{R}(\mathcal{D}')$ between finitary rational spaces. The problem of deciding whether or not $\widehat{h}$ is a rational embedding is co-NP-complete.*

**Proof.** To show the hardness, we reduce the complement of the non-emptiness problem to this problem. Given $\mathcal{D}'$, let $\mathcal{D}$ denote the disjoint union of two copies of $\mathcal{D}'$, and let $\widehat{h} : D \longrightarrow D'$ denote the function which maps a state in $D$ to the state in $D'$ of which it is a copy. $\widehat{h}$ is a rational embedding if and only if $\mathcal{R}(\mathcal{D}) = \emptyset$.

From Lemma 209 it follows that the problem lies in co-NP, because the problem of determining the existence of a $\mathcal{D}^*$ as defined in the proof of Lemma 209 can be shown to be in NP using Lemma 208. ∎

**Corollary 211** *Given a finitary rational space $\mathcal{R}(\mathcal{D})$. The problem of deciding whether or not $|\mathcal{R}(\mathcal{D})| > 1$ is NP-complete.*

**Proof.** The mapping $h : D \longrightarrow \{*\}$ defines a rational map $\widehat{h} : \mathcal{R}(\mathcal{D}) \longrightarrow \mathcal{R}(\mathcal{E})$, where $\mathcal{E}$ is the entire $\Sigma$-hypergraph over $\{*\}$, that is not a rational embedding if and only if $|\mathcal{R}(\mathcal{D})| > 1$. ∎

**Theorem 212** *Given two finitary rational spaces $\mathcal{R}(\mathcal{D})$ and $\mathcal{R}(\mathcal{D}')$. The problem of deciding whether or not there exists a rational embedding $\widehat{h} : \mathcal{R}(\mathcal{D}) \longrightarrow \mathcal{R}(\mathcal{D}')$ is NP-hard, co-NP-hard, and lies in $\Sigma_2^P$.*

**Proof.** The co-NP-hardness result can be obtained by combining the techniques from Theorem 210 and Corollary 211.

The NP-hardness result can be obtained by a reduction from the satisfiability problem for 3-CNF. Given $F$, a 3-CNF, one constructs $\mathcal{D}$ and $\mathcal{D}'$, such that $\mathcal{R}(\mathcal{D})$ is a singleton and such that there exists a rational map $\widehat{h} : \mathcal{R}(\mathcal{D}) \longrightarrow \mathcal{R}(\mathcal{D}')$ if and only if $F$ is satisfiable. The constructions resembles then one use in the proof of Lemma 205.

To see that the problems lies in $\Sigma_2^P$, observe that it can be formulated as

$$\{s_{(\mathcal{D},\mathcal{D}')} \mid \exists s_h.\, \forall s_{\mathcal{D}^*}, s_{(d_0,d_0')}, s_{comp(\mathcal{D}^*,(d_0,d_0'))}.\\ (s_h, s_{\mathcal{D}^*}, s_{(d_0,d_0')}, s_{comp(\mathcal{D}^*,(d_0,d_0'))}, s_{(\mathcal{D},\mathcal{D}')}) \in L\}\ ,$$

where $|s_h| < p(|s_{(\mathcal{D},\mathcal{D}')}|)$ and $|s_{\mathcal{D}^*} s_{(d_0,d_0')} s_{comp(\mathcal{D}^*,(d_0,d_0'))}| < p(|s_{(\mathcal{D},\mathcal{D}')}|)$ for some polynomial $p$, and $L \in P$ is the language of a deterministic polynomial time machine that checks that if

- $s_{(\mathcal{D},\mathcal{D}')}$ encodes two hypergraphs $\mathcal{D}$ and $\mathcal{D}'$,
- $s_h$ encodes a mapping $h : D \longrightarrow D'$ satisfying (8.6),
- $s_{\mathcal{D}^*}$ encodes a hypergraph as described in the proof of Lemma 209, given $\mathcal{D}$, $\mathcal{D}'$, and $h$, and
- $s_{(d_0,d_0')}$ encodes a pair $(d_0, d_0')$ belonging to $\mathcal{D}^*$ such that $d_0 \neq d_0'$,

then $s_{comp(\mathcal{D}^*,(d_0,d_0'))}$ does not encode an accepting computation on input $(s_{\mathcal{D}^*}, s_{(d_0,d_0')})$ of a (given) nondeterministic polynomial time Turing machine solving the accessibility problem, given the encoding $(s_{\mathcal{D}^*}, s_{(d_0,d_0')})$. ■

## 8.6 Summary

We have continued the investigation of the rational spaces introduced in [Koz95]. We have given a Myhill-Nerode-like characterisation of rational points and results that suggests that rational points in an essential way captures the topological structure of finitary rational spaces. In [Koz94], a congruence on Σ-hypergraphs akin to that investigated here occur in the context of efficient constraint solving. We investigated the interplay between congruences on Σ-hypergraphs and equivalence relations on the corresponding rational spaces. Finally, we investigated complexity issues for rational maps.

As for future work, we have identified several interesting directions. One is the remaining complexity issues. The gap in Theorem 212 suggests itself immediately, likewise decision problems concerning surjectivety of rational maps (between finitary rational spaces). These problems are closely related to the problem of deciding if a given rational map is a refinement, or whether or not two finitary rational spaces are rationally equivalent. A first step could be to show that if two finitary rational spaces are rationally equivalent because there exists a span of refinements from a common rational space, then there must also exist span of refinements from a common finitary rational space (and preferably a bound on its underlying hypergraph).

Also, since rational equivalence is defined as a span of refinements, one could try to apply the theory of open maps on the category of rational spaces.

# Chapter 9

# Conclusion

Reasoning about concurrent computational systems is a challenging task. One of the most successful approaches is model-checking. What makes it rather unique is that it is based on a well-developed theory. Verification tools have been developed based on studies of the expressive power of temporal logics and of the computational complexity of model-checking these logics. Different semantics have turned out to be useful for deriving clever algorithms and in providing us with a more accurate view of concurrency. However, the general picture is still not complete. Our results complete parts of the picture, as well as widening it; partial order semantics are becoming increasingly popular.

Moving to the field of semantical models of concurrency, in which "true concurrency" models have flourished in the same period, recent work has employed category theory as a common framework in which different models could be related and compared formally. Concrete ideas and notions have been re-casted abstractly; $\mathcal{P}$-bisimilarity, an abstract notion of bisimulation, has been presented by Joyal, Nielsen, and Winskel. We have examined its applicability and proposed a new notion of $\mathcal{P}$-factorisability, as an abstract way of capturing congruence properties of $\mathcal{P}$-bisimilarity.

Set constraints have also recently received much attention because of their wide applicability. Yet, many interesting results about them are now known. Especially on the theoretical side, there has recently been fast progress in understanding the mathematical structure and computational complexity of various classes of set constraints. We have contributed with an axiomatisation of set constraints as well as with an investigation of Kozen's rational spaces.

As sketched in our summaries, there are still many interesting directions to go. Especially the two last parts present results in two rather new areas, whose general pictures are currently only outlined.

# Bibliography


[AJ92] S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. In *Proceedings, Symposium on Logic in Computer Science*, pages 211–222. IEEE, June 1992.

[AKVW93] Alexander Aiken, Dexter Kozen, Moshe Vardi, and Edward Wimmers. The complexity of set constraints. In E. Börger, Y. Gurevich, and K. Meinke, editors, *Proc. 1993 Conf. Computer Science Logic (CSL'93)*, volume 832 of *Lect. Notes in Comput. Sci.*, pages 1–17. Eur. Assoc. Comput. Sci. Logic, Springer, September 1993.

[AKW95] Alexander Aiken, Dexter Kozen, and Edward Wimmers. Decidability of systems of set constraints with negative constraints. *Infor. and Comput.*, 122(1):30–44, October 1995.

[AL94] A. Aiken and T.K. Lakshman. Directional type checking of logic programs. In *Proceedings of the 1st International Static Analysis Symposium*, September 1994.

[AM91a] A. Aiken and B. Murphy. Implementing regular tree expressions. In *Proc. 1991 Conf. Functional Programming Languages and Computer Architecture*, pages 427–447, August 1991.

[AM91b] A. Aiken and B. Murphy. Static type inference in a dynamically typed language. In *Proc. 18th Symp. Principles of Programming Languages*, pages 279–290. ACM, January 1991.

[And95] Henrik Reif Andersen. Partial model checking. In *Proc. LICS'95, Tenth Annual Symposium on Logic in Computer Science*, pages 398–407, 1995.

[APP95] Rajeev Alur, Doron Peled, and Wojciech Penczek. Model-checking of causality properies. In *Proceedings, Symposium on Logic in Computer Science*, pages 90–100. IEEE, 1995.

[AW92] A. Aiken and E. Wimmers. Solving systems of set constraints. In *Proc. 7th Symp. Logic in Computer Science*, pages 329–340. IEEE, June 1992.

[AW93] A. Aiken and E. Wimmers. Type inclusion constraints and type inference. In *Proceedings fo the 1993 Conference on Functional Programming Languages and Computer Architecture*, pages 31–41, Denmark, June 1993.

[AWL94] A. Aiken, E. Wimmers, and T.K. Lakshman. Soft typing with conditional types. In *Twenty-First Annual ACM Symposium on Principles of Programming Languages*, pages 163–173, Portland, Oregon, January 1994.

[BC92] Luca Bernardinello and Fiorella De Cindio. A survey of basic net models and modular net classes. In *Advances in Petri Nets 1992*, pages 304–351. Springer-Verlag (*LNCS* 609), 1992.

[BCM+92] J. R. Bruch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: $10^{20}$ states and beyond. *Information and Computation*, 98:142–170, 1992.

[BD90] Eike Best and Jörg Desel. Partial order behaviour and structure of Petri nets. *Formal Aspects of Computing*, 2:123–138, 1990.

[BDH92] Eike Best, Raymond Devillers, and Jon G. Hall. The Box calculus: a new causal algebra with multi-label communication. In *Advances in Petri Nets 1992*, pages 21–69. Springer-Verlag (*LNCS* 609), 1992.

[Bed88] M. A. Bednarczyk. *Categories of asynchronous systems.* PhD thesis, University of Sussex, 1988. PhD in computer science, report no.1/88.

[Ber66] R. Berger. The undecidability of the domino problem. *Memoirs American Math. Sco.*, 66, 1966.

[BF88] Eike Best and César Fernández. *Nonsequential Processes – a Petri Net View.* EATCS Monographs on Theoretical Computer Science Vol.13, 1988.

[BG94] O. Bernholtz and O. Grumberg. Buy one, get one free !!! In *Proceedings of the First international Conference on Temporal Logic*, volume 827 of *Lect. Notes in Comput. Sci.*, pages 210–224. Springer, 1994.

[BGW93] L. Bachmair, H. Ganzinger, and U. Waldmann. Set constraints are the monadic class. In *Proc. 8th Symp. Logic in Computer Science*, pages 75–83. IEEE, June 1993.

[BIM88] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation Can't be Traced. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 229–239, 1988.

[Bra91] Julian Bradfield. *Verifying Temporal Properties of Systems with Applications to Petri Nets.* PhD thesis, The University of Edinburgh, 1991. PhD in computer science, report CST-83-91.

[BS92] Julian Bradfield and Colin P. Stirling. Local model checking for infinite state spaces. *Theoretical Computer Science*, 96:157–174, 1992.

[BT87] Eike Best and P.S. Thiagarajan. Some classes of live and save Petri nets. In K. Voss, H.J. Genrich, and G. Rozenberg, editors, *Advances in Petri Nets*, pages 71–94. Springer-Verlag, 1987.

[BVW94] O. Bernholtz, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *Computer Aided Verification, Proc. of 6th International Workshop*, Lect. Notes in Comput. Sci., pages 142—155. Springer, june 1994.

[CEP93] Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. In *Proc. FST&TCS 13, Thirteenth Conference on the Foundations of Software Technology & Theoretical Computer Science*, pages 326–337. Springer-Verlag (*LNCS* 761), Bombay, India, December 1993. A journal version appears in TCS, volume 147.

[CEP95] Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. *Theoretical Computer Science*, 147(1-2):117–136, August 1995.

[CES86] Edmund M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent system using temporal logic. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[Che95a] Allan Cheng. Complexity Results for Model Checking. Research Series RS-95-18, BRICS, Department of Computer Science, University of Aarhus, February 1995.

[Che95b] Allan Cheng. Petri nets, traces, and local model checking. In *Proceedings of AMAST'95, Algebraic Methodology and Software Technology*, pages 322–337. Springer-Verlag (*LNCS* 936), Montréal, Canada, July 1995.

[Che95c] Allan Cheng. Petri nets, traces, and local model checking. Research Series RS-95-39, BRICS, Department of Computer Science, University of Aarhus, July 1995. 32 pp. Full version of paper appearing in Proceedings of AMAST '95, LNCS 936, 1995.

[CHEP71] Fred Commoner, Anatole W. Holt, S. Even, and Amir Pnueli. Marked directed graphs. *Journal of Computer and System Sciences*, 5:511–523, 1971.

[CK95] Allan Cheng and Dexter Kozen. A complete Gentzen-style axiomatization for set constraints. Technical Report TR95-1518, Cornell University, May 1995. To be presented at ICALP'96.

[CK96] Allan Cheng and Dexter Kozen. A complete Gentzen-style axiomatization for set constraints. In F. Meyer auf der Heide and B. Monien, editors, *Proc. ICALP'96*, volume 1099 of *Lect. Notes in Comp. Sci.*, pages 134–145. Springer, July 1996.

[CN95] Allan Cheng and Mogens Nielsen. Open maps (at) work. Research Series RS-95-23, BRICS, Department of Computer Science, University of Aarhus, April 1995. 33 pp.

[CN96] Allan Cheng and Mogens Nielsen. Open maps, behavioural equivalences, and congruences. Research Series RS-96-2, BRICS, Department of Computer Science, University of Aarhus, jan 1996. A short version of this paper is to appear in the proceedings of CAAP '96.

[Col82] A. Colmerauer. PROLOG and infinite trees. In S.-A. Tärnlund and K. L. Clark, editors, *Logic Programming*, pages 231–251. Academic Press, January 1982.

[Cou83] Bruno Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25(1):95–169, 1983.

[CP94a] W. Charatonik and L. Pacholski. Negative set constraints with equality. In *Proc. 9th Symp. Logic in Computer Science*, pages 128–136. IEEE, July 1994.

[CP94b] W. Charatonik and L. Pacholski. Set constraints with projections are in *NEXPTIME*. In *Proc. 35th Symp. Foundations of Computer Science*, pages 642–653. IEEE, November 1994.

[CPS89] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: A semantics based tool for the verification of concurrent systems. Technical Report LFCS Report Series, ECS-LFCS-89-83, Department of Computer Science, University of Edinburgh, 1989.

[CW96] G-L. Cattani and G. Winskel. Presheaf models for concurrency. Technical report, BRICS, 1996. To appear.

[DE91] Jörg Desel and Javier Esparza. Reachability in reversible free choice systems. In *Proc. STACS'91*, pages 384–397. Springer-Verlag (*LNCS* 480), 1991.

[DE93a] J. Desel and J. Esparza. Reachability in cyclic extended free-choice systems. *Theoretical Computer Science*, 43:99–105, 1993.

[DE93b] Jörg Desel and Javier Esparza. Shortest paths in reachability graphs. In *Proc. Application and Theory of Petri Nets*, pages 224–241. Springer-Verlag (*LNCS* 691), 1993. To appear in Journal of Computer and System Sciences.

[Des92] Jörg Desel. A proof of the rank theorem for extended free choice nets. In *Application and Theory of Petri Nets 1992*, pages 134–153. Springer-Verlag (*LNCS* 616), 1992.

[DNM88] Pierpaolo Degano, Rocco De Nicola, and Ugo Montanari. A distributed operational semantics for CCS based on C/E systems. *Acta Informatica*, 26:59–91, 1988.

[EH86] E.A. Emerson and J.Y. Halpern. "sometimes" and "not never" revisited. *Journal of the ACM*, 33(1):151–178, 1986.

[Eme90] E. Allen Emerson. *Temporal and Modal Logic*, chapter 16. Elsevier Science Publishers, 1990. in Handbook Of Theoretical Computer Science, editor J. van Leeuwen.

[EN94] Javier Esparza and Mogens Nielsen. Decidability issues for petri nets. *Journal of Information Processing and Cybernet. EIK*, 30:143–160, 1994.

[ES92] Javier Esparza and Manuel Silva. A polynomial-time algorithm to decide liveness of bounded free choice nets. *Theoretical Computer Science*, 102:185–205, 1992.

[Esp93] Javier Esparza. Model checking using net unfoldings. In *Proc. TAPSOFT'93*, pages 613–628. Springer-Verlag (*LNCS* 668), 1993. Full version to appear in Science of Computer Programming.

[GG89] R. J. Van Glabeek and U. Goltz. Equivalence notions for concurrent systems and refinement of actions. In *Proceedings of MFCS*, pages 237–248. Springer-Verlag (*LNCS* 379), 1989.

[Gir87a] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[Gir87b] J.-Y. Girard. Towards a geometry of interactions. *Categories in Computer Science and Logic, Contemporary Mathematics*, 22:69–108, 1987.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability.* Freeman, 1979.

[GL73] Hartmann J. Genrich and Kurt Lautenbach. Synchronisationsgraphen. *Acta Informatica*, 2:143–161, 1973.

[God90] Patrice Godefroid. Using partial orders to improve automatic verification methods. In *Proc. CAV'90, 2nd Workshop on Computer-Aided Verification*, pages 176–185. Springer-Verlag (*LNCS* 531), 1990.

[Gol88] Ursula Goltz. On representing CCS programs by finite Petri nets. In *Proc. MFCS'88, Mathematical Foundations of Computer Science*, pages 339–350. Springer-Verlag (*LNCS* 324), 1988.

[GTT93a] R. Gilleron, S. Tison, and M. Tommasi. Solving systems of set constraints using tree automata. In *Proc. Symp. Theor. Aspects of Comput. Sci.*, volume 665, pages 505–514. Springer-Verlag Lect. Notes in Comput. Sci., February 1993.

[GTT93b] R. Gilleron, S. Tison, and M. Tommasi. Solving systems of set constraints with negated subset relationships. In *Proc. 34th Symp. Foundations of Comput. Sci.*, pages 372–380. IEEE, November 1993.

[GW91] Patrice Godefroid and Pierre Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In Kim G. Larsen and Arne Skou, editors, *Proc. CAV'91, Computer-Aided Verification*, pages 332–343. Springer-Verlag (*LNCS* 575), 1991.

[GW94] Patrice Godefroid and Pierre Wolper. A partial approach to model checking. *Information and Computation*, 110(2):305–326, 1994.

[Hac72] Michel Hack. Analysis of production schemata by Petri nets. Master's thesis, MIT, 1972.

[Hac74] Michel Hack. The recursive equivalence of the reachability problem and the liveness problem for Petri nets and vector addition systems. In *Proc. 15th Annual Symposium on Switching and Automata Theory*, pages 156–164, 1974.

[Hac76] Michel Hack. The equality problem for vector addition systems is undecidable. *Theoretical Computer Science*, 2:77–95, 1976.

[Har85] David Harel. Recurring dominoes: making the highly undecidable highly understandable. *Ann. Disc. Math.*, 24:51–72, 1985.

[Hei92] Nevin Heintze. *Set Based Program Analysis.* PhD thesis, Carnegie Mellon University, 1992.

[Hei94] N. Heintze. Set-based analysis of ml programs (extended abstract). In *Proceedings of the 1994 ACM Conference on Lisp and Functional Programming*, June 1994.

[Hen88] Matthew Hennessy. *Algebraic Theory of Processes.* MIT Press series in the foundations of computing, 1988.

[HJ90a] N. Heintze and J. Jaffar. A decision procedure for a class of set constraints. In *Proc. 5th Symp. Logic in Computer Science*, pages 42–51. IEEE, June 1990.

[HJ90b] N. Heintze and J. Jaffar. A finite presentation theorem for approximating logic programs. In *Proc. 17th Symp. Principles of Programming Languages*, pages 197–209. ACM, January 1990.

[HJ92] N. Heintze and J. Jaffar. An engine for logic program analysis. In *Proceedings, Symposium on Logic in Computer Science*, pages 318–328. IEEE, June 1992.

[HJJ+95] Jesper G. Henriksen, Ole J.L. Jensen, Michael E. Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders B. Sandholm. Mona: Monadic second-order logic in practice. In Brinksma, Cleaveland, Larsen, Margaria, and Steffen, editors, *Tools and Algorithms for The Construction and Analysis of Systems: International Workshop, TACAS '95*, volume 1019 of *Lect. Notes in Comput. Sci.*, pages 89–110. Springer, 1995.

[HR88] Rodney R. Howell and Louis E. Rosier. Completeness results for conflict-free vector replacement systems. *Journal of Computer and System Sciences*, 37:349–366, 1988.

[HR89] Rodney R. Howell and Louis E. Rosier. Problems concerning fairness and temporal logic for conflict-free Petri nets. *Theoretical Computer Science*, 64(3):305–329, 1989.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 1979.

[Jan86] M. Jantzen. Complexity of place/transition nets. In *Advances in Petri Nets 86*, volume 254 of *Lect. Notes in Comput. Sci.*, pages 413–435. Springer, 1986.

[Jat93] Lalita Jategaonkar. *Observing "True" Concurrency.* PhD thesis, MIT, July 1993. MIT/LCS/TR-618.

[JLL77] Neil D. Jones, Lawrence H. Landweber, and Y. Edmund Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4:277–299, 1977.

[JM79] N. D. Jones and S. S. Muchnick. Flow analysis and optimization of LISP-like structures. In *Proc. 6th Symp. Principles of Programming Languages*, pages 244–256. ACM, January 1979.

[JM93] Lalita Jategaonkar and Albert Meyer. Deciding true concurrency equivalences on finite safe nets. In *Proc. ICALP'93*, pages 519–531, 1993.

[JM94] A. Joyal and I. Moerdijk. A completeness theorem for open maps. *Annals of Pure and Applied Logic*, 70:51–86, 1994.

[JNW93] André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation and open maps. In *Proc. LICS'93, Eighth Annual Symposium on Logic in Computer Science*, pages 418–427, 1993.

[JNW94] André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from open maps. Research Series RS-94-7, BRICS, Department of Computer Science, University of Aarhus, May 1994. 42 pp. Journal version of LICS '93 paper.

[JT51] B. Jónsson and A. Tarski. Boolean algebras with operators. *Amer. J. Math.*, 73:891–939, 1951.

[JT52] B. Jónsson and A. Tarski. Boolean algebras with operators. *Amer. J. Math.*, 74:127–162, 1952.

[Kei76] H. Jerome Keisler. *Foundations of Infinitesimal Calculus.* Prindle, Weber & Schmidt, Incorporated, 1976.

[KM69] R.M. Karp and R.E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3:147–195, 1969.

[Kos82] S.R. Kosaraju. Decidability of reachability in vector addition systems. In *Proceedings of 14th Annual ACM Symposium on Theory of Computing*, pages 267–281, 1982.

[Koz93] Dexter Kozen. Logical aspects of set constraints. In E. Börger, Y. Gurevich, and K. Meinke, editors, *Proc. 1993 Conf. Computer Science Logic (CSL'93)*, volume 832 of *Lect. Notes in Comput. Sci.*, pages 175–188. Eur. Assoc. Comput. Sci. Logic, Springer, September 1993.

[Koz94] Dexter Kozen. Set constraints and logic programming (abstract). In J.-P. Jouannaud, editor, *Proc. First Conf. Constraints in Computational Logics (CCL'94)*, volume 845 of *Lect. Notes in Comput. Sci.*, pages 302–303. ESPRIT, Springer, September 1994.

[Koz95] Dexter Kozen. Rational spaces and set constraints. In Peter D. Mosses, Mogens Nielsen, and Michael I. Schwartzbach, editors, *Proc. Sixth Int. Joint Conf. Theory and Practice of Software Develop. (TAPSOFT'95)*, volume 915 of *Lect. Notes in Comput. Sci.*, pages 42–61. Springer, May 1995.

[KPS92] Dexter Kozen, Jens Palsberg, and Michael I. Schwartzbach. Efficient inference of partial types. In *Proc. 33rd Symp. Found. Comput. Sci.*, pages 363–371. IEEE, October 1992.

[KPS93] Dexter Kozen, Jens Palsberg, and Michael I. Schwartzbach. Efficient recursive subtyping. In *Proc. 20th Symp. Princip. Programming Lang.*, pages 419–428. ACM, January 1993.

[KPS94] Dexter Kozen, Jens Palsberg, and Michael I. Schwartzbach. Efficient inference of partial types. *J. Comput. Syst. Sci.*, 49(2):306–324, October 1994.

[Kup95] Orna Kupferman. *Model Checking for Branching-Time Temporal Logics.* PhD thesis, Technion, Israil Institue of Technology, 1995. Sivan 5754 Haifa June 1995.

[KV92] Kaivola and A. Valmari. The weakest compositional semantic equivalence preserving nexttime-less linear temporal logic. In *CONCUR'92*, volume 630 of *Lect. Notes in Comput. Sci.* Springer, 1992.

[Kwi89] Marta Z. Kwiatkowska. Event fairness and non-interleaving concurrency. *Formal Aspects of Computing*, 1:213–228, 1989.

[Lam80] Leslie Lamport. "Not never" – on the Temporal Logic of programs. *Proc. 7th Ann. ACM Symp. on Princples of Programming Languages*, pages 174–185, 1980.

[Lar88] Kim G. Larsen. Proof systems for Hennessy-Milner logic with recursion. In *Proceedings of CAAP, Nancy France*, pages 215–230. Springer-Verlag (*LNCS* 299), March 1988.

[Lip76] Richard J. Lipton. The reachability problem requires exponential space. Technical Report 62, Yale University, 1976.

[LPRT93] Kamal Lodaya, Rohit Parikh, R. Ramanujam, and P. S. Thiagarajan. A logical study of distributed transition systems. Technical report, School of Mathematics, SPIC Science Foundation, Madras, 1993. To appear in Information and Computation, a preliminary version appears as Report TCS–93–8.

[LR75] Lawrence H. Landweber and E. L. Robertson. Properties of conflict-free and persistent Petri nets. *Journal of the ACM*, 3:352–364, 1975.

[LS91] Kim G. Larsen and Arne Skou. Bisimulation through Probabilistic Testing. *Information and Computation*, 94:1–28, 1991.

[Mar77] A. Marzurkiewicz. Concurrent program schemes and their interpretations. Technical report, Daimi, University of Aarhus, 1977.

[May81] E.W. Mayr. Persistence of vector replacement systems is decidable. *Acta Informatica*, 15:309–318, 1981.

[May84] Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing*, 13:441–460, 1984.

[Maz86] Antoni Mazurkiewicz. Trace theory. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, pages 279–324. Springer-Verlag (*LNCS* 255), 1986.

[McM92] Kenneth L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Proc. CAV'92, Fourth Workshop on Computer-Aided Verification*, pages 164–174, 1992.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lect. Notes in Comput. Sci.* Springer, 1980.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall International Series In Computer Science, C. A. R. Hoare series editor, 1989.

[Mil90] Robin Milner. *Operational and Algebraic Semantics of Concurrent Processes*, chapter 19. Elsevier Science Publishers, 1990. in Handbook Of Theoretical Computer Science, editor J. van Leeuwen.

[Mis84] P. Mishra. Towards a theory of types in PROLOG. In *Proc. 1st Symp. Logic Programming*, pages 289–298. IEEE, 1984.

[MM90] José Meseguer and Ugo Montanari. Petri nets are monoids. *Information and Computation*, 88:105–155, 1990.

[MMP95] A. Mifsud, R. Milner, and J. Power. Control structures. In *Proceedings, Symposium on Logic in Computer Science*, pages 188–198. IEEE, June 1995.

[MN92] Madhavan Mukund and Mogens Nielsen. CCS, Locations and Asynchronous Transition Systems. *Proc. Foundations of Software Technology and Theoretical Computer Science 12*, pages 328–341, 1992.

[MOP89] Antoni Mazurkiewicz, Edward Ochmański, and Wojciech Penczek. Concurrent systems and inevitability. *Theoretical Computer Science*, 64():281–304, 1989.

[MP92] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer Verlag, 1992.

[MR85] P. Mishra and U. Reddy. Declaration-free type checking. In *Proc. 12th Symp. Principles of Programming Languages*, pages 7–21. ACM, 1985.

[MS92] Robin Milner and Davide Sangiorgi. Barbed Bisimulation. In *Automata, Languages and Programming, 19th International Colloquium, Wien, Austria (Proc. ICALP'92)*, pages 685–695. Springer-Verlag (*LNCS* 623), July 1992.

[Mur89] Tadao Murata. Petri nets: Properties, analysis and applications. In *Proc. of the IEEE*, volume 77(4), pages 541–580, 1989.

[NC94] Mogens Nielsen and Christian Clausen. Bisimulations, games and logic. In *ew Results and Trends in Computer Science*, volume 812 of *Lect. Notes in Comput. Sci.*, pages 289–305. Springer, 1994.

[NC95] Mogens Nielsen and Allan Cheng. Observe behaviour categorically. In *Proc. FST&TCS 15, Fifteenth Conference on the Foundations of Software Technology & Theoretical Computer Science*, pages 263–278. Springer-Verlag (*LNCS* 1026), Bangalore, India, December 1995.

[NPW81] Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri nets, event structures and domains, Part I. *Theoretical Computer Science*, 13:85–108, 1981.

[NRT90] Mogens Nielsen, Grzegorz Rozenberg, and P.S. Thiagarajan. Behavioural notions for elementary net systems. *Distributed Computing*, 4(1):45–57, 1990.

[NW95] Mogens Nielsen and Glynn Winskel. Petri nets and bisimulations. Research Series RS-95-4, BRICS, Department of Computer Science, University of Aarhus, January 1995. 36 pp. To appear in *Theoretical Computer Science*, 1995.

[Old91] Ernst R. Olderog. *Nets, Terms and Formulas*. Cambridge University Press, 1991. *Number 23 Tracts in Theoretical Computer Science*.

[Par81] D. Park. Concurrency adn automata on infinite sequences. In *Theoretic Computer Science*, volume 104 of *Lect. Notes in Comput. Sci*, pages 167–183. Springer, 1981.

[Pen93] Wojciech Penzcek. Temporal logics for trace systems: On automated verification. *International Journal of Foundations of Computer Science*, 4 (1):31–67, 1993.

[Pet62] C.A. Petri. Kommunikation mit automaten. Schriften des iim nr. 2, Bonn: Institut für Instrumentelle Mathematik, 1962. Also in: New York: Griffiss Air Force Base, Technical Report RADC-TR-65–377, Vol.1 Suppl. 1, 1966. English translation.

[Pet81] J. L. Peterson. *Petri net theory and the modeling af systems*. Prentice-Hall, 1981.

[PK95] Wojciech Penczek and Ruurd Kuiper. *Traces and Logic, eds. V. Diekert and G. Rozenberg*, chapter 10. World Scientific Publishing Co. Pte. Ltd., 1995.

[Plo81] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN–19, Computer Science Department, Aarhus University, September 1981.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th Ann. IEEE symp. on Foundations of Computer Science*, pages 46–57, 1977.

[Pnu85] A. Pnueli. Linear and branching structures in the semantics and logics fo reactive systems. In W. Brauer, editor, *Automata, Languages and Programming, 12th International Colloquium, Nafplion, Greece (Proc. ICALP'85)*, volume 194 of *Lect. Notes in Comput. Sci.*, pages 15–32. Springer, 1985.

[PP90] Doron Peled and Amir Pnueli. Proving partial order liveness properties. In *Proc. ICALP'90*, pages 553–571. Springer-Verlag (*LNCS* 443), 1990.

[Pra86] V. Pratt. Modelling Concurrency with Partial Orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986.

[Rei85] Wolfgang Reisig. *Petri Nets – An Introduction.* EATCS Monographs in Computer Science Vol.4, 1985.

[Rey69] J. C. Reynolds. Automatic computation of data set definitions. In *Information Processing 68*, pages 456–461. North-Holland, 1969.

[RT88] A. Rabinovitch and B. Trakhtenbrot. Behaviour structures and nets. *Fundamenta Informatica*, 11(4):357–404, 1988.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of Propositional Linear Temporal Logic. *Journal of the ACM*, 32(3):733–749, 1985.

[Shi85] M. W. Shields. Concurrent machines. *Computer Journal*, 28:449–465, 1985.

[Sta89] Eugene W. Stark. Concurrent transition systems. *Theoretical Computer Science*, 64():221–269, 1989.

[Sta90] Peter H. Starke. *Analyse von Petri-Netz-Modellen.* Teubner, 1990.

[Ste92] Iain A. Stewart. On the reachability problem for some classes of Petri nets. Research Report, University of Newcastle upon Tyne, to appear in Fundamenta Informaticae., 1992.

[Ste94] K. Stefánsson. Systems of set constraints with negative constraints are NEXPTIME-complete. In *Proc. 9th Symp. Logic in Computer Science*, pages 137–141. IEEE, June 1994.

[Sti93] Colin P. Stirling. Modal and temporal logics for processes. Technical report, Department of Computer Science, University of Edinburgh, August 1993. Stirling-1, Hand-outs at Summerschool in Logical Methods In Concurrency Aarhus'93.

[SW89] Colin P. Stirling and David Walker. Local model checking in the modal mu-calculus. Technical Report ECS–LFCS–89–78, Laboratory for Foundations of Computer Science, Department of Computer Science – University of Edinburgh, May 1989.

[Tau89] D. Taubner. *Finite Representations of CCS and TCSP Programs by Automata and Petri Nets.* Springer-Verlag (*LNCS* 369), 1989.

[Thi87] P.S. Thiagarajan. Elementary net systems. In *Advances in Petri Nets 1986, part I*, pages 26–59. Springer-Verlag (*LNCS* 254), 1987.

[Thi94] P.S. Thiagarajan. A trace based extension of linear time temporal logic. In *Proceedings, Symposium on Logic in Computer Science.* IEEE, 1994.

[Val88] Antti Valmari. Error detection by reduced reachability graph generation. In *Proc. Ninth European Workshop on Application and Theory of Petri Nets (Venice, Italy)*, pages 95–112, 1988.

[Val90] Antti Valmari. Stubborn sets for reduced state space generation. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1990*, pages 491–515. Springer-Verlag (*LNCS* 483), 1990.

[vG90] R.J. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions.* PhD thesis, CWI Amsterdam, 1990.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings, Symposium on Logic in Computer Science*, pages 332–344, june 1986.

[WG93] Pierre Wolper and Patrice Godefroid. Partial-order methods for temporal verification. Technical report, Université de Liège, Institut Montefiore, August 1993. Wolper-1, Hand-outs at Summerschool in Logical Methods In Concurrency Aarhus'93, To appear in: Concur'93 Proceedings.

[Win80] Glynn Winskel. *Events in Computation.* PhD thesis, University of Edinburgh, 1980.

[Win86] Glynn Winskel. Event structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, pages 325–390. Springer-Verlag (*LNCS* 255), 1986.

[Win87] Glynn Winskel. Petri nets, algebras, morphisms and compositionality. *Information and Computation*, 72(3):197–238, 1987.

[WN94] Glynn Winskel and Mogens Nielsen. Models for concurrency. Research Series RS-94-12, BRICS, Department of Computer Science, University of Aarhus, May 1994. 144 pp. To appear as a chapter in the *Handbook of Logic and the Foundations of Computer Science*, Oxford University Press.

[WN95] Glynn Winskel and Mogens Nielsen. *Models for Concurrency*, volume 4, chapter 1, pages 1–148. Oxford University Press, 1995. *eds.* S. Abramsky, D. M. Gabbay, and T. S. E. Gabbay.

[YO88] J. Young and P. O'Keefe. Experience with a type evaluator. In D. Bjørner, A. P. Ershov, and N. D. Jones, editors, *Partial Evaluation and Mixed Computation*, pages 573–581. North-Holland, 1988.

## Recent Publications in the BRICS Dissertation Series

**DS-96-3** **Lars Arge.** ***Efficient External-Memory Data Structures and Applications.*** **August 1996. Ph.D. thesis. xii+169 pp.**

**DS-96-2** **Allan Cheng.** ***Reasoning About Concurrent Computational Systems.*** **August 1996. Ph.D. thesis. xiv+229 pp.**

**DS-96-1** **Urban Engberg.** ***Reasoning in the Temporal Logic of Actions — The design and implementation of an interactive computer system.*** **August 1996. Ph.D. thesis. xvi+222 pp.**